PenPoint™ Application Programmatic Interface
Volume I
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.
PenPoint™

PenPoint™
API Reference

VOLUME I

GO CORPORATION

GO TECHNICAL LIBRARY

Addison-Wesley Publishing Company

Reading, Massachusetts • Menlo Park, California • New York
Don Mills, Ontario • Wokingham, England • Amsterdam
Bonn • Sydney • Singapore • Tokyo • Madrid • San Juan
Paris • Seoul • Milan • Mexico City • Taipei
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 PEPPOINT 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 0–201–60862–6
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 `msgListAddltem`, `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 1 / Class Manager

<table>
<thead>
<tr>
<th>File</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHMGR.H</td>
<td>357</td>
</tr>
<tr>
<td>CHOICE.H</td>
<td>359</td>
</tr>
<tr>
<td>CLAYOUT.H</td>
<td>359</td>
</tr>
<tr>
<td>CLOSEBOX.H</td>
<td>371</td>
</tr>
<tr>
<td>CMDBAR.H</td>
<td>373</td>
</tr>
<tr>
<td>CONTROL.H</td>
<td>375</td>
</tr>
<tr>
<td>COUNTER.H</td>
<td>383</td>
</tr>
<tr>
<td>FIELD.H</td>
<td>389</td>
</tr>
<tr>
<td>FONTLBOX.H</td>
<td>401</td>
</tr>
<tr>
<td>FRAME.H</td>
<td>405</td>
</tr>
<tr>
<td>GRABBOX.H</td>
<td>417</td>
</tr>
<tr>
<td>ICHOICE.H</td>
<td>423</td>
</tr>
<tr>
<td>ICON.H</td>
<td>425</td>
</tr>
<tr>
<td>ITABLE.H</td>
<td>431</td>
</tr>
<tr>
<td>ITOGGLE.H</td>
<td>433</td>
</tr>
<tr>
<td>LABEL.H</td>
<td>437</td>
</tr>
<tr>
<td>LISTBOX.H</td>
<td>451</td>
</tr>
<tr>
<td>MANAGER.H</td>
<td>461</td>
</tr>
<tr>
<td>MBUTTON.H</td>
<td>463</td>
</tr>
<tr>
<td>MFILTER.H</td>
<td>471</td>
</tr>
<tr>
<td>MENU.H</td>
<td>475</td>
</tr>
<tr>
<td>NOTE.H</td>
<td>481</td>
</tr>
<tr>
<td>OPTION.H</td>
<td>485</td>
</tr>
<tr>
<td>OPTTABLE.H</td>
<td>491</td>
</tr>
<tr>
<td>PAGENUM.H</td>
<td>513</td>
</tr>
<tr>
<td>POPUPCH.H</td>
<td>515</td>
</tr>
<tr>
<td>PROGRESS.H</td>
<td>517</td>
</tr>
<tr>
<td>SBAR.H</td>
<td>523</td>
</tr>
<tr>
<td>SELCHMGR.H</td>
<td>531</td>
</tr>
<tr>
<td>SHADOW.H</td>
<td>539</td>
</tr>
<tr>
<td>STDMSG.H</td>
<td>543</td>
</tr>
<tr>
<td>STRLBOX.H</td>
<td>547</td>
</tr>
<tr>
<td>SWIN.H</td>
<td>555</td>
</tr>
<tr>
<td>TABBAR.H</td>
<td>561</td>
</tr>
<tr>
<td>TAB.H</td>
<td>573</td>
</tr>
<tr>
<td>TBUTTON.H</td>
<td>579</td>
</tr>
<tr>
<td>TKFIELD.H</td>
<td>581</td>
</tr>
<tr>
<td>TTABLE.H</td>
<td>585</td>
</tr>
<tr>
<td>TLAYOUT.H</td>
<td>593</td>
</tr>
<tr>
<td>TRACK.H</td>
<td>601</td>
</tr>
<tr>
<td>TTABLE.H</td>
<td>611</td>
</tr>
</tbody>
</table>

### Part 2 / PenPoint Application Framework

<table>
<thead>
<tr>
<th>File</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP.H</td>
<td>77</td>
</tr>
<tr>
<td>APPDIR.H</td>
<td>79</td>
</tr>
<tr>
<td>APPMGR.H</td>
<td>111</td>
</tr>
<tr>
<td>APPMON.H</td>
<td>119</td>
</tr>
<tr>
<td>APPTAG.H</td>
<td>127</td>
</tr>
<tr>
<td>APPWIN.H</td>
<td>137</td>
</tr>
<tr>
<td>CBWIN.H</td>
<td>143</td>
</tr>
<tr>
<td>CLSPRN.H</td>
<td>149</td>
</tr>
<tr>
<td>EMBEDWIN.H</td>
<td>151</td>
</tr>
<tr>
<td>EWNEW.H</td>
<td>157</td>
</tr>
<tr>
<td>GOTO.H</td>
<td>173</td>
</tr>
<tr>
<td>ICONWIN.H</td>
<td>175</td>
</tr>
<tr>
<td>MARK.H</td>
<td>179</td>
</tr>
<tr>
<td>PRFRAME.H</td>
<td>183</td>
</tr>
<tr>
<td>PRINT.H</td>
<td>199</td>
</tr>
<tr>
<td>PRLAYOUT.H</td>
<td>203</td>
</tr>
<tr>
<td>PRMARGIN.H</td>
<td>213</td>
</tr>
<tr>
<td>RCAPPH</td>
<td>215</td>
</tr>
<tr>
<td>VIEW.H</td>
<td>217</td>
</tr>
<tr>
<td>MBUTTON.H</td>
<td>219</td>
</tr>
<tr>
<td>MCICON.H</td>
<td>231</td>
</tr>
<tr>
<td>MENU.H</td>
<td>233</td>
</tr>
<tr>
<td>MFILTER.H</td>
<td>241</td>
</tr>
<tr>
<td>NOTE.H</td>
<td>243</td>
</tr>
<tr>
<td>OPTION.H</td>
<td>253</td>
</tr>
<tr>
<td>OPTTABLE.H</td>
<td>257</td>
</tr>
<tr>
<td>PAGENUM.H</td>
<td>287</td>
</tr>
<tr>
<td>POPUPCH.H</td>
<td>293</td>
</tr>
<tr>
<td>PROGRESS.H</td>
<td>295</td>
</tr>
<tr>
<td>SBAR.H</td>
<td>325</td>
</tr>
<tr>
<td>SELCHMGR.H</td>
<td>327</td>
</tr>
<tr>
<td>SHADOW.H</td>
<td>345</td>
</tr>
<tr>
<td>STDMSG.H</td>
<td>347</td>
</tr>
<tr>
<td>STRLBOX.H</td>
<td>371</td>
</tr>
<tr>
<td>SWIN.H</td>
<td>373</td>
</tr>
<tr>
<td>TABBAR.H</td>
<td>375</td>
</tr>
<tr>
<td>TAB.H</td>
<td>383</td>
</tr>
<tr>
<td>TBUTTON.H</td>
<td>389</td>
</tr>
<tr>
<td>TKFIELD.H</td>
<td>401</td>
</tr>
<tr>
<td>TTABLE.H</td>
<td>405</td>
</tr>
<tr>
<td>TLAYOUT.H</td>
<td>417</td>
</tr>
<tr>
<td>TRACK.H</td>
<td>423</td>
</tr>
<tr>
<td>TTABLE.H</td>
<td>425</td>
</tr>
</tbody>
</table>

### Part 3 / Windows and Graphics

<table>
<thead>
<tr>
<th>File</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BITMAP.H</td>
<td>223</td>
</tr>
<tr>
<td>CCITT.H</td>
<td>225</td>
</tr>
<tr>
<td>GEO.H</td>
<td>229</td>
</tr>
<tr>
<td>PICSEG.H</td>
<td>233</td>
</tr>
<tr>
<td>SYSPONT.H</td>
<td>241</td>
</tr>
<tr>
<td>SYSGRAE.H</td>
<td>253</td>
</tr>
<tr>
<td>SBAR.H</td>
<td>257</td>
</tr>
<tr>
<td>SHADOW.H</td>
<td>287</td>
</tr>
<tr>
<td>STDMSG.H</td>
<td>293</td>
</tr>
<tr>
<td>STRLBOX.H</td>
<td>295</td>
</tr>
<tr>
<td>SWIN.H</td>
<td>325</td>
</tr>
<tr>
<td>TABBAR.H</td>
<td>327</td>
</tr>
<tr>
<td>TAB.H</td>
<td>345</td>
</tr>
<tr>
<td>TBUTTON.H</td>
<td>347</td>
</tr>
</tbody>
</table>

### Part 4 / UI Toolkit

<table>
<thead>
<tr>
<th>File</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BORDER.H</td>
<td>359</td>
</tr>
<tr>
<td>BUSY.H</td>
<td>359</td>
</tr>
<tr>
<td>BUTTON.H</td>
<td>359</td>
</tr>
</tbody>
</table>
Part 5 / Input and Handwriting

Translation

<table>
<thead>
<tr>
<th>File</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACETATE.H</td>
<td>625</td>
</tr>
<tr>
<td>ANIMSP.H</td>
<td>627</td>
</tr>
<tr>
<td>GWIN.H</td>
<td>631</td>
</tr>
<tr>
<td>HWCUSTOM.H</td>
<td>637</td>
</tr>
<tr>
<td>HWLETTER.H</td>
<td>655</td>
</tr>
<tr>
<td>INPUT.H</td>
<td>657</td>
</tr>
<tr>
<td>INSERT.H</td>
<td>659</td>
</tr>
<tr>
<td>KEY.H</td>
<td>671</td>
</tr>
<tr>
<td>KEYBOARD.H</td>
<td>689</td>
</tr>
<tr>
<td>KEYCAP.H</td>
<td>693</td>
</tr>
<tr>
<td>KEystate.H</td>
<td>697</td>
</tr>
<tr>
<td>PEN.H</td>
<td>701</td>
</tr>
<tr>
<td>SCRIBBLE.H</td>
<td>703</td>
</tr>
<tr>
<td>SPAPER.H</td>
<td>711</td>
</tr>
<tr>
<td>XGESTURE.H</td>
<td>719</td>
</tr>
<tr>
<td>XGATE.H</td>
<td>733</td>
</tr>
<tr>
<td>XLFILTER.H</td>
<td>737</td>
</tr>
<tr>
<td>XLIST.H</td>
<td>749</td>
</tr>
<tr>
<td>XSHAPE.H</td>
<td>751</td>
</tr>
<tr>
<td>XTEACH.H</td>
<td>761</td>
</tr>
<tr>
<td>XTEMPLT.H</td>
<td>769</td>
</tr>
<tr>
<td>XTEXT.H</td>
<td>773</td>
</tr>
<tr>
<td>XTRACT.H</td>
<td>779</td>
</tr>
<tr>
<td>XWORD.H</td>
<td>781</td>
</tr>
<tr>
<td>Index</td>
<td>785</td>
</tr>
</tbody>
</table>
Part 1 /
Class Manager
The Class Manager supports object-oriented programming.

clsObject inherits from null.

clsObject is the root of the Object System. It defines the basic capabilities of all objects.

clsClass inherits from clsObject.

clsClass is the root of all classes. clsClass provides class creation capabilities.

```c
#ifndef CLSMGR_INCLUDED
#define CLSMGR_INCLUDED
... // Definitions...
#endif
```

**Overview**

This file defines all the subroutines and messages that implement Object-oriented programing under PenPoint. The most important of these are:
- `ObjectCall()` and related routines, especially the Debugging Routines
- `MsgHandler()` and related macros
- `MakeMsg()` macro
- `clsClass, CLASS_NEW_DEFAULTS, etc.
- `clsObject, OBJECT_NEW_DEFAULTS, etc.
- `msgNew, msgNewDefaults, msgInit, msgDestroy, msgFree, msgSave, msgRestore

Look at the functions starting with ClsStsToString too.

This is one of PenPoint's key header files. Developers should browse through this file and be familiar with its contents. Other key header files are go.h, app.h, and win.h.

To fully understand what's going on here, you should read the Class Manager section of the PenPoint Architecture Reference.

**Guidelines**

Normally you should call your ancestor before processing a message. Possible exceptions include:
- messages that are defined by your class. Obviously, these shouldn't go to your ancestor at all.
- messages that you want to explicitly override. Depending on whether you want to override the message some of the time or all the time.
- `msgFreeOK, msgFreeing, msgFree` should use `objCallAncestorAfter`.
- protocols that requires the subclass to act on the message before the ancestor receives it.
Debugging Flags

The ClsMgr debugging flag set is 'C'. Defined values are:

000001  Show calls to ObjectCall().
000002  Show calls to ObjectCallAncestor().
000004  Show calls to ObjectSend().
000008  Show calls to ObjectPost().
000010  Show indirect calls (class messages) for traced objects.
000020  Show object new and free calls.
000040  Show observer related actions: add, remove, notify and post.
000080  Show messages as they are dispatched.
000100  Show objects as they are saved and restored.
000200  Gather ObjectCall depth statistics.
000400  Show objects as they are scavenged at task termination.
000800  Enter Debugger(), if bad object is passed to ObjectCall().
001000  Show calls to ObjectCallAncestor() for traced objects.
002000  Enable detailed messages from ObjectValid(). These messages are not necessarily errors if the client code handles stsBadObject. Because null objects are common they are not reported under C2000.
004000  Enable miscellaneous error/warning messages: Bad newStruct, Message not understood, WKN already exists, WKN replaced (warning).
008000  Enter the debugger after printing a warning.

Temporary flags:

010000  Fills the stack w/F0's before calling a method. This is useful for catching uninitialized variables.
020000  Show calls to extended ObjectCall() and ObjectCallAncestor().

Implementor Flags:

100000  Show all clsmgr statuses, legitimate errors are included.

#ifdef GO_INCLUDED
#include <go.h>
#endif
#ifdef OSTYPES_INCLUDED
#include <ostypes.h>
#endif
#ifdef UID_INCLUDED
#include <uid.h>
#endif
Status Values

#define stsBadObject
#define stsBadAncestor
#define stsBadContext
#define stsProtectionViolation
#define stsScopeViolation
#define stsTaskTerminated
#define stsSizeLimit
#define stsBadPropTag
#define stsNewStructError
#define stsClassHasReferences
#define stsNotUnderstood
#define stsVetoed
#define stsWellKnownExists
#define stsBadMethodTable

Non-Error Status Values

stsMessageIgnored is equal to stsRequestForward for historical reasons. MakeWarning is used to force the entry into the symbols DB.

#define stsMessageIgnored
#define stsAlreadyAdded
#define stsAlreadyRemoved
#define stsSendTaskInvalid
#define stsWellKnownReplaced
#define stsTraceOn
#define stsTraceOff

Object Capabilities

ifndef M_I86
Enum32 (OBJ_CAPABILITY)
{
    objCapOwner = flag1,
    objCapFree = flag2,
    objCapSend = flag3,
    objCapObservable = flag4,
    objCapInherit = flag6,
    objCapScavenge = flag7,
    objCapCreate = flag8,
    objCapProp = flag9,
    objCapMutate = flag10,
    objCapCall = flag15,
    objCapCreateNotify = flag16,
    objCapUnprotected = flag17,
    objCapNonSwappable = flag18
};
#else
typedef U32 OBJ_CAPABILITY;
#endif
### Types Derived Directly from Base Types

OBJECT, TAG, STATUS, etc. are defined in `<go.h>`

typedef OBJECT CLASS, *P_CLASS;
typedef TAG MESSAGE, *P_MESSAGE;
typedef P_UNKNOWN P_ARGS, *PP_ARGS;
typedef P_UNKNOWN CONTEXT, *P_CONTEXT;
typedef P_UNKNOWN P_IDATA, *PP_IDATA;
typedef U32 OBJ_KEY, *P_OBJ_KEY;
#define objWKNKey ((OBJ_KEY) 0)
typedef const U32 *P_MSG, **PP_MSG;

### Constants and Types Derived from Structs

NewArgs used to create an object.

typedef struct OBJECT_NEW {
    U32 newStructVersion; // Out: [msgNew::defaults] Validate msgNew
    OBJ_KEY key; // In: [msgNew] Lock for the object
    OBJECT uid; // In: [msgNew] Well-known uid
    OBJ_CAPABILITY cap; // Out: [msgNew] Dynamic or Well-known uid
    CLASS objClass; // In: [msgObjectNew] Class of instance
    OS_HEAP_ID heap; // Out: [msgNew::defaults] Heap to use for additional storage. If capCall then OSProcessSharedHeap else OSProcessHeap
    U32 spare1; // Unused (reserved)
    U32 spare2; // Unused (reserved)
} OBJECT_NEW_ONLY, OBJECT_NEW, *P_OBJECT_NEWONLY, *P_OBJECT_NEW;

New defaults fields for subclassing OBJECT.

#define objectNewFields OBJECT_NEWONLY object;

Fields for initializing a class.

typedef struct CLASS_NEW ONLY {
    P_MSG pMsg; // In: Can be pNul1 for abstract class
    CLASS ancestor; // In: Ancestor to inherit behavior from
    SIZEOF size; // In: Size of instance data, can be 0
    // (see comment below)
    SIZEOF newArgsSize; // In: Size of XX_NEW struct, can be 0
    // Value limited to U16
    U32 spare1; // Unused (reserved)
} CLASS_NEW ONLY, *P_CLASS_NEW ONLY;

Limits on instance data size:

Instance data for any class is limited to 64K bytes. Instance data for an entire objects is limited to 64K of protected data. Unprotected instance data is limited to 64K bytes per class but there is no limit for the object.

New defaults fields for subclassing CLASS.

#define classNewFields objectNewFields CLASS_NEWONLY cls;

NewArgs used to create a class.

typedef struct CLASS_NEW {
    classNewFields
} CLASS_NEW, *P_CLASS_NEW;
Enable/Disable capabilities

typedef struct OBJ_CAPABILITY_SET {
    OBJ_CAPABILITY     cap; // In: Capabilities to enable/disable
    OBJ_KEY             key; // In: Unlocks object, e.g., objWKNKey
} OBJ_CAPABILITY_SET, *P_OBJ_CAPABILITY_SET;

Set/Get owner

typedef struct OBJ_OWNER {
    OS_TASK_ID          task; // In: [msgSetOwner] New owner
                        // Out: [msgObjectOwner] Current owner
    OBJECT             object; // In: [msgObjectOwner] Source object
    OBJ_KEY            key; // In: [msgSetOwner] If required by caps
} OBJ_OWNER, *P_OBJ_OWNER;

Set/Get properties

typedef struct OBJ_PROP {
    TAG                    propId; // In: [msgProp] Name of property
    P_IDATA               pData; // In: [msgProp] Pointer to data
                        // In: [msgSetProp] Data to copy to prop
    SIZEOF               length; // In: [msgProp] # of bytes to copy
                        // Out: [msgProp] Length of data in bytes
                        // In: [msgSetProp] # of bytes to write
    OBJ_KEY              key; // In: [msgSetProp] If required by cap
} OBJ_PROP, *P_OBJ_PROP;

Add/Get observers

typedef struct OBJ_OBSERVER_POS {
    OBJECT              observer; // In: [msgAddObserverAt] New observer
                        // Out: [msgGetObserver] Observer at pos
    U16                 position; // In: Position in observer list
} OBJ_OBSERVER_POS, *P_OBJ_OBSERVER_POS;

Notify observers

typedef struct OBJ_NOTIFY_OBSERVERS {
    MESSAGE             msg; // In: Message to send/post observers
    P_ARGS              pArgs; // In: Args for message
    SIZEOF             lenSend; // In: Length of Args
} OBJ_NOTIFY_OBSERVERS, *P_OBJ_NOTIFY_OBSERVERS;

Buffer to hold symbol string. Used with ClsStsToString, etc.

#define clsSymBufSize 80
typedef char       P_CLS_SYMBUF[clsSymBufSize];

Array entry for OBJECT in the symbols database.

typedef struct CLS_SYM_OBJ {
    OBJECT              obj;
    P_STRING            name;
} CLS_SYM_OBJ, *P_CLS_SYM_OBJ, *PP_CLS_SYM_OBJ;

Array entry for message in symbols database.

typedef struct CLS_SYM_MSG {
    MESSAGE             msg;
    P_STRING            name;
} CLS_SYM_MSG, *P_CLS_SYM_MSG, *PP_CLS_SYM_MSG;

Array entry for STATUS in symbols database.

typedef struct CLS_SYM_STS {
    STATUS              sts;
    P_STRING            name;
} CLS_SYM_STS, *P_CLS_SYM_STS, *PP_CLS_SYM_STS;
Types Required for msgSave and msgRestore

Resource IDs

typedef TAG RES_ID, *P_RES_ID; // Resource ID

System flags for save and restore.

Enum16(RES_SAVE_RESTORE_FLAGS) {
    resDoingCopy = flag0 // Creating a copy of object
};

typedef struct OBJ_SAVE {
    OBJECT file; // In: File to save object to
    RES_ID resId; // In: Resource Id of root-level object
    OBJECT root; // In: Uid of root-level object
    P_UNKNOWN pEnv; // In: Environment to be saved
    U16 minSysVersion; // In/Out: Min acceptable system version
    U16 minAppVersion; // In/Out: Min acceptable app version
    RES_SAVE_RESTORE_FLAGS sysFlags; // In: System flags
    U16 appFlags; // In: App flags
    P UNKNOWN pFile; // In: StdIO FILE* bound to file above
    U32 spare1; // Unused (reserved)
    U32 spare2; // Unused (reserved)
} OBJ_SAVE, *P_OBJ_SAVE;

typedef struct OBJ_RESTORE {
    OBJECT_NEW object; // In: New defaults for restored object
    OBJECT file; // In: File to restore object from
    RES_ID resId; // In: Resource Id of root-level object
    OBJECT root; // In: Uid of root-level object
    P UNKNOWN pEnv; // In: Saved environment
    U16 sysVersion; // In: Sys version number of filed data
    U16 appVersion; // In: App version number of filed data
    RES_SAVE_RESTORE_FLAGS sysFlags; // In: System flags
    U16 appFlags; // In: App flags
    P UNKNOWN pFile; // In: StdIO FILE* bound to file above
    U32 spare1; // Unused (reserved)
    U32 spare2; // Unused (reserved)
} OBJ_RESTORE, *P_OBJ_RESTORE;

Method Definition Macros

Definition of a pointer to a method.

#define _HIGC_

Function Prototype typedef CDECL STATUS (* P_MSG_HANDLER) (#endif
#else
Function Prototype typedef STATUS (CDECL * P_MSG_HANDLER) (#endif
    MESSAGE msg,
    OBJECT self,
    P_ARGS pArgs,
    CONTEXT ctx,
    P_IDATA pData
    );
#endif

Definition of a method.

#define MSG_HANDLER STATUS CDECL

Shorthand used to declare a method.

#define MsgHandler(fn) MSG_HANDLER MsgHandler Primitive(fn, P_ARGS, P_IDATA)
Shorthand used to declare a method with \texttt{pArgs} cast to appropriate type. Note: \texttt{pArgsType} must be a pointer type.
\begin{verbatim}
#define MsgHandlerArgType(fn, pArgsType) \n    MSG_HANDLER MsgHandlerPrimitive(fn, pArgsType, P_IDATA)
\end{verbatim}

Shorthand used to declare a method with casts for \texttt{pArgs} and instance data. Note: \texttt{pArgsType} and \texttt{pInstData} must be pointer types.
\begin{verbatim}
#define MsgHandlerWithTypes(fn, pArgsType, pInstData) \n    MSG_HANDLER MsgHandlerPrimitive(fn, pArgsType, pInstData)
\end{verbatim}

Shorthand used to declare a method. Very fast and very dangerous. DS is NOT loaded. Don't use strings, local functions, statics, etc.
\begin{verbatim}
#define MsgHandlerRingCHelper(fn) \n    STATUS CDECL MsgHandlerPrimitive(fn, P_ARGS, P_IDATA)
\end{verbatim}

Shorthand used to declare a method.
\begin{verbatim}
#define MsgHandlerPrimitive(fn, pArgsType, pInstData) fn(\n    const MESSAGE msg, \n    const OBJECT self, \n    const pArgsType pArgs, \n    const CONTEXT ctx, \n    const pInstData pData)
\end{verbatim}

Cast \texttt{pData} to the appropriate type.
\begin{verbatim}
#define IDataPtr(pData, type) ((type*)pData)
\end{verbatim}

Copy protected instance data block into local storage.
\begin{verbatim}
#define IDataDeref(pData, type) (*(type*)pData)
\end{verbatim}

Shorthand used to ignored any unused parameters in a method.
\begin{verbatim}
#define MsgHandlerParametersNoWarning \n    Unused(msg); Unused(self); Unused(pArgs); Unused(ctx); Unused(pData)
\end{verbatim}

\section*{Message Macros}

Message numbers are between 0 and 254, inclusive. Message number 255
\begin{verbatim}
#define MakeMsg(wkn,msg) MakeTag(wkn,msg)
\end{verbatim}

Extract the message portion of a message.
\begin{verbatim}
#define MsgNum(msg) TagNum(msg)
\end{verbatim}

The WKNValue unique represents a class.
\begin{verbatim}
#define ClsNum(msg) WKNValue(msg)
\end{verbatim}

Messages defined with MsgNoError() will not generate a msgNotUnderstood error if they reach clsObject. Instead, stsMessageIgnored is returned.
\begin{verbatim}
#define MsgNoError(msg) ((msg)|msgNoErrorFlag)
#define msgNoErrorFlag (1l<<21)
\end{verbatim}

Messages that are handled as class messages have this flag added to the message value.
\begin{verbatim}
#define msgClassMessageFlag (1l<<22)
\end{verbatim}

Compare two messages for equality.
\begin{verbatim}
#define MsgEqual(m1,m2) (m1==m2)
\end{verbatim}
## Object Scope Macros
(Well-Known and Dynamic)

```c
#define ObjectIsDynamic(o) ((U32)(o)&objDynamicFlag)
#define ObjectIsWellKnown(o) (!ObjectIsDynamic(o))
#define ObjectIsGlobal(o) ObjectIsWellKnown(o)
#define ObjectIsLocal(o) (!ObjectIsGlobal(o))
#define ObjectIsGlobalWKN(o) (ObjectIsWKN(o) && WKNScope(o)==wknGlobal)
#define ObjectIsProcessGlobalWKN(o) (ObjectIsWKN(o) && WKNScope(o)==wknProcessGlobal)
#define ObjectIsPrivateWKN(o) (ObjectIsWKN(o) && WKNScope(o)==wknPrivate)
```

All dynamic objects have this bit set in their UID.

```c
#define objDynamicFlag 0x800000
```

## Messages

```c
// Recycle:
// Next available: 120
```

**msgNull**

Not a real message, just a place holder.

Takes pNull, returns STATUS.

```c
#define msgNull MakeMsg(objNull, 0)
```

**msgNewDefaults**

Initializes new struct to default values.

Takes new struct for object being created, returns STATUS. Category: class message.

```c
#define msgNewDefaults MakeMsg(clsObject, 2)
```

**msgNew**

Creates an object and sends msgInit to the new object.

Takes new struct for object being created, returns STATUS. Category: class message.

```c
#define msgNew MakeMsg(clsObject, 4)
```

**Return Value**

- **stsNewStructError** The new struct was not properly initialized, it was used more than once, or it was overwritten.
- **stsBadParam** Format of well-known UID was invalid.
- **stsWellKnownExists** Well-known UID has already been created with a different key.
- **stsOSOutOfMem** Too many objects have been created or system memory is exhausted.
- **stsProtectionViolation** (clsClass) objCapInherit is disabled.
- **stsSizeLimit** (clsClass) More than the maximum amount of instance data has been requested.
- **stsBadAncestor** (clsClass) Ancestor is not a class.
msgNewWithDefaults
Creates an object with default values.
Takes new struct for object being created, returns STATUS. Category: class message.
#define msgNewWithDefaults MakeMsg(clsObject, 5)
Comments
Self sends msgNewDefaults followed by msgNew. Useful when changes to the new struct are NOT required.

msgInit
Sent to the object immediately after it is created.
Takes new struct for object being created, returns STATUS.
#define msgInit MakeMsg(clsObject, 6)
Comments
When msgInit reach clsObject the capabilities and the key in the newArgs are set for the object. This means that, unlike most messages, developers must call their ancestor AFTER processing this one.

msgCreated
Sent to the object after it is fully created, i.e., after msgInit.
Takes new struct for object being created, returns STATUS.
#define msgCreated MsgNoError(MakeMsg(clsObject, 46))
Comments
This message is only sent if objCapCreateNotify is enabled.

msgDestroy
Destroys the object.
Takes OBJ_KEY, returns STATUS.
#define msgDestroy MakeMsg(clsObject, 28)
Comments
When msgDestroy is sent to the object, clsObject sends msgFreeOK, msgFreeing and msgFree to self. msgFreePending is sent to the observers. Only clsObject should handle msgDestroy. (That is, like msgNew, developers send msgDestroy but never handle it.)

Return Value
stsProtectionViolation objCapFree is disabled and the key does not open the object.
stsClassHasReferences (clsClass) Instances of the class object still exists. Only returned when the object being destroyed is a class.

msgFreeOK
Sent as the first of three messages to destroy the object.
Takes OBJ_KEY, returns STATUS.
#define msgFreeOK MsgNoError(MakeMsg(clsObject, 14))
Comments
There is no point in handling this message unless you have some reason to refuse to be freed, in which case return stsVetoed. Note that if the process that owns the object or the class of the object is destroyed, the object will be destroyed too, regardless of what it does with msgFreeOK. This is mainly useful for immortal system objects.

See Also
msgDestroy
Return Value

stsClassHasReferences (clsClass) Instances of the class object still exists. Only returned when the object being destroyed is a class.

---

**msgFreeing**

Sent as the second of three messages to destroy the object.

Takes OBJ_KEY, returns STATUS.

```c
#define msgFreeing MsgNoError (MakeMsg(clsObject, 90))
```

**Comments**

Most developers never handle this message either. If an object is part of a tangled web of other objects, all of which are supposed to be freed whenever any of them is freed, it's possible to get a loop where two objects respond to msgFree by trying to free each other. The first object that receives msgFreeing should extract itself from any other object that might try to free it. When it receives msgFree, it can then safely send msgDestroy to those other objects.

**See Also**

msgDestroy

---

**msgFree**

Sent as the last of three messages to destroy the object.

Takes OBJ_KEY, returns STATUS.

```c
#define msgFree MakeMsg(clsObject, 8)
```

**Comments**

msgFree must succeed and error status should never be returned. Any validation should be done during msgFreeOK. (Like msgInit, developers handle this message but never send it.)

**See Also**

msgDestroy

---

**msgFreePending**

Sent to observers immediately before the object is freed.

Takes OBJECT, returns STATUS. Category: observer notification.

```c
#define msgFreePending MsgNoError (MakeMsg(clsObject, 70))
```

**Comments**

If an observer cares about the final state of the object, this is the last opportunity to send it a message.

**See Also**

msgDestroy

---

**msgRestoreInstance**

Creates and restores an object from an object file.

Takes P_OBJ_RESTORE, returns STATUS.

```c
#define msgRestoreInstance MakeMsg(clsObject, 80)
```

**Message Arguments**

```c
typedef struct OBJ_RESTORE {
    OBJECT_NEW object; // In: New defaults for restored object
    OBJECT file; // In: File to restore object from
    RES_ID resId; // In: Resource Id of root-level object
    OBJECT root; // In: Uid of root-level object
    PUNKNOWN pEnv; // In: Saved environment
    U16 sysVersion; // In: Sys version number of filed data
    U16 appVersion; // In: App version number of filed data
    RES_SAVE_RESTORE_FLAGS sysFlags; // In: System flags
    U16 appFlags; // In: App flags
    PUNKNOWN pFile; // In: StdIO FILE* bound to file above
    U32 spare1; // Unused (reserved)
    U32 spare2; // Unused (reserved)
} OBJ_RESTORE, * P_OBJ_RESTORE;
```
msgRestore

Creates and restores an object from an object file.

Takes P_OBJ_RESTORE, returns STATUS.

#define msgRestore MakeMsg(clsObject, 10)

typedef struct OBJ_RESTORE {
    OBJECT NEW object;    // In: New defaults for restored object
    OBJECT file;          // In: File to restore object from
    RES_ID resid;         // In: Resource Id of root-level object
    OBJECT root;          // In: Uid of root-level object
    P UNKNOWN pEnv;       // In: Saved environment
    U16 sysVersion;       // In: Sys version number of filed data
    U16 appVersion;       // In: App version number of filed data
    RES_SAVE_RESTORE_FLAGS sysFlags;  // In: System flags
    U16 appFlags;         // In: App flags
    P UNKNOWN pFile;      // In: StdIO FILE* bound to file above
    U32 spare1;           // Unused (reserved)
    U32 spare2;           // Unused (reserved)
} OBJ_RESTORE, * P_OBJ_RESTORE;

msgRestoreMsgTable

Returns the message table for the class.

Takes PP_MSG, returns STATUS.

#define msgRestoreMsgTable MakeMsg(clsObject, 116)

msgSave

Causes the object to file itself in an object file.

Takes P_OBJ_SAVE, returns STATUS.

#define msgSave MakeMsg(clsObject, 12)

typedef struct OBJ_SAVE {
    OBJECT file;          // In: File to save object to
    RES_ID resid;         // In: Resource Id of root-level object
    OBJECT root;          // In: Uid of root-level object
    P UNKNOWN pEnv;       // In: Environment to be saved
    U16 minSysVersion;    // In/Out: Min acceptable system version
    U16 minAppVersion;    // In/Out: Min acceptable app version
    RES_SAVE_RESTORE_FLAGS sysFlags;  // In: System flags
    U16 appFlags;         // In: App flags
    P UNKNOWN pFile;      // In: StdIO FILE* bound to file above
    U32 spare1;           // Unused (reserved)
    U32 spare2;           // Unused (reserved)
} OBJ_SAVE, * P_OBJ_SAVE;

clsObject files the capabilities of the object and any property that has tag flag1 set. For example:
#define MY_PROP MakeTagWithFlags(clsFoo, tagNum, 1)

Return Value

stsRequestNotSupported (clsClass) Classes not do file.

---

**msgCopy**

Passes back a copy of the object.

Takes P_OBJ_COPY, returns STATUS.

```c
#define msgCopy MakeMsg(clsObject, 54)
```

**Arguments**

```c
typedef struct OBJ_COPY {
    OBJECT requestor;  // In: Object to receive msgCopyRestore
    OBJECT object;     // Out: UID of copied object
    U32 reserved[4];   // Reserved.
} OBJ_COPY, * P_OBJ_COPY;
```

**Comments**

This message will pass back a copy of the object receiving the message. This object will be created by opening a temporary resource file, sending msgSave to the object, and then sending msgCopyRestore to the passed in requestor object. It will then close and destroy the temporary file. Note that the requestor object could be in a different task from the object receiving this message. In this situation, the copy of the object will exist in new task.

Return Value

stsFailed Could not open temporary resource file.

**See Also**

msgCopyRestore

---

**msgCopyRestore**

Restores the passed in object.

Takes P_OBJ_COPY_RESTORE, returns STATUS.

```c
#define msgCopyRestore MakeMsg(clsObject, 56)
```

**Arguments**

```c
typedef struct OBJ_FS_LOCATOR {
    OBJECT uid;
    P_STRING pPath;
} OBJ_FS_LOCATOR;
```

```c
typedef struct OBJ_COPY_RESTORE {
    OBJ_FS_LOCATOR locator;  // In: File locator that the object is in
    RES_ID resId;            // In: Resource id of filed object
    OBJECT object;           // Out: Uid of object to return
    U32 reserved[4];         // Reserved.
} OBJ_COPY_RESTORE, * P_OBJ_COPY_RESTORE;
```

**Comments**

This message is sent to the object with an object resource Id, and a file locator (a resource file). This will result in msgRestore being sent to the appropriate object to read in the resource object. Sent to the requestor object when performing a msgCopy.

**See Also**

msgCopy

---

**msgDump**

Causes each ancestor to print interesting debugging information.

Takes $32, returns STATUS.

```c
#define msgDump MakeMsg(clsObject, 52)
```

**Comments**

Each class should implement a msgHandler for msgDump. The msgHandler should print out interesting information for the object.
The parameter to \texttt{msgDump} is used to determine how much information to print.

Suggested values for \texttt{pArgs}:

0  Implementer's choice. Print whatever information is most useful.
1  Terse. One line only.
-1  Terse including embedded objects. One line of information plus one line for each embedded object, e.g., a menu would display information about each menu item.

\texttt{maxS32} Verbose. All possible information about the object.

\texttt{minS32} Verbose including embedded objects. The maximum amount of information.

\texttt{other}  All other values are implementation dependent.

If the value of the parameter is in between two defined values the action should be based on the smaller value.

Suggested format:

"\texttt{msgDump(yourClassName): yourDebuggingInformation}"

c\texttt{lsObject} defines \texttt{pArgs} as:

0  The object's capabilities and internal address.
1  Same as 0.
2  Same as 1 plus owner, number of observers, number of properties, the size of instance data and size of property list. \texttt{maxS32}: Same as 2 plus hex dump of instance data.
-1  Same as 0 plus \texttt{msgDump} to observers. ([Not implemented])
-2  Same as -1 plus owner, number of observers, number of properties, the size of instance data. ([Not implemented])

\texttt{minS32}  Same as -2 plus hex dump of instance data. ([Not implemented])

c\texttt{lsclass} defines \texttt{pArgs} as:

0  The class capabilities, size of data for instances, the number of instances and subclasses of the class.
1  Same as 0.
2  Same as 1 plus ancestor and \texttt{newArgs} size.

\texttt{maxS32}  Same as 2. ([Not implemented])

\textbf{msgException}

Sent to observers of \texttt{theProcess}, an object within each process, when an exception occurs within that process.

Takes \texttt{P\_OBJ\_EXCEPTION}, returns \texttt{STATUS}. Category: observer notification.

\begin{verbatim}
#define msgException    MsgNoError(MakeMsg(clsObject, 100))
\end{verbatim}

\begin{verbatim}
typedef struct OBJ_EXCEPTION {
    OS_TASK_ERROR  errorCode;  // In: Type of exception
    OS_TASK_ID     task;       // In: Task that received the exception
    U32            spare;     // Unused (reserved)
} OBJ_EXCEPTION, *P_OBJ_EXCEPTION;
\end{verbatim}

\textbf{Arguments}

If a subtask is being terminated only objects owned by the subtask are notified.
**msgTaskTerminated**

Sent to observers of the `Process`, an object within each process, after the task is terminated.

Takes `P_OBJ_EXCEPTION`, returns STATUS. Category: observer notification.

```c
#define msgTaskTerminated MsgNoError(MakeMsg(clsObject, 112))
```

**Message**

typedef struct OBJ_EXCEPTION {
  OS_TASK_ERROR errorCode;  // In: Type of exception
  OS_TASK_ID task;         // In: Task that received the exception
  U32 spare;               // Unused (reserved)
} OBJ_EXCEPTION, *P_OBJ_EXCEPTION;

**Arguments**

**msgScavenge**

Sent to the object when a class has `objCapScavenge` set and the object’s task is being terminated by request or because of an error.

Takes `OS_TASK_ERROR`, returns STATUS. Category: descendant responsibility.

```c
#define msgScavenge MsgNoError(MakeMsg(clsObject, 102))
```

**Comments**

This message will only be executed by class that set `objCapScavenge`. Do not pass this message to your ancestor.

**msgScavenged**

Sent to the observers AFTER the object has been scavenged.

Takes `OS_TASK_ERROR`, returns STATUS. Category: observer notification.

```c
#define msgScavenged MsgNoError(MakeMsg(clsObject, 104))
```

**msgFreeSubTask**

Sent to the `Process` to free a subtask.

Takes `P_SUBTASK_FREE`, returns STATUS.

```c
#define msgFreeSubTask MsgNoError(MakeMsg(clsObject, 104))
```

**Arguments**

typedef struct OBJ_SUBTASK_FREE {
  OS_TASK_ID task;         // In: Task to be terminated
  OS_TASK_ERROR exitCode;  // In: Exit code for task termination
} OBJ_SUBTASK_FREE, *P_OBJ_SUBTASK_FREE;

**Comments**

Useful for delayed termination when message is posted to the `Process`.

**Return Value**

`stsOSStrInvalidOperationForTask`  Task was not a subtask of this process.

**msgHeap**

Returns the preferred heap to use when allocating storage for this object.

Takes `P_O Given

```c
#define msgHeap MakeMsg(clsObject, 96)
```
**msgCan**

Checks the object's capabilities.

Takes OBJ_CAPABILITY, returns STATUS.

```c
#define msgCan MakeMsg(clsObject, 36)
```

<table>
<thead>
<tr>
<th>Message</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>enum32 (OBJ_CAPABILITY)</td>
<td>// default for: OBJECT CLASS</td>
</tr>
<tr>
<td></td>
<td>// ---------------------------</td>
</tr>
<tr>
<td>objCapOwner</td>
<td>= flag1, // TRUE FALSE</td>
</tr>
<tr>
<td>objCapFree</td>
<td>= flag2, // TRUE FALSE</td>
</tr>
<tr>
<td>objCapSend</td>
<td>= flag3, // TRUE FALSE</td>
</tr>
<tr>
<td>objCapObservable</td>
<td>= flag4, // TRUE</td>
</tr>
<tr>
<td>objCapInherit</td>
<td>= flag6, // n/a TRUE</td>
</tr>
<tr>
<td>objCapScavenge</td>
<td>= flag7, // enable only: n/a FALSE</td>
</tr>
<tr>
<td>objCapCreate</td>
<td>= flag8, // FALSE FALSE</td>
</tr>
<tr>
<td>objCapProp</td>
<td>= flag9, // TRUE TRUE</td>
</tr>
<tr>
<td>objCapMutate</td>
<td>= flag10, // TRUE TRUE</td>
</tr>
<tr>
<td>objCapCall</td>
<td>= flag11, // FALSE TRUE</td>
</tr>
<tr>
<td>objCapCreateNotify</td>
<td>= flag15, // create only: FALSE FALSE</td>
</tr>
<tr>
<td>objCapUnprotected</td>
<td>= flag16, // create only: n/a FALSE</td>
</tr>
<tr>
<td>objCapNonSwappable</td>
<td>= flag18, // create only: FALSE</td>
</tr>
</tbody>
</table>

};

If the capabilities in the parameter are all enabled, msgCan returns stsOK otherwise stsProtectionViolation is returned.

**Return Value**

stsProtectionViolation Capability disabled.

**msgDisable**

Disables some or all of the object's capabilities.

Takes P_OBJ_CAPABILITY_SET, returns STATUS.

```c
#define msgDisable MakeMsg(clsObject, 16)
```

<table>
<thead>
<tr>
<th>Message</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>typedef struct Obj_CAPABILITY_SET {</td>
<td></td>
</tr>
<tr>
<td>OBJ_CAPABILITY cap; // In: Capabilities to enable/disable</td>
<td></td>
</tr>
<tr>
<td>OBJ_KEY key; // In: Unlocks object, e.g., objWKNKey</td>
<td></td>
</tr>
<tr>
<td>} OBJ_CAPABILITY_SET, * P_OBJ_CAPABILITY_SET;</td>
<td></td>
</tr>
</tbody>
</table>

**Return Value**

stsProtectionViolation Key does not open the object.

**msgEnable**

Enables some or all of the object's capabilities.

Takes P_OBJ_CAPABILITY_SET, returns STATUS.

```c
#define msgEnable MakeMsg(clsObject, 18)
```

<table>
<thead>
<tr>
<th>Message</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>typedef struct Obj_CAPABILITY_SET {</td>
<td></td>
</tr>
<tr>
<td>OBJ_CAPABILITY cap; // In: Capabilities to enable/disable</td>
<td></td>
</tr>
<tr>
<td>OBJ_KEY key; // In: Unlocks object, e.g., objWKNKey</td>
<td></td>
</tr>
<tr>
<td>} OBJ_CAPABILITY_SET, * P_OBJ_CAPABILITY_SET;</td>
<td></td>
</tr>
</tbody>
</table>

**Return Value**

stsProtectionViolation Key does not open the object.
msgIsA
Tests if the object's class inherits from the class.
Takes CLASS, returns STATUS.
#define msgIsA MakeMsg(clsObject, 30)

Return Value
stsOK   Class is an ancestor of the object's class.
stsBadAncestor   Class is not an ancestor of the object's class.

msgAncestorIsA
Tests if self inherits from the class.
Takes CLASS, returns STATUS.
#define msgAncestorIsA MakeMsg(clsObject, 32)

Comments
This is a clsClass message and can only be sent to a class. Consider using msgIsA if the object is not a class.

Return Value
stsOK   Class parameter is an ancestor.
stsBadObject   Class parameter is not an object.
stsBadAncestor   Class parameter is not an ancestor.

msgClass
Passes back the class of the object.
Takes P_CLASS, returns STATUS.
#define msgClass MakeMsg(clsObject, 34)

msgAncestor
Passes back the ancestor of the class.
Takes P_CLASS, returns STATUS.
#define msgAncestor MakeMsg(clsObject, 20)

Comments
This is a clsClass message and can only be sent to a class. Consider using msgClass if the object is not a class.

msgSetLock
Sets or changes the key of the object.
Takes OBJ_LOCK_SET, returns STATUS.
#define msgSetLock MakeMsg(clsObject, 106)

Arguments
typedef struct OBJ_LOCK_SET {
   OBJ_KEY oldKey;   // In: Required to set lock
   OBJ_KEY newKey;   // In: New key, if successful
} OBJ_LOCK_SET, * P_OBJ_LOCK_SET;

Return Value
stsProtectionViolation   Old key does not open the object.
**msgUnlocks**
Tests if a key will unlock the object.
Takes OBJ_KEY, returns STATUS.

```c
#define msgUnlocks MakeMsg(clsObject, 38)
```

Return Value
stsProtectionViolation Key does not open the object.

**msgDuplicateLock**
Locks the pArgs object with the same key as object.
Takes OBJECT, returns STATUS.

```c
#define msgDuplicateLock MakeMsg(clsObject, 40)
```

Return Value
stsBadObject Parameter is not an object.

**msgVersion**
Returns the version of the object.
Takes pNull, returns STATUS.

```c
#define msgVersion MakeMsg(clsObject, 82)
```

Return Value
stsScopeViolation Object was dynamic, request is nonsense.

**msgNewArgsSize**
Returns the size of the new struct required to create an instance of this class.
Takes pNull, returns STATUS.

```c
#define msgNewArgsSize MakeMsg(clsObject, 92)
```

Comments
This is a clsClass message and can only be sent to a class.

**msgOwner**
Passes back the task that owns this object.
Takes P_OS_TASK_ID, returns STATUS.

```c
#define msgOwner MakeMsg(clsObject, 22)
```

**msgSetOwner**
Changes the owner task.
Takes P_OBJ_OWNER, returns STATUS.

```c
#define msgSetOwner MakeMsg(clsObject, 24)
```

Message
```c
typedef struct OBJ_OWNER {
    OS_TASK_ID task; // In: [msgSetOwner] New owner
    OBJECT object; // Out: [msgObjectOwner] Current owner
    OBJ_KEY key; // In: [msgObjectOwner] Source object
    OBJ_OWNER, * P_OBJ_OWNER;
} OBJ_OWNER;
```

Return Value
stsProtectionViolation Key does not open the object.
**msgProp**

Passes back the value of a property for the object.

Takes P_OBJ_PROP, returns STATUS.

```c
#define msgProp MakeMsg(clsObject, 108)
```

<table>
<thead>
<tr>
<th>Message</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>objProp</td>
<td>Passes back the value of a property for the object.</td>
</tr>
<tr>
<td>takes</td>
<td>P_OBJ_PROP, returns STATUS.</td>
</tr>
</tbody>
</table>

```c
typedef struct OBJ_PROP {
  TAG propId; // [msgProp] Name of property
  P_IDATA pData; // [msgProp] Pointer to data
  SIZEOF length; // [msgProp] Data to copy to prop
} OBJ_PROP, *P_OBJ_PROP;
```

**Return Value**

- **stsBadPropTag** Tag value was not in the proper range.

**msgSetProp**

Sets a property on the object.

Takes P_OBJ_PROP, returns STATUS.

```c
#define msgSetProp MakeMsg(clsObject, 110)
```

<table>
<thead>
<tr>
<th>Message</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>objSetProp</td>
<td>Sets a property on the object.</td>
</tr>
<tr>
<td>takes</td>
<td>P_OBJ_PROP, returns STATUS.</td>
</tr>
</tbody>
</table>

```c
typedef struct OBJ_PROP {
  TAG propId; // [msgProp] Name of property
  P_IDATA pData; // [msgProp] Pointer to data
  SIZEOF length; // [msgProp] Data to copy to prop
  OBJ_KEY key; // [msgProp] # of bytes to copy
} OBJ_PROP, *P_OBJ_PROP;
```

**Return Value**

- **stsBadPropTag** Tag value was not in the proper range.
- **stsProtectionViolation** Key does not open the object.

**msgObjectXXX**

These msgObjectXXX messages can be used with ObjectCall() to get information about all objects, regardless of their task. Functionally they are equivalent to msgXXX, when applicable.

**msgObjectIsA**

Using the object and the class in the pArgs. Tests if the object's class inherits from the class.

Takes P_OBJ_IS_A, returns STATUS.

```c
#define msgObjectIsA MakeMsg(clsObject, 84)
```

<table>
<thead>
<tr>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>object:</td>
</tr>
<tr>
<td>takes</td>
</tr>
</tbody>
</table>

```c
typedef struct OBJ_IS_A {
  OBJECT object; // In: Source object
  CLASS objClass; // In: Ancestor of source object's class
} OBJ_IS_A, *P_OBJ_IS_A;
```

**Return Value**

- **stsBadObject** Parameter is not an object.
- **stsBadAncestor** Class is not an ancestor of the object's class.
### msgObjectAncestorIsA

Tests if the descendant class inherits from the ancestor.

Takes P_OBJ_ANCESTOR_IS_A, returns STATUS.

```c
#define msgObjectAncestorIsA MakeMsg(clsObject, 86)
```

**Arguments**

typedef struct OBJ_ANCESTOR_IS_A {
    CLASS descendant;  // In: Source class (always a class)
    CLASS ancestor;    // In: Ancestor of the descendant
} OBJ_ANCESTOR_IS_A, * P_OBJ_ANCESTOR_IS_A;

**Comments**

This is a clsClass message and can only be sent to a class.

**Return Value**

- **stsBadObject**  One of the parameters is not a class.
- **stsBadAncestor**  Ancestor parameter is not an ancestor.

### msgObjectClass

Passes back the class for the object in pArgs.

Takes P_OBJ_CLASS, returns STATUS.

```c
#define msgObjectClass MakeMsg(clsObject, 88)
```

**Arguments**

typedef struct OBJ_CLASS {
    OBJECT object;  // In: Source object
    CLASS objClass; // Out: Class of source object
} OBJ_CLASS, * P_OBJ_CLASS;

**Return Value**

- **stsBadObject**  Object or class parameters are not objects.

### msgObjectOwner

Passes back the owning task for the object in pArgs.

Takes P_OBJ_OWNER, returns STATUS.

```c
#define msgObjectOwner MakeMsg(clsObject, 26)
```

**Message**

```c
typedef struct OBJ_OWNER {
    OS_TASK_ID task;         // In: [msgSetOwner] New owner
    OBJECT object;           // Out: [msgObjectOwner] Current owner
    OBJ_KEY key;             // In: [msgObjectOwner] Source object
} OBJ_OWNER, * P_OBJ_OWNER;
```

**Arguments**

```c
#define msgObjectOwner MakeMsg(clsObject, 26)
```

**Return Value**

- **stsBadObject**  Parameter is not an object.

### msgObjectValid

Tests that the object in pArgs exists.

Takes OBJECT, returns STATUS.

```c
#define msgObjectValid MakeMsg(clsObject, 42)
```

**Return Value**

- **stsBadObject**  Parameter is not an object.
- **stsBadAncestor**  Invalid ancestor.
**msgObjectVersion**
Returns the version of the object in pArgs.
Takes OBJECT, returns STATUS.

```c
#define msgObjectVersion MakeMsg(clsObject, 44)
```

**Return Value**
- **stsBadObject** Parameter is not an object.
- **stsScopeViolation** Parameter was dynamic, request is nonsense.

**msgObjectNew**
Creates a new object in the same context as the object that receives this message.
Takes newArgs, returns STATUS.

```c
#define msgObjectNew MakeMsg(clsObject, 98)
```

**Return Value**
- **stsProtectionViolation** objCapCreate is disabled.
- **stsScopeViolation** Must be executed in the owner task of the receiving object.

**msgTrace**
Turn tracing on for classes and objects. Return value is **stsTraceOn** if tracing was on and **stsTraceOff** if tracing was off.
Takes TAG, returns STATUS.

```c
#define msgTrace MakeMsg(clsObject, 48)
define objTraceOn (P_ARGS)stsTraceOn
define objTraceOff (P_ARGS)stsTraceOff
```

**Comments**
When tracing is turned on for the object, every ObjectCall() to the object causes a 3-line message to be printed. The format of the output is:

```
C> Trace ObjectCall: @ cls="ancestor name" task="task"
C> object="object name" depth="D"
C> msg="message name", pArgs="address", pData="address"
```

On return from the ObjectCall() a 2-line message is printed. The format of the output is:

```
C> Trace ObjectCall: returns="status value" task="task"
C> object="object name" depth="D/C"
```

where task is the task id in hex, depth is the number of recursive dispatch loops. All names are printed symbolically when symbols are available.

ObjectCallAncestor() calls are traced for objects if tracing is on for the object and the debug flag /DC1000 is set.

When tracing is turned on for a class, the class is traced as an object. In addition, all ObjectCallAncestor() calls that pass through the class are traced.
**msgMutate**

Changes the ancestor of the object to be the `newAncestor` class.

Takes `P_OBJ_MUTATE`, returns `STATUS`.

```c
#define msgMutate MakeMsg(clsObject, 46)
```

**Arguments**

```c
typedef struct OBJ_MUTATE {
    CLASS newClass;    // In: Object's new class
    OBJ_KEY key;       // In: If required by caps
} OBJ_MUTATE, *P_OBJ_MUTATE;
```

**Comments**

The total size of the instance data for the new and old ancestors must be equal, this is the sum for all the ancestors up to `clsObject`. This message is **NOT** intended for general use. Use it when the behavior of an existing object needs to be overridden.

**Return Value**

- `stsBadAncestor` The `newAncestor` class is not a valid class.
- `stsSizeLimit` The sizes of new and old instance data don't match.

---

**msgAddObserver**

Adds an observer to the end of the object's observer list.

Takes `OBJECT`, returns `STATUS`.

```c
#define msgAddObserver MakeMsg(clsObject, 58)
```

**Return Value**

- `stsBadObject` Parameter is not an object.
- `stsProtectionViolation` `objCapObservable` is disabled.
- `stsScopeViolation` Observer is local and has a different owner than the observed object or the observed object is callable.
- `stsAlreadyAdded` The same observer has been added twice. This is only a warning, the observers are ref counted. Two adds require two removes.

---

**msgAddObserverAt**

Adds an observer at the specified position in the observer list.

Takes `P_OBJ_OBSERVER_POS`, returns `STATUS`.

```c
#define msgAddObserverAt MakeMsg(clsObject, 78)
```

**Arguments**

```c
typedef struct OBJ_OBSERVER_POS {
    OBJECT observer;  // In: [msgAddObserverAt] New observer
    U16 position;     // In: Position in observer list
} OBJ_OBSERVER_POS, *P_OBJ_OBSERVER_POS;
```

**Return Value**

- `stsBadObject` Parameter is not an object.
- `stsProtectionViolation` `objCapObservable` is disabled.
- `stsScopeViolation` Observer is local and has a different owner than the observed object or the observed object is callable.
- `stsAlreadyAdded` The same observer has been added twice. This is only a warning, the observers are ref counted. Two adds require two removes.
msgRemoveObserver
Removes an observer from the object’s observer list.
Takes OBJECT, returns STATUS.
#define msgRemoveObserver MakeMsg(clsObject, 60)

msgRemoved is sent to the observer after it is removed.
stsProtectionViolation objCapObservable disabled.
stsAlreadyRemoved Observer was not on the list.

msgNotifyObservers
Sends a message to the observers.
Takes P_OBJ_NOTIFY_OBSERVERS, returns STATUS.
#define msgNotifyObservers MakeMsg(clsObject, 62)

typedef struct OBJ_NOTIFY_OBSERVERS {
    MESSAGE msg;          // In: Message to send/post observers
    P_ARGS pArgs;         // In: Args for message
    SIZEOF lenSend;       // In: Length of Args
} OBJ_NOTIFY_OBSERVERS, *P_OBJ_NOTIFY_OBSERVERS;

Any observer that returns stsBadObject is removed from the observer list.

msgPostObservers
Posts a message to the observers.
Takes P_OBJ_NOTIFY_OBSERVERS, returns STATUS.
#define msgPostObservers MakeMsg(clsObject, 94)

typedef struct OBJ_NOTIFY_OBSERVERS {
    MESSAGE msg;          // In: Message to send/post observers
    P_ARGS pArgs;         // In: Args for message
    SIZEOF lenSend;       // In: Length of Args
} OBJ_NOTIFY_OBSERVERS, *P_OBJ_NOTIFY_OBSERVERS;

Any observer that returns stsBadObject is removed from the observer list.

msgEnumObservers
Passes back the observer list.
Takes P_OBJ_ENUM_OBSERVERS, returns STATUS.
#define msgEnumObservers MakeMsg(clsObject, 64)

typedef struct OBJ_ENUM_OBSERVERS {
    U16 max;             // In: Size of pObservers[].
    U16 count;           // In: # to pass back in pObservers[].
    // If count > max memory may be allocated
    // Out: # of valid entries in pObservers[].
    P_OBJECT pObservers; // In: ptr to array
    // Out: If memory was allocated
    // client should free the memory
    U16 next;            // In: Set to 0 for the first call
    // Out: Next available entry
} OBJ_ENUM_OBSERVERS, *P_OBJ_ENUM_OBSERVERS;

stsEndOfFile The size of the array is greater than or equal to the number of observer.
msgGetObserver
Passes back the observer at the specified position in the observer list.
Takes P_OBJ_OBSERVER_POS, returns STATUS.
#define msgGetObserver MakeMsg(clsObject, 74)

<table>
<thead>
<tr>
<th>Message</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>typedef struct OBJ_OBSERVER_POS {</td>
<td>In: [msgAddObserverAt] New observer</td>
</tr>
<tr>
<td>OBJECT observer;</td>
<td>Out: [msgGetObserver] Observer at pos</td>
</tr>
<tr>
<td>U16 position;</td>
<td>In: Position in observer list</td>
</tr>
<tr>
<td>OBJ_OBSERVER_POS, * P_OBJ_OBSERVER_POS;</td>
<td></td>
</tr>
</tbody>
</table>

Comments
objNull is returned if the position is not in the observer list.

msgNumObservers
Passes back the number of observers for this object.
Takes P_U16, returns STATUS.
#define msgNumObservers MakeMsg(clsObject, 72)

msgAdded
Sent to the observer when it is added to an object's observer list.
Takes OBJECT, returns STATUS. Category: observer notification.
#define msgAdded MsgNoError(MakeMsg(clsObject, 66))

msgRemoved
Sent to the observer when it is removed from an object's observer list.
Takes OBJECT, returns STATUS. Category: observer notification.
#define msgRemoved MsgNoError(MakeMsg(clsObject, 68))

msgChanged
Generic message that can be used to notify observers that a change has occurred.
Takes OBJECT, returns STATUS. Category: observer notification.
#define msgChanged MsgNoError(MakeMsg(clsObject, 76))

msgNotUnderstood
Sent by clsObject when an unrecognized message is received.
Takes P_MSG_NOT_UNDERSTOOD, returns STATUS.
#define msgNotUnderstood MakeMsg(clsObject, 50)

Arguments
typedef struct MSG_NOT_UNDERSTOOD {
 MESSAGE msg; // In: Message not understood
 PARGS pArgs; // In: Args of message
} MSG_NOT_UNDERSTOOD, * P_MSG_NOT_UNDERSTOOD;

Return Value
stsNotUnderstood Always returned by clsObject when this message reaches clsObject.
Message wild cards

Used to define a class wild card and as a table wild card.

#define objWildCard -1
Wild card for clsObject.

#define clsObjWildCard MakeMsg(clsObject, objWildCard)
Wild card for clsClass.

#define clsClsWildCard MakeMsg(clsClass, objWildCard)

Functions

ObjectCall
Maps the message to the object's method (MsgHandler) and calls it with pArgs.
Returns STATUS.

Function Prototype
STATUS EXPORTED ObjectCall(
    MESSAGE msg,
    OBJECT object,
    PARGS pArgs
);

Return Value
stsBadObject Object was invalid.
stsScopeViolation Object owned by a different task and does not have objCapCall set.

ObjectCallAncestorCtx
Calls the next ancestor in the class chain.
Returns STATUS.

Function Prototype
STATUS EXPORTED ObjectCallAncestorCtx(
    CONTEXT ctx
);

Comments
Developers usually can avoid calling this explicitly by specifying objCallAncestorBefore or (for a few messages) objCallAncestorAfter in the method table. Occasionally, you need to call your ancestor in the middle of things, and this is the call you do it with.

See Also
ObjectCallAncestor

Return Value
stsBadContext if ctx parameter is bad.

ObjectCallAncestor
Calls the ancestor with the parameters supplied.
Returns STATUS.

Function Prototype
STATUS EXPORTED ObjectCallAncestor(
    MESSAGE msg,
    OBJECT self,
    PARGS pArgs,
    CONTEXT ctx
);

Comments
In general you should use ObjectCallAncestorCtx().

Return Value
stsBadContext if ctx parameter is bad.
ObjectSend
Generalized version of ObjectCall() that works across tasks boundaries.
Returns STATUS.

Function Prototype

```c
STATUS EXPORTED ObjectSend(
    MESSAGE msg,
    OBJECT object,
    P_ARGS pArgs, // In only: Not updated
    SIZEOF lenArgs
);
```

Comments
The pArgs block is copied into the address space of the task that owns the object and an ObjectCall() is executed in that task's context. If lenArgs equals 0, pArgs block is not copied and the pointer is passed directly. In this case, pArgs must point to global storage.

While the current task is waiting for ObjectSend() to return, the task will continue to dispatch messages sent to objects owned by the task. This allows sending to an object in another task, which in turns sends to an object owned by the current task, without deadlock.

Return Value
stsProtectionViolation objCapSend is disabled.
stsSendTaskInvalid Object's owning task is invalid.
stsTaskTerminated While waiting for a reply the object's task died.

ObjectSendUpdate
Same as ObjectSend(), additionally the pArgs block is copied back to the current task.
Returns STATUS.

Function Prototype

```c
STATUS EXPORTED ObjectSendUpdate(
    MESSAGE msg,
    OBJECT object,
    P_ARGS pArgs, // In/Out: Updated
    SIZEOF lenArgs
);
```

ObjectSendU32
Same as ObjectSend() without the length arg, lenArgs = 0.
Returns STATUS.

Function Prototype

```c
STATUS EXPORTED ObjectSendU32(
    MESSAGE msg,
    OBJECT object,
    P_ARGS pArgs // In only: Not updated
);
```

ObjectSendTask
Same as ObjectSend() except the task is specified explicitly.
Returns STATUS.

Function Prototype

```c
STATUS EXPORTED ObjectSendTask(
    MESSAGE msg,
    OBJECT object,
    P_ARGS pArgs, // In only: Not updated
    SIZEOF lenArgs,
    OS_TASK_ID task
);
```
Comments

For experts only: Use this routine with care, the task of the object is ignored. ObjectSendTask() allows sending to well-known process-globals from outside the process, such as, theProcess. You might use this to communicate with theUndoManager in an embedded application.

ObjectSendUpdateTask

Same as ObjectSendTask(), additionally the pArgs are updated.

Returns STATUS.

Function Prototype

STATUS EXPORTED ObjectSendUpdateTask(
  MESSAGE msg,
  OBJECT object,
  P_ARGS pArgs,  // In/Out: Updated
  SIZEOF lenArgs,
  OS_TASK_ID task
);

Comments

Experts only, use this routine with care.

ObjectPost

Posts a message to the object via the system input queue.

Returns STATUS.

Function Prototype

STATUS EXPORTED ObjectPost(
  MESSAGE msg,
  OBJECT object,
  P_ARGS pArgs,
  SIZEOF lenArgs
);

Comments

ObjectPost() is similar to ObjectSend() but the message delivery is deferred and the current task continues to run. Because the current task does not wait, it is not possible to return a status value or pArgs.

The most common use of ObjectPost() is to delay the effect of a msgDestroy. For example, if a button sends you a message when it is pressed, and you want to destroy the button at that point, you cannot use ObjectCall() to send msgDestroy to it until after you have returned from processing the message the button sent. If you ObjectPost() the msgDestroy, this guarantees the button won't receive it until you have returned.

ObjectPost() is synchronized with respect to the input system. A posted message is placed in the system input queue. When the message reaches the head of the queue it is sent to the object in the context of the task that owns the object. A posted message is typically dispatched by a task’s top-level dispatch loop. If the task is already processing a message or waiting for a reply to a sent message the posted message is queued. The one exception is when the input system is running system modal, in this case the posted messages are delivered to any dispatch loop. Dispatch loops are created whenever an ObjectSend() is waiting for a reply. The side effect is that any task that is running concurrently may receive a posted message at any time.
**ObjectPostU32**

Same as `ObjectPost()` without the length arg, `lenArgs = 0`.

Returns STATUS.

**Function Prototype**

```c
STATUS EXPORTED ObjectPostU32(
    MESSAGE msg,
    OBJECT object,
    P ARGS pArgs
);
```

**ObjectPostTask**

Same as `ObjectPost()` except the task is specified explicitly.

Returns STATUS.

**Function Prototype**

```c
STATUS EXPORTED ObjectPostTask(
    MESSAGE msg,
    OBJECT object,
    P ARGS pArgs,
    SIZEOF lenArgs,
    OS_TASK_ID task
);
```

**Comments**

For experts only: Use this routine with care, the owning task of the object is ignored. `ObjectPostTask()` allows posting to WKN process-globals from outside the process, such as, `theProcess`.

**ObjectPostAsync**

Similar to `ObjectPost()` but not synchronized with the input system.

Returns STATUS.

**Function Prototype**

```c
STATUS EXPORTED ObjectPostAsync(
    MESSAGE msg,
    OBJECT object,
    P ARGS pArgs,
    SIZEOF lenArgs
);
```

**Comments**

This call causes concurrency and all the difficulties associated with it.

One of these difficulties, described in detail under `ObjectPost`, is the handling of posted messages when the input system is running system modal.

**ObjectPostAsyncTask**

Same as `ObjectPostAsync()` except the task is specified explicitly.

Returns STATUS.

**Function Prototype**

```c
STATUS EXPORTED ObjectPostAsyncTask(
    MESSAGE msg,
    OBJECT object,
    P ARGS pArgs,
    SIZEOF lenArgs,
    OS_TASK_ID task
);
```

**Comments**

This call causes concurrency and all the difficulties associated with it.
For experts only: Use this routine with care, the owning task of the object is ignored. ObjectPostAsyncTask() allows posting to WKN process-globals from outside the process, such as, theProcess.

**ObjectPostDirect**

Similar to ObjectPostAsync() but can be dispatched by any dispatch loop.

Returns STATUS.

```c
Function Prototype
STATUS EXPORTED ObjectPostDirect(
    MESSAGE msg,
    OBJECT object,
    P_ARGS pArgs,
    SIZEOF lenArgs
);
```

Comments

This call causes concurrency and all the difficulties associated with it.

One of these difficulties, described in detail under ObjectPost, is the handling of posted messages when the input system is running system modal.

**ObjectPostDirectTask**

Same as ObjectPostDirect() except the task is specified explicitly.

Returns STATUS.

```c
Function Prototype
STATUS EXPORTED ObjectPostDirectTask(
    MESSAGE msg,
    OBJECT object,
    P_ARGS pArgs,
    SIZEOF lenArgs,
    OS_TASK_ID task
);
```

Comments

This call causes concurrency and all the difficulties associated with it.

For experts only: Use this routine with care, the owning task of the object is ignored. ObjectPostDirectTask() allows posting to WKN process-globals from outside the process, such as, theProcess.

**ObjectWrite**

Writes the instance data for self in a protected area.

Returns STATUS.

```c
Function Prototype
STATUS EXPORTED0 ObjectWrite(
    OBJECT self,
    CONTEXT ctx,
    P_UNKNOWN pData
);
Return Value
stsBadContext Invalid context.
ObjectWritePartial
Updates part of the instance data for self in a protected area.

Returns STATUS.

Function Prototype: STATUS EXPORTED ObjectWritePartial(
  OBJECT self,
  CONTEXT ctx,
  P_UNKNOWN pData,
  SIZEOF offset,
  SIZEOF length
);

Return Value: stsBadContext Invalid context.

ObjectRead
Copies the instance data from protected storage into pBuf.

Returns STATUS.

Function Prototype: STATUS EXPORTED ObjectRead(
  OBJECT self,
  CONTEXT ctx,
  P UNKNOWN pBuf
);

Comments: The pData pointer passed into the MsgHandler is a faster way to read the protected data.

ObjectPoke
Writes the object's instance data.

Returns STATUS.

Function Prototype: STATUS EXPORTED ObjectPoke(
  OBJECT object,
  P MSG classMsgTable,     // Address of the class's table
  OBJ KEY key,             // Key for the class
  P UNKNOWN pBuf
);

Comments: Copies pBuf into the instance data block for the class specified.

Return Value: stsBadAncestor ClassMsgTable did not correspond to an ancestor.
stsProtectionViolation Key does not open the object.

ObjectPeek
Reads the object's instance data.

Returns STATUS.

Function Prototype: STATUS EXPORTED ObjectPeek(
  OBJECT object,
  P MSG classMsgTable,
  OBJ KEY key,
  P UNKNOWN pBuf
);

Comments: Copies the instance data block for the class specified into pBuf.

Return Value: stsBadAncestor ClassMsgTable did not correspond to an ancestor.
stsProtectionViolation Key does not open the object.
ObjectOwner

Returns the object's owner.
Returns STATUS.

Function Prototype: OS_TASK_ID EXPORTED ObjectOwner(
  OBJECT  object
);

ObjectValid

Returns stsOK if the object is validate, otherwise an error is returned.
Returns STATUS.

Function Prototype: STATUS EXPORTED ObjectValid(
  OBJECT  object
);

Default MsgHandlers

Default MsgHandler that always returns stsOK.

Function Prototype: MsgHandler (StsOKMsgHandler);

Default MsgHandler that always returns stsFailed.

Function Prototype: MsgHandler (StsFailedMsgHandler);

Default MsgHandler that always returns stsReqNotSupported.

Function Prototype: MsgHandler (StsReqNotSupportedMsgHandler);

Default MsgHandler that always returns stsNotYetImplemented.

Function Prototype: MsgHandler (StsNotYetImplemented);

Default MsgHandler that always returns stsMessageIgnored.

Function Prototype: MsgHandler (StsMessageIgnoredMsgHandler);

Functions for Generating Symbolic Names

These routines are very useful for debugging. It is MUCH more useful to be able to print "stsBadParameter" instead of some 32-bit hex number.

ClsStsToString

Takes a STATUS and returns its symbolic name or [wkn=num:sts=num].
Returns P_STRING.

Function Prototype: P_STRING EXPORTED ClsStsToString(
  STATUS  sts,
  P_STRING  pStr
);

Comments: Returns either an internal pointer to a symbolic name or the pArgs buffer. If a symbolic name is not found, a string [wkn=num:sts=num] is constructed in the pArgs buffer.
Symbolic names are added via ClsMgrSymbolsInit().
ClsMsgToString
Takes a message and returns its symbolic name or [wkn=num:msg=num].
Returns P_STRING.

Function Prototype
P_STRING EXPORTED ClsMsgToString(
    MESSAGE    msg,
    P_STRING   pStr
);

Comments
Returns either an internal pointer to a symbolic name or the pArgs buffer. If a symbolic name is not found, a string [wkn=num:msg=num] is constructed in the pArgs buffer.
Symbolic names are added via ClsMgrSymbolsInit().

ClsTagToString
Takes a message and returns its symbolic name or [wkn=num:tag=num].
Returns P_STRING.

Function Prototype
P_STRING EXPORTED ClsTagToString(
    TAG        tag,
    P_STRING   pStr
);

Comments
Returns either an internal pointer to a symbolic name or the pArgs buffer. If a symbolic name is not found, a string [wkn=num:tag=num] is constructed in the pArgs buffer.
Currently, TAGs and MSGs are kept in the same list. If a TAG and MSG have the same value then first one found will be displayed. This may change in the future.
Symbolic names are added via ClsMgrSymbolsInit().

ClsObjToString
Takes an OBJECT and returns its symbolic name or [type:num:num].
Returns P_STRING.

Function Prototype
P_STRING EXPORTED ClsObjToString(
    OBJECT    object,
    P_STRING  pStr
);

Comments
Returns either an internal pointer to a symbolic name or the pArgs buffer. If a symbolic name is not found, a string [type=num:num] is constructed in the pArgs buffer.
Symbolic names are added via ClsMgrSymbolsInit().

ObjectInfoString
Takes an OBJECT and returns its symbolic name and additional information.
Returns P_STRING.

Function Prototype
P_STRING EXPORTED ObjectInfoString(
    OBJECT    object,
    P_STRING  pStr
);

Comments
Formats is the first if the name is found, and the second if not:
name (cls=name or [type=num:num])
[type=num:num] (cls=name or [type=num:num])
Return Value

stsBadObject Parameter is not an object.

**ClsStringToSts**

Takes a symbolic name as a string and returns the corresponding STATUS.

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED CIsStringToSts(
    P_STRING  sts
)
```

**ClsStringToMsg**

Takes a symbolic name as a string and returns the corresponding message.

Returns MESSAGE.

Function Prototype

```c
MESSAGE EXPORTED CIsStringToMsg(
    P_STRING  msg
)
```

**ClsStringToTag**

Takes a symbolic name as a string and returns the corresponding tag.

Returns TAG.

Function Prototype

```c
MESSAGE EXPORTED CIsStringToTag(
    P_STRING  tag
)
```

Comments

Currently, TAGs and MSGs are kept in the same list. If a TAG and MSG have the same value then first one found will be displayed. This may change in the future.

**ClsStringToObj**

Takes a symbolic name as a string and returns the corresponding OBJECT.

Returns OBJECT.

Function Prototype

```c
OBJECT EXPORTED CIsStringToObj(
    P_STRING  object
)
```

**ClsSymbolsInit**

Adds three arrays of symbolic names (OBJECT, MSG, STATUS) to the database.

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED0 CIsSymbolsInit(
    P_STRING       type,
    P_CLS_SYM_OBJ  objSymbols,
    P_CLS_SYM_MSG  msgSymbols,
    P_CLS_SYM_STS  stsSymbols
)
```

Comments

Each group of arrays is labelled with a tag. If two groups have the same tag, the last group to be added replaces the earlier group. The arrays must be in shared, user visible memory.

Return Value

stsBadParam symbols were not in shared, user visible memory
Low-Level Task Dispatch Routines

ObjectMsgLoop
Receives and dispatches object messages forever.
Returns STATUS.

#define ObjectMsgLoop() ObjectMsgDispatch(pNull)

Comments
If you create a sub-task with OSSubTaskCreate(), and you want that subtask to be able to receive messages, then you have to make it call this routine. ObjectMsgLoop() never returns. It just sits there waiting for messages generated by input events or sent from other processes and calling the appropriate local message handler for each one in turn. Even if you never use this directly, knowing that it exists makes it much easier to understand the difference between ObjectCall, ObjectPost, and ObjectSend.

Return Value
stsBadParam  Bad ITMSG_INFO parameter.

ObjectMsgDispatch
Dispatches object message received by OSITMsgReceive().
Returns STATUS.

Function Prototype
STATUS EXPORTED ObjectMsgDispatch(P_OS_ITMSG_INFO pITMsg);

Return Value
stsBadParam  Bad ITMSG_INFO parameter. ITMsg type must be one of osClsmgrSend or osClsmgrPost.

ObjectMsgDispatchInfo
Passes back information on the current ObjectMsgDispatch frame.
Returns STATUS.

Function Prototype
STATUS EXPORTED ObjectMsgDispatchInfo(
P_O S_ ITMSG_ INFO pinfo,
P_U32 plevel
);   // Out: ITMSG_INFO for requested frame
    // In/Out: requested frame
    // In: requested dispatch frame,
    // maxU32 = current, 1 = top level
    // Out: actual level of dispatch frame.

Enum32(SEND_TYPE)   // (.asm)
{
    objSendNoUpdate = flag0,
    objSendUpdate = flag1,
    objPostAsync = flag2,
    objPostDirect = flag3,
    objSendMax = flag10
};

Used by ObjectMsgExtract() and ObjectMsgAlter(). All fields are out parameters for ObjectMsgExtract and in parameters of ObjectMsgAlter. The token field is currently not used and not settable by ObjectMsgAlter.

typedef struct OBJ_DISPATCH_INFO {
    MESSAGE msg;
    OBJECT object;
    P_ARGS pArgs;
    U32 length;
    U32 token;
    SEND_TYPE type;
} OBJ_DISPATCH_INFO, *P_OBJ_DISPATCH_INFO;
**PENPOINT API REFERENCE**

**Part 1 / Class Manager**

Return Value

- **stsBadParam**  Bad ITMSG_INFO parameter.
- **stsFailed**  Not inside a dispatch loop or invalid frame number

---

**ObjectMsgExtract**

Extracts the interesting ObjectSend fields from the ITMsg packet.

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED ObjectMsgExtract(
    P_OS_ITMSG_INFO       pITMsg,
    P_OBJ_DISPATCH_INFO   pInfo
);
```

Return Value

- **stsBadParam**  Bad ITMSG_INFO parameter.

---

**ObjectMsgAlter**

Alters the ObjectSend fields of the ITMsg packet.

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED ObjectMsgAlter(
    P_OS_ITMSG_INFO       pITMsg,
    P_OBJ_DISPATCH_INFO   pInfo
);
```

---

These structs are used by the method compiler, outside of Penpoint.

```c
typedef struct MSG_INFO {
    MESSAGE       msg;
    P_U8           functionName;
    MSG_HANDLER_FLAGS    flags;
} MSG_INFO, * P_MSG_INFO;

typedef struct CLASS_INFO {
    P_U8            tableName;  // name to use for compiled table
    P_MSG_INFO      msgTable;   // message table to compile
    U32             flags;     // no flags, must be set to zero
} CLASS_INFO, * P_CLASS_INFO;
```

Return Value

- **stsBadParam**  Bad ITMSG_INFO parameter.


## Debugging Support

### ObjectCallNoDebug

Same as ObjectCall() but prevents tracing (i.e., no debug output for /DC1)

Returns STATUS.

**Function Prototype**  
```c
STATUS EXPORTED ObjectCallNoDebug(
    MESSAGE msg,
    OBJECT object,
    P_ARGS pArgs
);
```

```c
#define objMaxCallsDepth 10
```

```c
typedef struct OBJ_STATISTICS {
    U32 numObjReads;
    U32 numObjWrites;
    U32 numObjPeeks;
    U32 numObjPokes;
    U32 numObjCalls;
    U32 numObjSends;
    U32 numObjPosts;
    U32 depthObjCalls[objMaxCallsDepth];
    U32 numObjMaxDepth;
} OBJ_STATISTICS, *P_OBJ_STATISTICS;
```

### ClsClearStatistics

Zeros the statistics gathering counters.

Returns STATUS.

**Function Prototype**  
```c
STATUS EXPORTED ClsClearStatistics(void);
```

### ClsDumpStatistics

Prints the current value of the statistics.

Returns STATUS.

**Function Prototype**  
```c
STATUS EXPORTED ClsDumpStatistics(void);
```

### ClsStatistics

Passes back the current value of the statistics in stats parameter.

Returns STATUS.

**Function Prototype**  
```c
STATUS EXPORTED ClsStatistics(P_OBJ_STATISTICS stats);
```

### ClsSetStatistics

Resets the value of the statistics to stats parameter.

Returns STATUS.

**Function Prototype**  
```c
STATUS EXPORTED ClsSetStatistics(P_OBJ_STATISTICS stats);
```

**Comments**  
By calling ClsStatistics() at the beginning of a routine and ClsSetStatistics() at the end selected routines can be exempted from statistics gathering.
The debugging macros are short-hand for a call to the appropriate function followed by a conditional test and action. All the message passing functions have macros that: return if there is an error (Ret), jump to a label on an error (Jmp) and test for an error and return the value (OK). ObjectCall and ObjectCallAncestor have two additional macros, Failed and Chk.

Standard GO error recovery is done by using the Ret() form as long as there's nothing to clean up and then using the Jmp() form to jump to a label at the bottom of the routine that knows how to clean up. Note that both Ret() and Jmp() forms use Warn() forms of their respective calls, so any \text{sts} < \text{stsOK} generates an error message if DEBUG is set.

ObjectCall

\begin{verbatim}
#define ObjCallRet(m,o,p,s) \
   if (((s) = ObjCallWarn(m,o,p)) < \text{stsOK}) return s; else 
#define ObjCallJmp(m,o,p,s,x) \
   if (((s) = ObjCallWarn(m,o,p)) < \text{stsOK}) goto x; else 
#define ObjCallOK(m,o,p,s) \n   ((s = ObjCallWarn(m,o,p)) \text{ >= stsOK}) 
#define ObjCallFailed(m,o,p,s) ((s = ObjCallWarn(m,o,p)) < \text{stsOK}) 
#define ObjCallChk(m,o,p,s) ((s = ObjectCall(m,o,p)) < \text{stsOK})
\end{verbatim}

ObjectCallAncestor

\begin{verbatim}
#define ObjCallAncestorRet(m,o,p,c,s) \
   if (((s) = ObjCallAncestorWarn(m,o,p,c)) < \text{stsOK}) return s; else 
#define ObjCallAncestorJmp(m,o,p,c,s,x) \
   if (((s) = ObjCallAncestorWarn(m,o,p,c)) < \text{stsOK}) goto x; else 
#define ObjCallAncestorOK(m,o,p,c,s) \n   ((s = ObjCallAncestorWarn(m,o,p,c)) \text{ >= stsOK}) 
#define ObjCallAncestorFailed(m,o,p,c,s) ((s = ObjCallAncestorWarn(m,o,p,c)) < \text{stsOK}) 
#define ObjCallAncestorChk(m,o,p,c,s) ((s = ObjectCallAncestor(m,o,p,c)) < \text{stsOK})
\end{verbatim}

ObjectSend

\begin{verbatim}
#define ObjSendRet(m,o,p,l,s) \
   if (((s) = ObjSendWarn(m,o,p,l)) < \text{stsOK}) return s; else 
#define ObjSendJmp(m,o,p,l,s,x) \
   if (((s) = ObjSendWarn(m,o,p,l)) < \text{stsOK}) goto x; else 
#define ObjSendOK(m,o,p,l,s) ((s = ObjSendWarn(m,o,p,l)) \text{ >= stsOK})
\end{verbatim}

ObjectSendUpdate

\begin{verbatim}
#define ObjSendUpdateRet(m,o,p,l,s) \
   if (((s) = ObjSendUpdateWarn(m,o,p,l)) < \text{stsOK}) return s; else 
#define ObjSendUpdateJmp(m,o,p,l,s,x) \
   if (((s) = ObjSendUpdateWarn(m,o,p,l)) < \text{stsOK}) goto x; else 
#define ObjSendUpdateOK(m,o,p,l,s) ((s = ObjSendUpdateWarn(m,o,p,l)) \text{ >= stsOK})
\end{verbatim}

ObjectSendTask

\begin{verbatim}
#define ObjSendTaskRet(m,o,p,l,t,s) \
   if (((s) = ObjSendTaskWarn(m,o,p,l,t)) < \text{stsOK}) return s; else 
\end{verbatim}
#define ObjSendTaskJmp(m,o,p,l,t,x) 
    if (((s) = ObjSendTaskWarn(m,o,p,l,t)) < stsOK) goto x; else
#define ObjSendTaskOK(m,o,p,l,t,s) ((s = ObjSendTaskWarn(m,o,p,l,t)) >= stsOK)

ObjectSendUpdateTask

#define ObjSendUpdateTaskRet(m,o,p,l,t,s) 
    if (((s) = ObjSendUpdateTaskWarn(m,o,p,l,t)) < stsOK) return s; else
#define ObjSendUpdateTaskJmp(m,o,p,l,t,s,x) 
    if (((s) = ObjSendUpdateTaskWarn(m,o,p,l,t)) < stsOK) goto x; else
#define ObjSendUpdateTaskOK(m,o,p,l,t,s) ((s = ObjSendUpdateTaskWarn(m,o,p,l,t)) >= stsOK)

ObjectSendU32

#define ObjSendU32Ret(m,o,p,s) 
    if (((s) = ObjSendU32Warn(m,o,p)) < stsOK) return s; else
#define ObjSendU32Jmp(m,o,p,s,x) 
    if (((s) = ObjSendU32Warn(m,o,p)) < stsOK) goto x; else
#define ObjSendU32OK(m,o,p,s) ((s = ObjSendU32Warn(m,o,p)) >= stsOK)

ObjectPost

#define ObjPostRet(m,o,p,l,s) 
    if (((s) = ObjPostWarn(m,o,p,l)) < stsOK) return s; else
#define ObjPostJmp(m,o,p,l,s,x) 
    if (((s) = ObjPostWarn(m,o,p,l)) < stsOK) goto x; else
#define ObjPostOK(m,o,p,l,s) ((s = ObjPostWarn(m,o,p,l)) >= stsOK)

ObjectPostAsync

#define ObjPostAsyncRet(m,o,p,l,s) 
    if (((s) = ObjPostAsyncWarn(m,o,p,l)) < stsOK) return s; else
#define ObjPostAsyncJmp(m,o,p,l,s,x) 
    if (((s) = ObjPostAsyncWarn(m,o,p,l)) < stsOK) goto x; else
#define ObjPostAsyncOK(m,o,p,l,s) ((s = ObjPostAsyncWarn(m,o,p,l)) >= stsOK)

ObjectPostDirect

#define ObjPostDirectRet(m,o,p,l,s) 
    if (((s) = ObjPostDirectWarn(m,o,p,l)) < stsOK) return s; else
#define ObjPostDirectJmp(m,o,p,l,s,x) 
    if (((s) = ObjPostDirectWarn(m,o,p,l)) < stsOK) goto x; else
#define ObjPostDirectOK(m,o,p,l,s) ((s = ObjPostDirectWarn(m,o,p,l)) >= stsOK)

ObjectPostU32

#define ObjPostU32Ret(m,o,p,s) 
    if (((s) = ObjPostU32Warn(m,o,p)) < stsOK) return s; else
#define ObjPostU32Jmp(m,o,p,s,x) 
    if (((s) = ObjPostU32Warn(m,o,p)) < stsOK) goto x; else
#define ObjPostU32OK(m,o,p,s) ((s = ObjPostU32Warn(m,o,p)) >= stsOK)
# Debugging Helper Functions (with /DDEBUG)

```c
#if defined DEBUG || defined CLSMGR_COMPILE

ObjectCallWarning
Same as ObjectCall(), additionally prints a debugging message if status less than stsOK.
Returns STATUS.

Function Prototype
STATUS EXPORTED ObjectCallWarning(
    MESSAGE    msg,
    OBJECT     object,
    P_ARGS     pArgs,
    P_STRING   fn,
    U16         ln
);

Comments
In general, ObjCallWarn macro should be used to call this routine.

ObjectCallNoDebugWarning
Same as ObjectCallNoDebug(), additionally prints a debugging message if status less than stsOK.
Returns STATUS.

Function Prototype
STATUS EXPORTED ObjectCallNoDebugWarning(
    MESSAGE    msg,
    OBJECT     object,
    P_ARGS     pArgs,
    P_STRING   fn,
    U16         ln
);

Comments
In general, ObjCallNoDebugWarn macro should be used to call this routine.

ObjectCallAncestorCtxWarning
Same as ObjectCallAncestorCtx(), additionally prints a debugging message if status less than stsOK.
Returns STATUS.

Function Prototype
STATUS EXPORTED ObjectCallAncestorCtxWarning(
    CONTEXT    ctx,
    P_STRING   fn,
    U16         ln
);

Comments
In general, ObjCallAncestorCtxWarn macro should be used.

ObjectCallAncestorWarning
Same as ObjectCallAncestor(), additionally prints a debugging message if status less than stsOK.
Returns STATUS.

Function Prototype
STATUS EXPORTED ObjectCallAncestorWarning(
    MESSAGE    msg,
    OBJECT     object,
    P_ARGS     pArgs,
    CONTEXT    ctx,
    P_STRING   fn,
    U16         ln
);
```
In general, ObjCallAncestorWarn macro should be used.

**ObjectSendWarning**

Same as ObjectSend(), additionally prints a debugging message if status less than stsOK.

Returns STATUS.

Function Prototype

```
STATUS EXPORTED ObjectSendWarning(
    MESSAGE msg,
    OBJECT object,
    P_ARGS pArgs,
    SIZEOF lenArgs,
    P_STRING fn,
    UI6 ln
);
```

Comments

In general, ObjectSendWarn macro should be used.

**ObjectSendUpdateWarning**

Same as ObjectSendUpdate(), additionally prints a debugging message if status less than stsOK.

Returns STATUS.

Function Prototype

```
STATUS EXPORTED ObjectSendUpdateWarning(
    MESSAGE msg,
    OBJECT object,
    P_ARGS pArgs,
    SIZEOF lenArgs,
    P_STRING fn,
    UI6 ln
);
```

Comments

In general, ObjectSendUpdateWarn macro should be used.

**ObjectSendTaskWarning**

Same as ObjectSendTask(), additionally prints a debugging message if status less than stsOK.

Returns STATUS.

Function Prototype

```
STATUS EXPORTED ObjectSendTaskWarning(
    MESSAGE msg,
    OBJECT object,
    P_ARGS pArgs,
    SIZEOF lenArgs,
    OS_TASK_ID task,
    P_STRING fn,
    UI6 ln
);
```

Comments

In general, ObjectSendTaskWarn macro should be used.
ObjectSendUpdateTaskWarning
Same as ObjectSendUpdateTask(), additionally prints a debugging message if status less than stsOK.
Returns STATUS.

Function Prototype

```c
STATUS EXPORTED ObjectSendUpdateTaskWarning(
    MESSAGE  msg,
    OBJECT   object,
    P_ARGS   pArgs,
    SIZEOF   lenArgs,
    OS_TASK_ID task,
    P_STRING fn,
    UI6      ln
);
```

Comments
In general, ObjectSendUpdateTaskWarn macro should be used.

ObjectPostWarning
Same as ObjectPost(), additionally prints a debugging message if status less than stsOK.
Returns STATUS.

Function Prototype

```c
STATUS EXPORTED ObjectPostWarning(
    MESSAGE  msg,
    OBJECT   object,
    P_ARGS   pArgs,
    SIZEOF   lenArgs,
    P_STRING fn,
    UI6      ln
);
```

Comments
In general, ObjectPostWarn macro should be used.

ObjectPostAsyncWarning
Same as ObjectPostAsync(), additionally prints a debugging message if status less than stsOK.
Returns STATUS.

Function Prototype

```c
STATUS EXPORTED ObjectPostAsyncWarning(
    MESSAGE  msg,
    OBJECT   object,
    P_ARGS   pArgs,
    SIZEOF   lenArgs,
    P_STRING fn,
    UI6      ln
);
```

Comments
In general, ObjectPostAsyncWarn macro should be used.

ObjectPostDirectWarning
Same as ObjectPostDirect(), additionally prints a debugging message if status less than stsOK.
Returns STATUS.

Function Prototype

```c
STATUS EXPORTED ObjectPostDirectWarning(
    MESSAGE  msg,
    OBJECT   object,
    P_ARGS   pArgs,
    SIZEOF   lenArgs,
    P_STRING fn,
    UI6      ln
);
```

Comments
In general, ObjectPostDirectWarn macro should be used.
ObjectPostTaskWarning
Same as ObjectPostTask(), additionally prints a debugging message if status less than stsOK.
Returns STATUS.

Function Prototype

STATUS EXPORTED ObjectPostTaskWarning(
    MESSAGE msg,
    OBJECT object,
    P_ARGS pArgs,
    SIZEOF lenArgs,
    OS_TASK_ID task,
    P_STRING fn,
    UI6 ln
);

Comments
In general, ObjectPostTaskWarn macro should be used.

ObjectPostAsyncTaskWarning
Same as ObjectPostAsyncTask(), additionally prints a debugging message if status less than stsOK.
Returns STATUS.

Function Prototype

STATUS EXPORTED ObjectPostAsyncTaskWarning(
    MESSAGE msg,
    OBJECT object,
    P_ARGS pArgs,
    SIZEOF lenArgs,
    OS_TASK_ID task,
    P_STRING fn,
    UI6 ln
);

Comments
In general, ObjectPostAsyncTaskWarn macro should be used.

ObjectPostDirectTaskWarning
Same as ObjectPostDirectTask(), additionally prints a debugging message if status less than stsOK.
Returns STATUS.

Function Prototype

STATUS EXPORTED ObjectPostDirectTaskWarning(
    MESSAGE msg,
    OBJECT object,
    P_ARGS pArgs,
    SIZEOF lenArgs,
    OS_TASK_ID task,
    P_STRING fn,
    UI6 ln
);

Comments
In general, ObjectPostDirectTaskWarn macro should be used.

ObjectWarning
Prints object warning message. Low-level routine.
Returns nothing.
Function Prototype: void EXPORTED ObjectWarning(
    P_STRING label,
    MESSAGE msg,
    OBJECT object,
    P_ARGS pArgs,
    STATUS sts,
    P_STRING fn,
    U16 ln
);

Debugging Helper Macros (with /DDEBUG)

Conditional macros. Under /DDEBUG generates indirect calls via debugging functions, without /DDEBUG generates direct calls.

The only difference between the Warn() form and the plain form of these calls is that Warn() prints an error message if sts < stsOK AND the module was compiled for DEBUG. Use of the Warn() form is strongly encouraged.

ObjectCall

#define ObjCallWarn(m,o,p) ObjectCallWarning(m,o,p,__FILE__,__LINE__)
#define ObjCallNoDebugWarn(m,o,p) \ ObjectCallNoDebugWarning(m,o,p,__FILE__,__LINE__)
#define ObjCallAncestorCtxWarn(c) \ ObjectCallAncestorCtxWarning(c,__FILE__,__LINE__)
#define ObjCallAncestorWarn(m,o,p,c) \ ObjectCallAncestorWarning(m,o,p,c,__FILE__,__LINE__)

ObjectSend

#define ObjSendWarn(m,o,p,l) ObjectSendWarning(m,o,p,l,__FILE__,__LINE__)
#define ObjSendUpdateWarn(m,o,p,l) \ ObjectSendUpdateWarning(m,o,p,l,__FILE__,__LINE__)
#define ObjSendTaskWarn(m,o,p,l,t) \ ObjectSendTaskWarning(m,o,p,l,t,__FILE__,__LINE__)
#define ObjSendUpdateTaskWarn(m,o,p,l,t) \ ObjectSendUpdateTaskWarning(m,o,p,l,t,__FILE__,__LINE__)
#define ObjSendU32Warn(m,o,p) ObjectSendWarning(m,o,p,OL,__FILE__,__LINE__)

ObjectPost

#define ObjPostWarn(m,o,p,l) ObjectPostWarning(m,o,p,l,__FILE__,__LINE__)
#define ObjPostAsyncWarn(m,o,p,l) \ ObjectPostAsyncWarning(m,o,p,l,__FILE__,__LINE__)
#define ObjPostDirectWarn(m,o,p,l) \ ObjectPostDirectWarning(m,o,p,l,__FILE__,__LINE__)
#define ObjPostTaskWarn(m,o,p,l,t) \ ObjectPostTaskWarning(m,o,p,l,t,__FILE__,__LINE__)
#define ObjPostAsyncTaskWarn(m,o,p,l,t) \ ObjectPostAsyncTaskWarning(m,o,p,l,t,__FILE__,__LINE__)
#define ObjPostDirectTaskWarn(m,o,p,l,t) \ ObjectPostDirectTaskWarning(m,o,p,l,t,__FILE__,__LINE__)
#define ObjPostU32Warn(m,o,p) ObjectPostWarning(m,o,p,OL,__FILE__,__LINE__)
#else // DEBUG
Debugging Helper Macros (without /DDEBUG)

ObjectCall

#define ObjCallWarn(m,o,p) ObjectCall(m,o,p)
#define ObjCallNoDebugWarn(m,o,p) ObjectCall(m,o,p)
#define ObjCallAncestorCtxWarn(c) ObjectCallAncestorCtx(c)
#define ObjCallAncestorWarn(m,o,p,c) ObjectCallAncestor(m,o,p,c)

ObjectSend

#define ObjSendWarn(m,o,p,l) ObjectSend(m,o,p,l)
#define ObjSendUpdateWarn(m,o,p,l) ObjectSendUpdate(m,o,p,l)
#define ObjSendTaskWarn(m,o,p,l,t) ObjectSendTask(m,o,p,l,t)
#define ObjSendUpdateTaskWarn(m,o,p,l,t) ObjectSendUpdateTask(m,o,p,l,t)
#define ObjSendU32Warn(m,o,p) ObjectSendU32(m,o,p)

ObjectPost

#define ObjPostWarn(m,o,p,l) ObjectPost(m,o,p,l)
#define ObjPostAsyncWarn(m,o,p,l) ObjectPostAsync(m,o,p,l)
#define ObjPostDirectWarn(m,o,p,l) ObjectPostDirect(m,o,p,l)
#define ObjPostTaskWarn(m,o,p,l,t) ObjectPostTask(m,o,p,l,t)
#define ObjPostAsyncTaskWarn(m,o,p,l,t) ObjectPostAsyncTask(m,o,p,l,t)
#define ObjPostDirectTaskWarn(m,o,p,l,t) ObjectPostDirectTask(m,o,p,l,t)
#define ObjPostU32Warn(m,o,p) ObjectPost(m,o,p,0L)

#endif // DEBUG
#endif
This file contains the definitions of some of PenPoint's debugging support.
The functions described in this file are contained in PENPOINT.LIB.

Introduction.
This file contains the definitions of some of PenPoint's debugging support.
One of the most important characteristics of this package is that many of the macros compile into
nothing unless the pre-processor variable DEBUG is defined during compilation.

Debugging Flags.
As part of its debugging support, PenPoint includes a collection of debugging flags which allow
developers to control the runtime behavior of their programs.
For convenience, the debugging flags are broken into "sets" of 32 one bit flags. In PenPoint 1.0, there
are 255 sets; future versions of PenPoint may have more sets. Some sets are reserved for use by PenPoint
itself; all other sets are available for use by other developers. The allocation of sets is documented
elsewhere in this file.

Setting and Examining Debug Flags.
The debugging flags can be set via the DebugSet environment variable in PenPoint's environ.ini file. The
d Debugging flags can also be set with the "fs" command in the MiniDebugger and DB. (The debugging
flags can be examined with the "fl" command.) Both the environ.ini file and the PenPoint debuggers
allow the flag sets to be identified with either a or an 8 bit hexadecimal number. See the PenPoint
developer's documentation for more information.

Example.
The debugging output in the following fragment appears only if the code was compiled with DEBUG
defined and the debug flag is on.
As illustrated in this example, most debugging code should surrounded by some sort of conditional
compilation that causes the debugging code to "disappear" when compiled appropriately.

if (someCondition) {
    DbgFlag(0x80, 0x1, Debugf("someCondition is TRUE"));
    ...
} else {
    DbgFlag(0x80, 0x1, Debugf("someCondition is FALSE"));
    ...
}

Here's an example of setting debugging flags in PenPoint's environ.ini file:

DebugSet=/DB8000 /DB800
...
### Exported Macros

**DbgFlag**

Executes an expression under control of a debug flag IF the source is compiled with DEBUG defined.

Returns void.

```c
#define DbgFlag(f, v, e) if (DbgFlagGet(f, v)) e
#else
#define DbgFlag(f, v, e)
#endif
```

**Comments**

The DbgFlag() macro is used to execute an expression if (1) the source module was compiled with DEBUG defined and (2) if the appropriate debugging flag is turned on at runtime.

**Dbg**

Used to control the compile-time inclusion of debugging code.

Returns void.

```c
#define Dbg(x) x
#endif
```

**Comments**

The Dbg() macro is used to comment out code when the DEBUG flags is undefined. For example, the following code is present if the source file is compiled with DEBUG defined but "disappears" if DEBUG is not defined.

```c
dbg(Debugf("Only shows up in DEBUG version");)
```

**ASSERT**

Used to verify that some runtime condition is true.

Returns void.

```c
#define ASSERT(cond, str) (void)!((cond) ?
    (Debugf("== ERROR, File: %s, Line: %d ==> %s
", __FILE__, __LINE__, str)), 1: 0))
#else
#define ASSERT(cond, str)
#endif
```

**Comments**

The ASSERT() macro is used to test for conditions and print out a warning if the condition is violated. The code "disappears" if the module is compiled without DEBUG being defined.

**See Also**

assert.h
Exported Functions

**Debugf**

Prints a formatted string on the debug output device, followed by a newline.

Returns void.

```c
void CDECL Debugf(char* str, ...);
```

Comments

Debugf is very similar to the standard C runtime library function `printf()` except that (1) Debugf directs its output to PenPoint's debug output device and (2) Debugf prints a newline at the end of its output.

Unless surrounded by something `DbgO` or `DbgFlagO`, Debugf does not disappear, even if compiled without `DEBUG` defined.

Use DPrintf to avoid having the trailing newline printed.

See Also

DPrintf

**DPrintf**

Prints a formatted string on the debug output device.

Returns void.

```c
void CDECL DPrintf(char* str, ...);
```

Comments

DPrintf is very similar to the standard C runtime library function `printf()` except that DPrintf directs its output to PenPoint's debug output device.

Unless surrounded by something `DbgO` or `DbgFlagO`, DPrintf does not disappear, even if compiled without `DEBUG` defined.

See Also

Debugf

**DbgFlagSet**

Sets the specified flag set to the value of the new flags.

Returns void.

```c
void EXPORTED
DbgFlagSet (U16 set, U32 flags);
```

Comments

This function sets the specified flag set to the value of the new flags.

It is unusual for a program to call this function; most developers should set the value of debugging flags using the techniques described in the introduction of this file rather than executing this function.

Unless surrounded by something `DbgO` or `DbgFlagO`, DPrintf does not disappear, even if compiled without `DEBUG` defined.
DbgFlagGet

Returns the state of the indicated flag set ANDed with the flags mask.

Returns void.

U32 EXPORTED

Function Prototype

DbgFlagGet (U16 set, U32 flags);

set flag set selector in the range 0..255, inclusive. (Defined as a U16 to allow for possible future expansion.)

flags flags mask

Unless surrounded by something Dbg() or DbgFlag(), DbgFlagGet does not disappear, even if compiled without DEBUG defined.

### Debugging Flag Set Allocations

Not to be used by anyone (interferes with parsing process):

0x00
0x09
0x0A
0x0D
0x1A
0x20

Reserved for use outside of GO:

Lower case alphabet, except f, h, i, s, and z.

0x30 .. 0x39 digits
0x80 .. 0xBF half of the upper range

Reserved for use by GO

'f'
'h'
'i'
'q'
's'
'z'
everything else

Here are the allocations within GO's range. See other header files for more information on the interpretation of these flags. Most flags only have effect if you load the debug versions of DLLs.

'f': GO Application Developer's Course
'h': Hwxtool and Insertion Pads
'q': Quick Help
's': Hwxtool
'z': Xlate
'A': Misc. system use.

A0001: Print loader information while loading
'B': System
B0001: Turns uuid cache tracing on
B0002: Enables OEM app/service installation after warm-boot This should only be turned on for tablet hardware; never on the SDK!
B0800: Enables theSelectedVolume disk viewing in Connections

'C': ClsMgr

'D': Debug system

d0001: disables all DebugStr output
d0002: disables StringPrint output
d0004: disables System Log output
d0008: disables System Log Non Error output
d0010: disables System Log App Error output
d0020: disables System Log System Error output
d8000: writes output to PENPOINT.LOG, file flushed every n chars based on the environment variable DebugLogFlushCount.
d10000: disables mini-debugger in production version of Penpoint
d20000: disables memory statistics gestures (M,N,T) on Bookshelf
d40000: disables °C entering the mini-debugger
d80000000: allows logging to log file even if in file system code (This may cause deadlocks and is for internal use only).

'E': Environment flags

'F': Application Developer's Course

'G': Kernel

'H': Service and Service Manager

'H0001: turns on message tracing in clsService
'H0002: turns on message tracing in clsServiceMgr
'H8000: run sanity test in service.dll

'I': Installers (see instlmgr.h)

'J': Notebook

'K': UI Toolkit

'L': PicSegs and TIFF images

'L0001: dumps the TIFF image tags.

'M': misc.lib

'M0001: tracing in OrderedSetDelete
'M0002: tracing in OrderedSetFindMinMax & MaxMin
'M0004: tracing in OrderedSetInsertn
'M0008: tracing in OrderedSetSearch
'M0100: write/read debug header&trailer when filing ByteArray

'N': MiniText

'O': Outbox (obxserv and oboxsect)

'O0001: enable automatic activate of outbox Notebook

'P': Printing

'Q': text.dll

'R': Application Framework
'S': Spelling, Proof, and XTemplate systems

S0001: low-level Spell/Proof debugs
S0002: medium-level Spell/Proof debugs
S0004: high-level Spell/Proof debugs
S0010: XTemplate display inputs
S0020: XTemplate display outputs

'T': text.dll

'U': undo.dll

'V': text.dll

'W': Window system

'X': xfer.lib

'Y': TOPS

'Z': Handwriting

'@': Bookshelf

'=': MiniNote/NotePaper

'#': GWin

'!': Test Manager

'$': File System

'%': UI Toolkit

'*': Heap Manager

0xCO: Fax Project

0xC1: Input

0xC2: VKey

0xC3: System Log trace flag

0xC4: 2.0 tools

0xFD: Memory Tests // Internal use only

0xF1: Memory Tests // Internal use only

0xFF: C Runtime Library
This file contains PenPoint’s standard #defines, types and intrinsics. Essentially all PenPoint source files must include this file.

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

```c
#ifndef GO_INCLUDED
#define GO_INCLUDED

Standard Definitions

Static Declarations

Functions declared STATIC (rather than static) will, when compiled with DEBUG defined, appear in map files.

```c
#ifndef DEBUG
#define STATIC static
#else
#define STATIC static
#endif
```

Function Scope Definitions

- LOCAL: Scope is module wide
- GLOBAL: Scope is subsystem wide
- EXPORTED: Scope is ring wide (either ring0 OR ring3)
- EXPORTEDO: Scope is system wide. For public ring0 functions.
- RINGCHELPER: Scope is system wide. For private ring0 functions.

```c
#define LOCAL STATIC PASCAL
#define GLOBAL PASCAL
#define EXPORTED PASCAL
#define EXPORTEDO PASCAL
#define RINGCHELPER PASCAL
```

Null values

```c
#ifndef M_I36
#define NULL 0 // 32 bit compiler
#else
#define NULL 0L // 16 bit compiler
#endif
```

```c
#define null 0
#define pNull ((P_UNKNOWN)0)
#define ppNull ((PP_UNKNOWN)0)
#define Nil(type) ((type)0)
```

Boolean operators

```c
#define AND &&
#define OR ||
#define NOT !
#define MOD %
```
Bit flags.

These flags can be used with FlagOn, FlagOff, FlagSet, and FlagClr.

```c
#define flag0  (0x0001)
#define flag1  (0x0002)
#define flag2  (0x0004)
#define flag3  (0x0008)
#define flag4  (0x0010)
#define flag5  (0x0020)
#define flag6  (0x0040)
#define flag7  (0x0080)
#define flag8  (0x0100)
#define flag9  (0x0200)
#define flag10 (0x0400)
#define flag11 (0x0800)
#define flag12 (0x1000)
#define flag13 (0x2000)
#define flag14 (0x4000)
#define flag15 (0x8000)
#define flag16 (0x00010000L)
#define flag17 (0x00020000L)
#define flag18 (0x00040000L)
#define flag19 (0x00080000L)
#define flag20 (0x00100000L)
#define flag21 (0x00200000L)
#define flag22 (0x00400000L)
#define flag23 (0x00800000L)
#define flag24 (0x01000000L)
#define flag25 (0x02000000L)
#define flag26 (0x04000000L)
#define flag27 (0x08000000L)
#define flag28 (0x10000000L)
#define flag29 (0x20000000L)
#define flag30 (0x40000000L)
#define flag31 (0x80000000L)
```

Limits

```c
#define maxU8    ((U8)0xFF)
#define minS8    ((S8)0x80)
#define maxS8    ((S8)0x7F)
#define maxU16   ((U16)0xFFFF)
#define minS16   ((S16)0x8000)
#define maxS16   ((S16)0x7FFF)
#define maxU32   ((U32)0xFFFFFFFF)
#define minS32   ((S32)0x80000000)
#define maxS32   ((S32)0x7FFFFFFF)
```

Name limits

```c
#define maxNameLength 32
#define nameBufLength  (maxNameLength+1)
```

Enums

Different compilers allocate different amounts of space for an enum. To avoid portability problems, use the Enum16 and Enum32 macros. They guarantee that the enum is 16 bits or 32 bits, respectively.

Example:

```c
Enum16(PRIMARY_COLOR) {
    red,
    green,
    blue
}
```
#define Enum16(name) typedef S16 name, * P_##name; enum name
#define Enum32(name) typedef S32 name, * P_##name;

calling conventions
#if defined __WATCOM__
#define PASCAL __pascal
#define cdecl __cdecl
#define Unused(x) (void) (x)
#define FunctionPtr(fn) (PASCAL * fn)
#define CFunctionPtr(fn) (cdecl * fn)
#endif
#if defined __386__
#pragma aux pascal "\" parm routine []
    value struct float struct caller [eax] modify [eax ecx edx gs];
#pragma aux cdecl "_*" parm caller []
    value struct float struct caller [eax] modify [eax edx gs];
#endif
#if defined __HIGHC__
#define PASCAL CC( REVERSE_PARM | CALLEE_POPS | STACK)
#define cdecl /1
#define Unused(x)
#define FunctionPtr(fn) PASCAL (* fn)
#define CFunctionPtr(fn) cdecl (* fn)
#else
#define PASCAL pascal
#define cdecl cdecl
#define Unused(x) (void) (x)
#define FunctionPtr(fn) cdecl (* fn)
#define CFunctionPtr(fn) cdecl (* fn)
#endif

typedefs

unsigned integers
typedef unsigned char U8, * P_U8, ** PP_U8; // 8-bit unsigned
typedef unsigned short U16, * P_U16, ** PP_U16; // 16-bit unsigned
#ifndef M_I86
    typedef unsigned int U32, * P_U32, ** PP_U32; // 32-bit unsigned
#else
    typedef unsigned int U32, * P_U32, ** PP_U32; // 16-bit unsigned
#endif

signed integers
typedef signed char S8, * P_S8, ** PP_S8; // 8-bit signed
typedef signed short S16, * P_S16, ** PP_S16; // 16-bit signed
#ifndef M_I86
    typedef signed int S32, * P_S32, ** PP_S32; // 32-bit signed
#else
    typedef signed int S32, * P_S32, ** PP_S32; // 16-bit signed
#endif

wide characters. In penpoint 1.0 these are 8 bit values. In penpoint 2.0 and forward they are 16 bit values.
typedef U8 CHAR;
typedef P_U8 P_CHAR;
typedef P_CHAR* PP_CHAR;

8 bit characters
typedef U8 CHAR8; // These are guaranteed to stay 8-bit
typedef P_U8 P_CHAR8;
typedef P_CHAR8* PP_CHAR8;
16 bit Characters

typedef U16 CHAR16; // These are guaranteed to stay 16-bit
typedef P_U16 P_CHAR16;
typedef P_CHAR16* PF_CHAR16;

Strings

typedef U8 STRING;
typedef P_U8 P_STRING;
typedef P_STRING* PF_STRING;

SIZEOF is the type returned by the SizeOf. It is guaranteed to be 32 bits.
typedef U32 SIZEOF, * P_SIZEOF;

Pointer to an opaque entity

typedef void* P_UNKNOWN;
typedef P_UNKNOWN* PF_UNKNOWN;

Generic pointer to procedure

typedef PUNKNOWN FunctionPtr(PPROC);

True/False values

Enum16(BOOLEAN) {
    FALSE = 0,
    TRUE = 1,
    False = 0,
    True = 1,
    false = 0,
    true = 1
};

**Intrinsics**

#define Abs(v) ((v)<0?(-(v)):v)
#define Max(a,b) ((a)>(b)?a:b)
#define Min(a,b) ((a)<(b)?a:b)
#define Odd(v) ((v)&1)
#define Even(v) (!Odd(v))
#define LowU16(dw) ((U16)(U32)(dw))
#define HighU16(dw) ((U16)(((U32)(dw)>>16))
#define LowU8(w) ((U8)(w))
#define HighU8(w) ((U8)((U16)(w)>>8))
#define MakeU16(lb,hb) (((U16)(hb)<<8)|(U16)(lb))
#define MakeU32(lw,hw) (((U32)(hw)<<16)|(U32)(lw))
#define FlagOn(f,v) (!FlagOff(f,v))
#define FlagOff(f,v) (!((v)&(f)))
#define FlagSet(f,v) ((v)|(f))
#define FlagClr(f,v) ((v) & ~ (f))
#define OutRange(v,l,h) ((v)<(l)||(v)>(h))
#define InRange(v,l,h) ((v)>(l)&&(v)<(h))
#define SizeOf(t) ((SIZEOF)sizeof(t))
Commonly Used Class Manager Types

A variable of type OBJECT identifies an object. The type UID is interchangeable with OBJECT. A variable of type TAG identifies one of the following:

- Tag
- Message
- Error status (values less than 0)
- Warning status (values greater than or equal to 0)

Well-known UID Structure

A UID is constructed as:

- Version: 7 bits
- Admin: 20 or 19 bits
- Scope: 1 or 2 bits
- Layout:

```
000000000111111112222222233
01234567890123456789012345678901
```

typedef P_UNKNOWN U_ID, * P_UID;
typedef UID OBJECT, * P_OBJECT, ** P_P_OBJECT;

Well-known UID Macros

Create a well-known UID

```c
#define MakeWKN(admin, version, scope) 
((UID)((U32)(0x7F&(version»24 | (U32)(admin) »1 + (scope&1) » scope))
```

Create a well-known UID

```c
#define MakeGlobalWKN(admin, version) MakeWKN(admin, version, wknGlobal)
```

Create a process-global well-known UID

```c
#define MakeProcessGlobalWKN(admin, version) 
MakeWKN(admin, version, wknProcessGlobal)
```

Create a private well-known UID

```c
#define MakeProcessPrivateWKN(admin, version) MakeWKN(admin, version, wknPrivate)
```

Extract the admin number plus the scope information

```c
#define WKNValue(wkn) (0x1FFFFF&(U32)wkn)
```

Extract the admin number

```c
#define WKNAdmin(wkn) (WKNValue(wkn»1+(U32)wkn&1» scope))
```

Extract the version number

```c
#define WKNVer(wkn) ((U32)wkn»24)
```
Extract the scope

```c
#define WKNScope(wkn)   ((U32) (wkn) &~((U32) (wkn) &1) &3)
```

Magic constants

```c
#define wknGlobal 0
#define wknProcessGlobal 1
#define wknPrivate 3
```

**Tag Structure**

Tags are created using a well-known Administered value and a tag number in the range 0-255.

- **X**: 1 bit. 0 for tag or Warning Status; 1 for an Error Status.
- **TagNum**: 8 bits
- **Flags**: 2 bits
- **Admin**: 20 or 19 bits
- **Scope**: 1 or 2 bits
- **Layout**:

```
00000000001111111122222222222233
01234567890123456789012345678901
```

<table>
<thead>
<tr>
<th>Name</th>
<th>X</th>
<th>tagNum</th>
<th>F</th>
<th>Admin+Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>1</td>
<td>8</td>
<td>2</td>
<td>20+1 or 19+2</td>
</tr>
</tbody>
</table>

```c
typedef S32 TAG, *P_TAG; // Tags are always positive
typedef S32 STATUS, *P_STATUS;
```

**Tag Macros**

Create a tag

```c
#define MakeTag(wkn,tagNum) (((TAG) (tagNum) &0xFF)<<23|WKNSValue(wkn))
```

Create a tag with flags

```c
#define MakeTagWithFlags(wkn, i, f) (MakeTag (wkn, i) | ((U32) (f) &3) <<21)
```

Extract the tag num

```c
#define TagNum(tag)   ((U32) (tag) <<1>>24)
#define Tag(tag)      TagNum(tag)
```

Extract the tag num and flags together

```c
#define TagAndFlags(tag) ((U32) (tag) <<1>>22)
```

Extract only the tag flags

```c
#define TagFlags(tag) (TagAndFlags(tag) &3)
```

Extract the tag admin

```c
#define TagAdmin(tag) WKNAadmin(tag)
```
### Status Macros

Create an error status

```
#define MakeStatus(wkn, sts) ((STATUS)(0x80000000|MakeTag(wkn, sts)))
```

Create a warning status

```
#define MakeWarning(wkn, sts) ((STATUS)MakeTag(wkn,sts))
```

Extract the status num from a STATUS

```
#define Sts(sts) Tag(sts)
```

### Debugging Macros

```
define StsRet(se, s)    if (((s) = StsWarn(se)) < stsOK) return s; else
#define StsJmp(se, s, x)   if (((s) = StsWarn(se)) < stsOK) goto x; else
#define StsOK(se, s)      (((s) = StsWarn(se)) >= stsOK)
define StsFailed(se, s) (((s) = StsWarn(se)) < stsOK)
define StsChk(se, s)    (((s) = (se)) < stsOK)
```

### Status Printing Macros

#### StsWarn

Prints status warning message.

Returns nothing.

```
#define StsWarn(se) StatusWarning(se, __FILE__, __LINE__)
define StsWarn(se) (se)
```

**Comments**

When DEBUG is defined during compilation, the StsWarn macro prints a status warning message if the status is less than stsOK (an error). When DEBUG is not defined during compilation, StsWarn simply evaluates its expression.

**See Also**

StsPrint

---

#### StsPrint

Prints status warning message.

Returns nothing.

```
define StsPrint(s)   StatusWarning(s, __FILE__, __LINE__)
define StsPrint(s)   // if not DEBUG
```

**Comments**

When DEBUG is defined during compilation, the StsPrint macro prints a status warning message regardless of the value of the status. When DEBUG is not defined during compilation, StsPrint does nothing.

**See Also**

StsWarn
### Status Values

// Next up: 11
Classes used to create generic status values (see uid.h)

```c
#define clsGO
#define clsOS
#define clsGOMath
```

Values

```c
#define stsBadParam
#define stsNoMatch
#define stsEndOfData
#define stsFailed
#define stsTimeOut
#define stsRequestNotSupported
#define stsReadOnly
#define stsIncompatibleVersion
#define stsNotYetImplemented
#define stsOutOfMem
```

Non-Error Status Values

// Next up: 4

```c
#define stsOK
#define stsRequestDenied
#define stsRequestForward
#define stsTruncatedData
```

### GO Math Support

Conceptually these declarations should be in gomath.h. They are defined here instead to ease the load on the compiler symbol tables.

```c
typedef S32 FIXED;
typedef FIXED* P_FIXED;
FIXED PASCAL FxMakeFixed(S16 whole, U16 frac);
```
Prototype for main().
#ifndef MAIN_INCLUDED
#define MAIN_INCLUDED
#ifndef GO_INCLUDED
#include <go.h>
#endif
#endif

Standard main()

Function Prototype  U32 CDECL main(S32 argc, CHAR* argv[], U32 instance);
This contains well-known uids for PenPoint.

```c
#ifndef UID_INCLUDED
#define UID_INCLUDED

Available for Testing (wknGlobals)
#define wknGDTa MakeWKN(3,1,wknGlobal)
#define wknGDTb MakeWKN(4,1,wknGlobal)
#define wknGDTc MakeWKN(5,1,wknGlobal)
#define wknGDTd MakeWKN(6,1,wknGlobal)
#define wknGDTe MakeWKN(7,1,wknGlobal)
#define wknGDTf MakeWKN(8,1,wknGlobal)
#define wknGDTg MakeWKN(9,1,wknGlobal)
#define wknGDTth MakeWKN(32,1,wknGlobal)
#define wknGDTi MakeWKN(45,1,wknGlobal)
#define wknGDTj MakeWKN(47,1,wknGlobal)
#define wknGDTk MakeWKN(73,1,wknGlobal)

Available for Testing (wknProcessGlobals)
#define wknLDTa MakeWKN(3,1,wknProcessGlobal)
#define wknLDTb MakeWKN(4,1,wknProcessGlobal)
#define wknLDTc MakeWKN(5,1,wknProcessGlobal)
#define wknLDTd MakeWKN(6,1,wknProcessGlobal)
#define wknLDTe MakeWKN(7,1,wknProcessGlobal)
#define wknLDTf MakeWKN(8,1,wknProcessGlobal)
#define wknLDTg MakeWKN(9,1,wknProcessGlobal)

Well-known Objects
#define objNull MakeWKN(0,0,0)
#define clsProcess MakeWKN(0,1,wknGlobal)
#define clsObject MakeWKN(1,1,wknGlobal)
#define clsClass MakeWKN(2,1,wknGlobal)
#define theProcess MakeWKN(0,0,wknProcessGlobal)
#define clsGO MakeWKN(14,1,wknGlobal)
#define clsOS MakeWKN(16,1,wknGlobal)
#define clsGOMath MakeWKN(162,1,wknGlobal)
#define clsMisc MakeWKN(112,1,wknGlobal)
#define clsSystem MakeWKN(174,1,wknGlobal)
#define theSystem MakeWKN(174,1,wknGlobal)
#define clsInitTask MakeWKN(433,1,wknGlobal)
#define theSystemInitTask MakeWKN(431,1,wknGlobal)
#define theThirdPartyInitTask MakeWKN(432,1,wknGlobal)
#define theBookshelf MakeWKN(127,1,wknGlobal)
#define theSystemResFile MakeWKN(172,1,wknGlobal)
#define theMILResFile MakeWKN(414,1,wknGlobal)
#define theDesktop MakeWKN(127,1,wknGlobal)  // obsolete
```
# Application Framework

```c
#define clsApp MakeWKN(13,1,wknGlobal)
#define clsAppMgr MakeWKN(69,1,wknGlobal)
#define clsAppDir MakeWKN(157,1,wknGlobal)
#define clsAppWin MakeWKN(159,1,wknGlobal)
#define clsAppWinIcon MakeWKN(153,1,wknGlobal)
#define clsContainerApp MakeWKN(121,1,wknGlobal)
#define clsRootContainerApp MakeWKN(218,1,wknGlobal)
#define clsList MakeWKN(10,1,wknGlobal)
#define clsView MakeWKN(15,1,wknGlobal)
#define clsEmbeddedWin MakeWKN(11,1,wknGlobal)
#define clsIconWin MakeWKN(80,1,wknGlobal)
#define clsGotoButton MakeWKN(183,1,wknGlobal)
#define clsPowerButtonUI MakeWKN(458,1,wknGlobal)
#define clsCorkBoardWin MakeWKN(148,1,wknGlobal)
#define clsMemoryCop MakeWKN(443,1,wknGlobal)
#define theMemoryCop MakeWKN(457,1,wknGlobal)
```

# Bookshelf

```c
#define clsBSApp MakeWKN(168,1,wknGlobal) // PenPoint internal
#define clsBSMainWin MakeWKN(167,1,wknGlobal) // PenPoint internal
#define clsBSWin MakeWKN(359,1,wknGlobal) // PenPoint internal
#define clsBSZTWin MakeWKN(164,1,wknGlobal) // PenPoint internal
```

# Notebook

```c
#define clsNBApp MakeWKN(44,1,wknGlobal)
#define clsNBToc MakeWKN(136,1,wknGlobal)
#define clsSectApp MakeWKN(145,1,wknGlobal)
#define clsNBFframe MakeWKN(92,1,wknGlobal) // PenPoint internal
#define clsBookmark MakeWKN(184,1,wknGlobal) // PenPoint internal
#define clsPageControl MakeWKN(156,1,wknGlobal) // PenPoint internal
#define clsPageWin MakeWKN(161,1,wknGlobal) // PenPoint internal
#define clsSectMenu MakeWKN(226,1,wknGlobal) // PenPoint internal
#define clsBSApp MakeWKN(284,1,wknGlobal) // PenPoint internal
#define clsBSMenu MakeWKN(83,1,wknGlobal) // PenPoint internal
```

# Input

```c
#define theInputManager MakeWKN(17,1,wknGlobal)
#define clsInput MakeWKN(17,1,wknGlobal)
#define thePen MakeWKN(18,1,wknGlobal)
#define clsPen MakeWKN(18,1,wknGlobal)
#define theKeyboard MakeWKN(19,1,wknGlobal)
#define clsKey MakeWKN(19,1,wknGlobal)
#define clsAcetateAlign MakeWKN(90,1,wknGlobal)
```

# Hwx Tools

```c
#define clsScribble MakeWKN(20,1,wknGlobal)
#define clsSPaper MakeWKN(21,1,wknGlobal)
#define clsIP MakeWKN(77,1,wknGlobal)
#define clsIPButton MakeWKN(79,1,wknGlobal)
#define clsGWin MakeWKN(219,1,wknGlobal)
#define clsField MakeWKN(22,1,wknGlobal)
```
Virtual Keyboard

#define clsKeyCap MakeWKN(96,1,wknGlobal)
#define clsKeyboard MakeWKN(97,1,wknGlobal)
#define theVirtualKeyboard MakeWKN(199,1,wknGlobal)
#define clsVKeyApp MakeWKN(198,1,wknGlobal)
#define clsVKeyWin MakeWKN(132,1,wknGlobal)

The System Log Application

#define theSystemLog MakeWKN(46,1,wknGlobal)
#define clsSystemLog MakeWKN(78,1,wknGlobal)
#define clsSysLogApp MakeWKN(330,1,wknGlobal)
#define clsTextOut MakeWKN(39,1,wknGlobal)  // PenPoint internal

Quick Help

#define theQuickHelpManager MakeWKN(85,1,wknGlobal)
ifndef NO_GRANDFATHER
#define theQuickHelp theQuickHelpManager
endif
#define clsQuickHelp MakeWKN(85,1,wknGlobal)
#define clsQHWin MakeWKN(154,1,wknGlobal)

Printing

#define clsPrFrame MakeWKN(279,1,wknGlobal)
#define clsPrint MakeWKN(280,1,wknGlobal)
#define thePrintManager MakeWKN(281,1,wknGlobal)
#define clsPrMgr MakeWKN(281,1,wknGlobal)
#define clsPrintManager MakeWKN(379,1,wknGlobal)
#define clsPrMargin MakeWKN(283,1,wknGlobal)
#define clsPrLayout MakeWKN(397,1,wknGlobal)

Battery

#define theBatteries MakeWKN(354,1,wknGlobal)
#define theBattery MakeWKN(282,1,wknGlobal)

HWX

#define clsXlate MakeWKN(23,1,wknGlobal)
#define clsXtract MakeWKN(98,1,wknGlobal)
#define clsXText MakeWKN(99,1,wknGlobal)
#define clsXWord MakeWKN(101,1,wknGlobal)
#define clsXGesture MakeWKN(102,1,wknGlobal)
#define clsXNumber MakeWKN(103,1,wknGlobal)
#define clsXGeometric MakeWKN(104,1,wknGlobal)
#define theHWXProtos MakeWKN(105,1,wknGlobal)
#define clsHWXProto MakeWKN(105,1,wknGlobal)
#define clsXTeach MakeWKN(100,1,wknGlobal)
#define clsXShape MakeWKN(251,1,wknGlobal)
#define clsGOShape MakeWKN(252,1,wknGlobal)
#define clsGOShapeService MakeWKN(253,1,wknGlobal)
#define clsCTShape MakeWKN(254,1,wknGlobal)
#define clsCTShapeService MakeWKN(255,1,wknGlobal)
File System, etc

```c
#define theFileSystem MakeWKN(62,1,wknGlobal)
#define clsFileSystem MakeWKN(62,1,wknGlobal)
#define clsDirHandle MakeWKN(28,1,wknGlobal)
#define clsFileHandle MakeWKN(29,1,wknGlobal)
#define theVolSearcher MakeWKN(143,1,wknGlobal)
#define clsVolSearch MakeWKN(143,1,wknGlobal)
#define clsVolume MakeWKN(30,1,wknGlobal)
#define clsVolRAM MakeWKN(49,1,wknGlobal)
#define clsVolMSDisk MakeWKN(61,1,wknGlobal)
#define clsVolTOPS MakeWKN(120,1,wknGlobal)
#define theBlockDeviceManager MakeWKN(412,1,wknGlobal)
#define clsBlockDeviceManager MakeWKN(412,1,wknGlobal)
#define clsBlockDevice MakeWKN(413,1,wknGlobal)
#define theSCSIDriver MakeWKN(31,1,wknGlobal)
#define clsSCSI MakeWKN(31,1,wknGlobal)
#define clsSCSI SenseCodes MakeWKN(299,1,wknGlobal)
#define clsATBiosDisk MakeWKN(302,1,wknGlobal)
#define clsResFile MakeWKN(285,1,wknGlobal)
#define clsResList MakeWKN(286,1,wknGlobal)
#define theProcessResList MakeWKN(12,1,wknProcessGlobal)
#define theBootVolume MakeWKN(138,1,wknGlobal)
#define theSelectedVolume MakeWKN(125,1,wknGlobal)
#define theWorkingDir MakeWKN(10,1,wknProcessGlobal)
#define clsFileHandleAppendOnly MakeWKN(494,1,wknGlobal)
```

Disk Viewer

```c
#define clsDiskViewWin MakeWKN(384,1,wknGlobal)
#define clsDiskInstaller MakeWKN(385,1,wknGlobal)
#define clsDVBookshelf MakeWKN(188,1,wknGlobal)
#define clsDiskViewApp MakeWKN(243,1,wknGlobal) // Penpoint internal
#define clsDVBrowser MakeWKN(141,1,wknGlobal) // PenPoint internal
#define clsDTVtabButton MakeWKN(134,1,wknGlobal) // PenPoint internal
#define clsDVIcon MakeWKN(137,1,wknGlobal) // PenPoint internal
#define clsDVFoward MakeWKN(140,1,wknGlobal) // PenPoint internal
#define clsDVBrowser MakeWKN(171,1,wknGlobal) // PenPoint internal
#define clsDVIconWin MakeWKN(144,1,wknGlobal) // PenPoint internal
#define clsDynamicTableMgr MakeWKN(128,1,wknGlobal)
```

Configuration Notebook

```c
#define clsConfigurationApp MakeWKN(197,1,wknGlobal)
#define theConfigurationBook MakeWKN(206,1,wknGlobal)
```

Settings NB

```c
#define clsSettingsNB MakeWKN(239,1,wknGlobal)
#define clsSettingsNBAppWin MakeWKN(150,1,wknGlobal) // PenPoint internal
#define clsInstallUISheet MakeWKN(117,1,wknGlobal)
#define clsInstallUICard MakeWKN(256,1,wknGlobal) // PenPoint internal
#define clsInstallUIButton MakeWKN(209,1,wknGlobal) // PenPoint internal
#define clsInstallUIBrowser MakeWKN(387,1,wknGlobal) // PenPoint internal
#define clsQuickInstallUI MakeWKN(142,1,wknGlobal) // PenPoint internal
```
Install Manager classes

#define clsInstallMgr MakeWKN(249,1,wknGlobal)
#define clsCodeInstallMgr MakeWKN(193,1,wknGlobal)
#define clsAppInstallMgr MakeWKN(260,1,wknGlobal)
#define clsFontInstallMgr MakeWKN(268,1,wknGlobal)
#define clsHWXProtoInstallMgr MakeWKN(177,1,wknGlobal)
#define clsPDictInstallMgr MakeWKN(428,1,wknGlobal)
#define clsUpgradeApp MakeWKN(291,1,wknGlobal)
#define clsUpgradeAppMonitor MakeWKN(292,1,wknGlobal)

Install Manager well-known instances

#define theInstallManagers MakeWKN(236,1,wknGlobal)
#define theInstalledHWXProtos MakeWKN(250,1,wknGlobal)
#define theInstalledGestures MakeWKN(409,1,wknGlobal)
#define theInstalledApps MakeWKN(208,1,wknGlobal)
#define theInstalledPrefs MakeWKN(331,1,wknGlobal)
#define theInstalledPDicts MakeWKN(332,1,wknGlobal)
#define theInstalledServices MakeWKN(288,1,wknGlobal)
#define theInstalledFonts MakeWKN(211,1,wknGlobal)

Application Monitor

#define clsAppMonitor MakeWKN(278,1,wknGlobal)

Auxilliary Notebook Manager

#define clsAuxNotebookMgr MakeWKN(314,1,wknGlobal)
#define theAuxNotebookMgr MakeWKN(313,1,wknGlobal)
#define clsIniFileHandler MakeWKN(398,1,wknGlobal)
#define clsStationeryMenu MakeWKN(93,1,wknGlobal) // PenPoint internal
#define theStationeryMenu MakeWKN(93,1,wknGlobal) // PenPoint internal

Auxilliary Notebooks

#define clsHelpNB MakeWKN(335,1,wknGlobal)
#define clsStationeryNB MakeWKN(333,1,wknGlobal)
#define clsStationeryBrowWin MakeWKN(160,1,wknGlobal) // PenPoint internal
#define clsInboxNB MakeWKN(388,1,wknGlobal)
#define clsOutboxNB MakeWKN(389,1,wknGlobal)

Accessory Pallette

#define clsAccessoryPalette MakeWKN(391,1,wknGlobal)
#define clsAccessoryWin MakeWKN(396,1,wknGlobal)
#define clsAccessoryAppWin MakeWKN(440,1,wknGlobal)

Service Classes

#define clsService MakeWKN(349,1,wknGlobal)
#define clsMILService MakeWKN(434,1,wknGlobal)
#define clsServiceMgr MakeWKN(350,1,wknGlobal)
#define clsServiceInstallMgr MakeWKN(240,1,wknGlobal)
#define clsPrintSpoolSvc MakeWKN(363,1,wknGlobal)
#define clsSendableService MakeWKN(169,1,wknGlobal)
#define clsHWXEngineService MakeWKN(180,1,wknGlobal)
#define clsOpenServiceObject MakeWKN(176,1,wknGlobal)
#define clsMILConflictGroupMgr MakeWKN(415,1,wknGlobal)
#define theServiceResList MakeWKN(189,1,wknGlobal)
#define theServiceManagers MakeWKN(237,1,wknGlobal)
Service Managers

#define theMILDevices MakeWKN(383,1,wknGlobal)
#define theParallelDevices MakeWKN(152,1,wknGlobal)
#define theAppleTalkDevices MakeWKN(308,1,wknGlobal)
#define theSerialDevices MakeWKN(309,1,wknGlobal)
#define thePrinterDevices MakeWKN(310,1,wknGlobal)
#define thePrinters MakeWKN(210,1,wknGlobal)
#define theSendableServices MakeWKN(24,1,wknGlobal)
#define theTransportHandlers MakeWKN(25,1,wknGlobal)
#define theLinkHandlers MakeWKN(26,1,wknGlobal)
#define theHWXEngines MakeWKN(175,1,wknGlobal)
#define theModems MakeWKN(194,1,wknGlobal)
#define theHighSpeedPacketHandlers MakeWKN(439,1,wknGlobal)
#define theFaxIOServices MakeWKN(217,1,wknGlobal)

Service Sample Code

#define clsBasicService MakeWKN(460,1,wknGlobal)
#define clsTestService MakeWKN(186,1,wknGlobal)
#define clsTestOpenObject MakeWKN(207,1,wknGlobal)
#define clsTestMILService MakeWKN(459,1,wknGlobal)

Modem Component

#define clsModem MakeWKN(151,1,wknGlobal)

Parallel Port Component

#define clsParallelPort MakeWKN(196,1,wknGlobal)

Text Component

#define clsText MakeWKN(35,1,wknGlobal)
#define clsTextView MakeWKN(36,1,wknGlobal)
#define clsTextChar MakeWKN(33,1,wknGlobal)
#define clsTextMarkStore MakeWKN(34,1,wknGlobal)
#define clsTextBlock clsText
#define clsTextIP MakeWKN(355,1,wknGlobal)

Undo Manager

#define clsUndo MakeWKN(235,1,wknGlobal)
#define theUndoCoordinator MakeWKN(126,1,wknGlobal)
#define theUndoManager MakeWKN(11,1,wknProcessGlobal)

Windows and Graphics

#define clsDrwCtx MakeWKN(37,1,wknGlobal)
#define clsSysDrwCtx MakeWKN(38,1,wknGlobal)
#define clsPixDev MakeWKN(40,1,wknGlobal)
#define clsImgDev MakeWKN(41,1,wknGlobal)
#define clsWinDev MakeWKN(42,1,wknGlobal)
#define clsWin MakeWKN(43,1,wknGlobal)
#define theScreen MakeWKN(50,1,wknGlobal)
#define theRootWindow MakeWKN(67,1,wknGlobal)
#define clsBitmap MakeWKN(378,1,wknGlobal)
#define clsPicSeg MakeWKN(82,1,wknGlobal)
#define clsTiff MakeWKN(66,1,wknGlobal)
#define clsBorder MakeWRN(135,1,wknGlobal)
#define clsLayout MakeWRN(53,1,wknGlobal)
#define clsTableLayout MakeWRN(55,1,wknGlobal)
#define clsCustomLayout MakeWRN(54,1,wknGlobal)
#define clsTrack MakeWRN(12,1,wknGlobal)

#define clsImageWin MakeWRN(182,1,wknGlobal)
#define clsFrame MakeWRN(56,1,wknGlobal)
#define clsFrameBorder MakeWRN(337,1,wknGlobal)
#define clsScrollWin MakeWRN(155,1,wknGlobal)
#define clsScrollWinInnerWin MakeWRN(338,1,wknGlobal)
#define clsControl MakeWRN(48,1,wknGlobal)
#define clsCloseBox MakeWRN(71,1,wknGlobal)
#define clsGrabBox MakeWRN(266,1,wknGlobal)
#define clsScrollbar MakeWRN(58,1,wknGlobal)
#define clsLabel MakeWRN(75,1,wknGlobal)
#define clsButton MakeWRN(52,1,wknGlobal)
#define clsMenuButton MakeWRN(72,1,wknGlobal)
#define clsContentsButton MakeWRN(192,1,wknGlobal)
#define clsIcon MakeWRN(360,1,wknGlobal)
#define clsIconToggle MakeWRN(124,1,wknGlobal)
#define clsMoveCopyIcon MakeWRN(361,1,wknGlobal)
#define clsTitleBar MakeWRN(163,1,wknGlobal)
#define clsTkTable MakeWRN(68,1,wknGlobal)
#define clsOptionTable MakeWRN(298,1,wknGlobal)
#define clsContentsTable MakeWRN(190,1,wknGlobal)
#define clsMenu MakeWRN(57,1,wknGlobal)
#define clsShadow MakeWRN(181,1,wknGlobal)
#define clsPageNum MakeWRN(74,1,wknGlobal)
#define clsTabBar MakeWRN(70,1,wknGlobal)
#define clsTabButton MakeWRN(60,1,wknGlobal)
#define clsOption MakeWRN(224,1,wknGlobal)
#define clsOptionBook MakeWRN(191,1,wknGlobal)
#define clsCommandBar MakeWRN(228,1,wknGlobal)
#define clsCounter MakeWRN(110,1,wknGlobal)

#define clsChoice MakeWRN(59,1,wknGlobal)
#define clsPopupChoice MakeWRN(297,1,wknGlobal)
#define clsToggleTable MakeWRN(76,1,wknGlobal)
#define clsIconChoice MakeWRN(320,1,wknGlobal)
#define clsIconTable MakeWRN(321,1,wknGlobal)
#define clsListBox MakeWRN(94,1,wknGlobal)
#define clsListBoxDisplay MakeWRN(275,1,wknGlobal)
#define clsManager MakeWRN(244,1,wknGlobal)
#define clsChoiceMgr MakeWRN(241,1,wknGlobal)
#define clsSelChoiceMgr MakeWRN(246,1,wknGlobal)
#define clsTextField MakeWRN(95,1,wknGlobal)
#define clsTextDisplay MakeWRN(294,1,wknGlobal)
#define clsIntegerField MakeWRN(295,1,wknGlobal)
#define clsFixedField MakeWRN(296,1,wknGlobal)
#define clsDateField MakeWRN(297,1,wknGlobal)
#define clsBusy MakeWRN(242,1,wknGlobal)
#define clsModalFilter MakeWRN(311,1,wknGlobal)
#define clsNote MakeWRN(312,1,wknGlobal)
#define clsNoteBorder MakeWRN(195,1,wknGlobal)
#define clsStringListBox MakeWRN(343,1,wknGlobal)
#define clsFontListBox MakeWRN(344,1,wknGlobal)
#define clsProgressBar MakeWRN(187,1,wknGlobal)
# Import/Export

```c
#define clsImport MakeWKN(289,1,wknGlobal)
#define clsExport MakeWKN(290,1,wknGlobal)
#define theExportManager MakeWKN(84,1,wknGlobal)
#define clsExportManager MakeWKN(106,1,wknGlobal)
```

# Browser

```c
#define clsBrowser MakeWKN(87,1,wknGlobal)
#define clsBrowWin MakeWKN(178,1,wknGlobal)
#define clsBrowApp MakeWKN(179,1,wknGlobal)
#define clsBrowFrame MakeWKN(221,1,wknGlobal)
#define clsBrowMenu MakeWKN(261,1,wknGlobal)
#define clsBrowExport MakeWKN(300,1,wknGlobal)
#define clsBrowImport MakeWKN(303,1,wknGlobal)
#define clsBrowRename MakeWKN(326,1,wknGlobal)
#define clsLuke MakeWKN(222,1,wknGlobal) // PenPoint internal
```

# Communications

```c
#define clsStream MakeWKN(64,1,wknGlobal)
#define clsSccSio MakeWKN(351,1,wknGlobal)
#define clsSio MakeWKN(381,1,wknGlobal)
#define clsSioUI MakeWKN(122,1,wknGlobal)
#define clsFLAP MakeWKN(393,1,wknGlobal)
#define clsALAPSerial MakeWKN(394,1,wknGlobal)
#define clsIconCache MakeWKN(107,1,wknGlobal)
#define clsWSio MakeWKN(123,1,wknGlobal)
#define clsSioTest MakeWKN(158,1,wknGlobal)
```

# Fax Send/Receive Page Service

```c
#define clsFaxIOSvc MakeWKN(271,1,wknGlobal)
```

# Search and Replace

```c
#define clsSR MakeWKN(293,1,wknGlobal)
#define clsSF MakeWKN(382,1,wknGlobal) // search frame
#define theSearchManager MakeWKN(27,1,wknGlobal)
```

# Traverse

```c
#define clsMark MakeWKN(257,1,wknGlobal)
```

# Textedit Application

```c
#define clsTexteditApp MakeWKN(356,1,wknGlobal)
#define clsTexteditAppMonitor MakeWKN(357,1,wknGlobal)
```

# Networking

```c
#define clsTransport MakeWKN(88,1,wknGlobal)
#define clsLink MakeWKN(394,1,wknGlobal)
#define clsHighSpeedPacket MakeWKN(438,1,wknGlobal)
#define clsALAPHighSpeed MakeWKN(417,1,wknGlobal)
#define clsATP MakeWKN(89,1,wknGlobal)
#define clsATPHandle MakeWKN(318,1,wknGlobal)
#define theATPDriver MakeWKN(319,1,wknGlobal)
#define clsSoftTalk MakeWKN(119,1,wknGlobal)
```
#define theSoftTalkDriver MakeWRN(86,1,wknGlobal)
#define clsTopsMounter MakeWRN(116,1,wknGlobal)
#define theTopsMounter MakeWRN(118,1,wknGlobal)
#define theTopsService MakeWRN(345,1,wknGlobal)
#define clsTOPS MakeWRN(400,1,wknGlobal)
#define theTopsVolumes MakeWRN(401,1,wknGlobal)
#define theTopsPrinters MakeWRN(402,1,wknGlobal)
#define theRemoteServices MakeWRN(403,1,wknGlobal)

Selection and Data Transfer
#define theSelectionManager MakeWRN(111,1,wknGlobal)
#define clsSelection MakeWRN(111,1,wknGlobal)
#define clsXfer MakeWRN(139,1,wknGlobal)
#define clsXferList MakeWRN(322,1,wknGlobal)
#define clsPipe MakeWRN(63,1,wknGlobal) // PenPoint internal

Timer
#define theTimer MakeWRN(109,1,wknGlobal)
#define clsTimer MakeWRN(109,1,wknGlobal)

Preferences
#define theSystemPreferences MakeWRN(324,1,wknGlobal)
#define clsPreferences MakeWRN(323,1,wknGlobal)
#define clsPrefApp MakeWRN(115,1,wknGlobal)
#define clsPrefSheet MakeWRN(216,1,wknGlobal)

Power Management
#define clsPowerButton MakeWRN(348,1,wknGlobal)
#define thePowerButton MakeWRN(348,1,wknGlobal)
#define clsPowerMgr MakeWRN(416,1,wknGlobal)
#define thePowerMgr MakeWRN(416,1,wknGlobal)

Send and Address Book Managers
#define clsAddressBook MakeWRN(346,1,wknGlobal)
#define theAddressBookMgr MakeWRN(342,1,wknGlobal)
#define theSendManager MakeWRN(341,1,wknGlobal)

Spell Manager
#define theSpellManager MakeWRN(380,1,wknGlobal)
#define clsSpellManager MakeWRN(200,1,wknGlobal)
#define clsSpellField MakeWRN(386,1,wknGlobal)
#define theProcessSpellManager MakeWRN(2,1,wknProcessGlobal)

Personal Dictionary
#define clsPDict MakeWRN(328,1,wknGlobal)
#define thePersonalDictionary MakeWRN(329,1,wknGlobal)
#define clsPDApp MakeWRN(336,1,wknGlobal) // obsolete
#define clsPDUI MakeWRN(336,1,wknGlobal) // Replaces clsPDApp
Printer Drivers

```c
#define clsPrt
#define clsBndPrt
#define clsEpson
#define clsPcl
#define clsPscript
#define clsFaxPrt
#define clsPrtUI
#define clsRemora
```

```c
MakeWKN(201,1,wknGlobal)
MakeWKN(202,1,wknGlobal)
MakeWKN(203,1,wknGlobal)
MakeWKN(204,1,wknGlobal)
MakeWKN(205,1,wknGlobal)
MakeWKN(245,1,wknGlobal)
MakeWKN(91,1,wknGlobal)
MakeWKN(364,1,wknGlobal)
```

Handwriting Customization

```c
#define clsHWCustFrame
#define clsPlatoHomeWin
#define clsPlato26Win
#define clsPlato26WinKbd
#define clsPlatoCustomStat
#define clsPlatoBox
```

```c
MakeWKN(316,1,wknGlobal)
MakeWKN(347,1,wknGlobal)
MakeWKN(334,1,wknGlobal)
MakeWKN(339,1,wknGlobal)
MakeWKN(362,1,wknGlobal)
MakeWKN(232,1,wknGlobal)
```

Letter & Gesture Practice

```c
#define clsHWLetterFrame
#define clsHWLetterWin
#define clsHWLetterKbd
#define clsHWLetterBkg
#define clsHWGestFrame
#define clsHWGestWin
#define clsHWGestPracWin
```

```c
MakeWKN(146,1,wknGlobal)
MakeWKN(170,1,wknGlobal)
MakeWKN(390,1,wknGlobal)
MakeWKN(404,1,wknGlobal)
MakeWKN(410,1,wknGlobal)
MakeWKN(411,1,wknGlobal)
```

Animator

```c
#define clsAnimSPaper
#define clsAnimSysDc
```

```c
MakeWKN(234,1,wknGlobal)
MakeWKN(81,1,wknGlobal)
```

Inbox / Outbox /Wrapper

```c
#define clsOutboxSectApp
#define clsOBXService
#define clsOBXWin
#define clsOBXService
#define clsOBXWrapperApp
#define clsPrintWrapperApp
#define clsPrnInstlApp
#define clsINBXSectApp
#define clsINBXService
#define clsINBXService
#define clsINBXWin
#define clsINBXWin
#define clsTPSPSvc
#define clsTPnMgr
#define theTopsPSPManager
#define clsOBXBroWin
#define clsINBOXBrowWin
#define clsOBXStatusWin
#define theOutboxServices
#define theInboxServices
```

```c
MakeWKN(272,1,wknGlobal)
MakeWKN(273,1,wknGlobal)
MakeWKN(274,1,wknGlobal)
MakeWKN(395,1,wknGlobal)
MakeWKN(113,1,wknGlobal)
MakeWKN(114,1,wknGlobal)
MakeWKN(113,1,wknGlobal)
MakeWKN(129,1,wknGlobal)
MakeWKN(130,1,wknGlobal)
MakeWKN(131,1,wknGlobal)
MakeWKN(149,1,wknGlobal)
MakeWKN(173,1,wknGlobal)
MakeWKN(212,1,wknGlobal)
MakeWKN(429,1,wknGlobal)
MakeWKN(430,1,wknGlobal)
```
Mask App

#define clsMaskApp MakeWKN(327,1,wknGlobal)
#define clsMaskAppMonitor MakeWKN(325,1,wknGlobal)

Clock App

#define clsClockApp MakeWKN(165,1,wknGlobal)
#define clsClockLabel MakeWKN(220,1,wknGlobal)
#define clsClockWin MakeWKN(223,1,wknGlobal)

Note Icon Window (used in Clock App)

#define clsNoteIconWin MakeWKN(166,1,wknGlobal)

Miscellaneous

#define clsString MakeWKN(108,1,wknGlobal)
#define clsByteBuf MakeWKN(185,1,wknGlobal)

Test Support

#define clsTestNB MakeWKN(65,1,wknGlobal)

The MIL

#define theMIL MakeWKN(213,1,wknGlobal)
#define theMILMachineType MakeWKN(215,1,wknGlobal)
#define theMILUnitTag MakeWKN(227,1,wknGlobal)

MIL device ids, and the classes of the MIL services for these devices.

#define clsMILBaseDevice MakeWKN(214,1,wknGlobal)
#define clsMILInitDevice MakeWKN(229,1,wknGlobal)
#define clsMILPowerDevice MakeWKN(230,1,wknGlobal)
#define clsMILTimerDevice MakeWKN(231,1,wknGlobal)
#define clsMILRealTimeClockDevice MakeWKN(233,1,wknGlobal)
#define clsMILInterruptDevice MakeWKN(238,1,wknGlobal)
#define clsMILScreenDevice MakeWKN(247,1,wknGlobal)
#define clsMILStylusDevice MakeWKN(248,1,wknGlobal)
#define clsMILSoundDevice MakeWKN(259,1,wknGlobal)
#define clsMILKeyboardDevice MakeWKN(262,1,wknGlobal)
#define clsMILAsyncSIODevice MakeWKN(263,1,wknGlobal)
#define clsMILParallelPortDevice MakeWKN(264,1,wknGlobal)
#define clsMILAppleLAPDevice MakeWKN(265,1,wknGlobal)
#define clsMILNVMemDevice MakeWKN(267,1,wknGlobal)
#define clsMILSCSIDevice MakeWKN(269,1,wknGlobal)
#define clsMILFlashDevice MakeWKN(270,1,wknGlobal)
#define clsMILCompressionDevice MakeWKN(276,1,wknGlobal)
#define clsMILDebugDevice MakeWKN(277,1,wknGlobal)
#define clsMILBlockDevice MakeWKN(287,1,wknGlobal)
#define clsMILDiskDevice MakeWKN(301,1,wknGlobal)
#define clsMILDisketteDevice MakeWKN(304,1,wknGlobal)
#define clsMILLSPacketDevice MakeWKN(305,1,wknGlobal)
#define clsMILMemoryCardDevice MakeWKN(306,1,wknGlobal)
#define clsMILHSPacketDevice MakeWKN(435,1,wknGlobal)

These device Ids may be used for temporary testing of new device types. Code using these device types SHOULD NEVER BE RELEASED.

#define clsMILTest1Device MakeWKN(307,1,wknGlobal)
#define clsMILTest2Device MakeWKN(315,1,wknGlobal)
#define clsMILTest3Device MakeWKN(317,1,wknGlobal)
Predefined conflict group uids.

```plaintext
#define theMILConflictGroup1 MakeWKN(418, 1, wknGlobal)
#define theMILConflictGroup2 MakeWKN(419, 1, wknGlobal)
#define theMILConflictGroup3 MakeWKN(420, 1, wknGlobal)
#define theMILConflictGroup4 MakeWKN(421, 1, wknGlobal)
#define theMILConflictGroup5 MakeWKN(422, 1, wknGlobal)
#define theMILConflictGroup6 MakeWKN(423, 1, wknGlobal)
#define theMILConflictGroup7 MakeWKN(424, 1, wknGlobal)
#define theMILConflictGroup8 MakeWKN(425, 1, wknGlobal)
#define theMILConflictGroup9 MakeWKN(426, 1, wknGlobal)
#define theMILConflictGroup10 MakeWKN(427, 1, wknGlobal)
```

**The Connections Notebook**

```plaintext
#define clsConnectionsUI MakeWKN(365, 1, wknGlobal)
#define clsCNBSheet MakeWKN(366, 1, wknGlobal)
#define clsConnections MakeWKN(367, 1, wknGlobal)
#define clsPrinterView MakeWKN(368, 1, wknGlobal)
#define clsPrinterViewCV MakeWKN(495, 1, wknGlobal)
#define clsColumnView MakeWKN(369, 1, wknGlobal)
#define theConnections MakeWKN(370, 1, wknGlobal)
#define theVolumeServices MakeWKN(371, 1, wknGlobal)
#define thePrinterServices MakeWKN(372, 1, wknGlobal)
#define theConnectionsMenu MakeWKN(441, 1, wknGlobal)
#define clsNetView MakeWKN(373, 1, wknGlobal)
#define clsNetVolumeView MakeWKN(374, 1, wknGlobal)
#define clsNetPrinterView MakeWKN(375, 1, wknGlobal)
#define clsTOPSUI MakeWKN(376, 1, wknGlobal)
#define clsConnectionsUIAppWin MakeWKN(377, 1, wknGlobal)
```

**The Databases World**

```plaintext
#define theDatabases MakeWKN(405, 1, wknGlobal)
#define clsDbService MakeWKN(406, 1, wknGlobal)
#define clsDBCConnections MakeWKN(407, 1, wknGlobal)
#define clsDatabasesView MakeWKN(408, 1, wknGlobal)
#define clsDatabasesViewCV MakeWKN(496, 1, wknGlobal)
#define clsTechGnosis MakeWKN(437, 1, wknGlobal)
```

**The Hard Disk Installer**

```plaintext
#define clsHardinst MakeWKN(225, 1, wknGlobal)
#define theHardinst MakeWKN(436, 1, wknGlobal)
```

**The Symbolic Debugger**

```plaintext
#define theDebugger MakeWKN(358, 1, wknGlobal)
#define clsDebugger MakeWKN(358, 1, wknGlobal)
```

**The ASP/AFP & AppleTalk Related Defines**

```plaintext
#define clsASP MakeWKN(444, 1, wknGlobal)
#define clsASPClient MakeWKN(445, 1, wknGlobal)
#define clsASPServer MakeWKN(446, 1, wknGlobal)
#define clsASPServerSessionHandler MakeWKN(447, 1, wknGlobal)
#define clsVoLAFF MakeWKN(448, 1, wknGlobal)
#define clsAFP MakeWKN(449, 1, wknGlobal)
#define clsAfpMounter MakeWKN(450, 1, wknGlobal)
#define theAfpMounter MakeWKN(451, 1, wknGlobal)
```
#define theSessionHandlers MakeWKN(452,1,wknGlobal)
#define clsASPClientService MakeWKN(453,1,wknGlobal)
#define clsASPServerService MakeWKN(454,1,wknGlobal)
#define theAfpService MakeWKN(455,1,wknGlobal)
#define theAfpVolumes MakeWKN(456,1,wknGlobal)
#define clsAFPUI MakeWKN(493,1,wknGlobal)
#define clsPcTest MakeWKN(497,1,wknGlobal)
#define thePcTest MakeWKN(498,1,wknGlobal)
#define thePublicFileTypes MakeWKN(499,1,wknGlobal)
Part 2 / PenPoint Application Framework
APP.H

This file contains the API definition for clsApp. The functions described in this file are contained in APP.LIB.

clsApp inherits from clsObject.

Provides the standard behavior for a PenPoint application.

Introduction

PenPoint applications rely on clsApp to create and display their main window, save state, terminate the application instance, and so on. Every application developer needs to create a descendant of clsApp and have the descendant handle a few important messages. See clsTemplateApp in \penpoint\sdk\sample\templtap for an example of those messages an application typically must handle.

When the user turns to a document in the notebook, the PenPoint Application Framework creates an application instance to manage that document. Throughout this header file and the rest of our documentation, we use the term "document" to refer to an instance of an application class.

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

Common #defines and typedefs

typedef OBJECT APP, *P_APP;
#define AppDebug(v, e) DbgFlag('R', v, e)

Well-known Filenames

The Application Framework looks for information and stores document data in a series of well-known filenames. One of these is:

- appResFileName, the application's resource file for its icons, quick help, user interface strings, and so on.

Each document in the Notebook has its own directory, containing a collection of files for the document's data and subdirectories for any embedded documents. These are:

- appDocStateFileName, the resource file for any objects that the document saves. In general, this is called the document's resource file
- appDocResFileName, a resource file for preferences, including print metrics (once they are changed from the defaults) and comments that the user wrote in the "Comments" option sheet
- appDocLinkFileName, the document's saved Reference Buttons and descriptors for what they are linked to
appActiveDocLinkFileName, a working document of newly created (but not yet saved) Reference Buttons
appCorkboardDirName, the name of the subdirectory for documents embedded on the document's corkboard
subdirectories for any other embedded documents.

#define appResFileName "APP.RES"
#define appDocStateFileName "DOCSTATE.RES"
#define appDocResFileName "DOC.RES"
#define appDocLinkFileName "DOC.LNK"
#define appActiveDocLinkFileName "ACTDOC.LNK"
#define appCorkboardDirName "CORKBD"

Status Codes

These are the status codes returned by clsApp.

#define stsAppRefused MakeStatus(clsApp, 1)
#define stsAppMoveRCAppToCApp MakeStatus(clsApp, 2)
#define stsAppMoveCAppToInvalid MakeStatus(clsApp, 3)
#define stsAppCopyRCAppToCApp MakeStatus(clsApp, 13)
#define stsAppCopyCAppToInvalid MakeStatus(clsApp, 14)
#define stsAppNotMovable MakeStatus(clsApp, 4)
#define stsAppNotCopyable MakeStatus(clsApp, 5)
#define stsAppNotDeletable MakeStatus(clsApp, 6)
#define stsAppDuplicateName MakeStatus(clsApp, 7)
#define stsAppBadName MakeStatus(clsApp, 17)
#define stsAppNotFound MakeStatus(clsApp, 8)
#define stsAppOpened MakeStatus(clsApp, 9)
#define stsAppNoSelection MakeStatus(clsApp, 10)
#define stsAppSelRequestNotSupported MakeStatus(clsApp, 11)
#define stsAppOutOfMemory MakeStatus(clsApp, 15)
#define stsAppCrashed MakeStatus(clsApp, 16)
#define stsAppOpenFailedSuppressError MakeStatus(clsApp, 18)
#define stsAppErrorStartingDoc MakeStatus(clsApp, 19)
#define stsAppErrorEmbedPrintApply MakeStatus(clsApp, 20)
#define stsAppErrorLeftPrintMargin MakeStatus(clsApp, 21)
#define stsAppErrorRightPrintMargin MakeStatus(clsApp, 22)
#define stsAppErrorTopPrintMargin MakeStatus(clsApp, 23)
#define stsAppErrorBottomPrintMargin MakeStatus(clsApp, 24)
#define stsAppErrorHeaderPrintMargin MakeStatus(clsApp, 25)
#define stsAppErrorFooterPrintMargin MakeStatus(clsApp, 26)

Document States

A document can be in one of three states. When the user opens a document, its state becomes appOpened. Once the user closes it, the document's state can be either appTerminated or appActivated.

There are conditions when, after the user closes a document, the document's objects needs to stay around (and not be freed). Such conditions include when the document's access speed is set to accelerated (a.k.a., "hot mode") and when the document owns the selection. If a document is closed but needs to stay active, its state is set to appActivated. If there is no reason to keep a document around after it has been closed, its state becomes appTerminated (and the document is freed soon thereafter).

You can specify additional conditions for keeping a closed document active by handling msgAppTerminateOK. See the description of this message for further details.

#define appTerminated 0 // closed doc, on its way to being freed
#define appActivated 1 // closed doc, with a reason to stay active
#define appOpened 2 // opened doc
## App toggle

These are toggles used as parameters to various messages.

```c
#define appOff 0
#define appOn   1
#define appToggle 2
```

## Printing Flags

The Application Framework uses these flags when opening a document to print it and its embedded documents. The typical application developer does not need to use these flags. However, if you open your own embedded documents, you should never pass on `appPrintingTopLevel` to them (even if you were opened with `appPrintingTopLevel` set).

```c
#define appPrinting    ((U16)flag0)
#define appPrintingTopLevel ((U16)flag1)
```

## App Flags

This structure defines the application flags. They include the state of the document (see Document States above) and other common booleans. This structure is used in `APP_METRICS` and by `APP_DIR_FLAGS` (defined in `appdir.h`).

```c
typedef struct APP_FLAGS {  
    U16 state        : 2;  // Document state.
    U16 hotMode      : 1;  // True = app is in hot mode.
    U16 floating     : 1;  // True = app is floating.
    U16 printing     : 1;  // True = app is printing.
    U16 topLevel     : 1;  // True = app is printing as top level.
    U16 reserved1    : 10; // Reserved.
    U16 reserved2    : 16; // Reserved.
} APP_FLAGS, *P_APP_FLAGS;
```

## App Metrics

This structure defines the public instance data for `clsApp`. You get a copy of this structure when you send `msgAppGetMetrics` to an application object. The fields of `APP_METRICS` are as follows:

- **uuid**: The document's uuid. It is stamped as an attribute on the document directory (see `appdir.h`). You can pass a document's uuid to `clsDirHandle` or `clsAppDir` in `msgNew` to create a handle to the document directory.
- **dir**: An instance of `clsAppDir`. It is the handle to the filesystem directory for a document.
- **parent**: An instance of `clsApp`. A document's parent is the document that activated it (see `appmgr.h` - `msgAppMgrActivate`). If the user opens a document from the Notebook, the Notebook is the parent. If the opened document is embedded within another document, its parent is the embedder.
- **children**: An instance of `clsObject`. This represents a list of the documents that this document activated. There is often a one-to-one correspondence between a document's children and its embedded documents.
- **mainWin**: The document's main window. If this field is `objNull` when a document receives `msgAppInit`, the document self sends `msgAppProvideMainWin` to create one.
- **floatingWins**: An instance of `clsList`. It is the list of subordinate windows that are floating above a document (e.g., option sheets). See `msgAppAddFloatingWin` and `msgAppRemoveFloatingWin` for more info.
childAppParentWin: The preferred parent window for embedded documents.

resList: An instance of clsResList. It is list of clsResFile objects. The default list consists of (1) a document resource file, (2) an application resource file, (3) a preference resource file, and (4) the system resource file. See resfile.h for more details.

resFile: The document’s resource file (the same one as in the resList).

flags: Various flags for the document. See the discussion of APP_FLAGS given above.

define struct APP_METRICS {
    UUID uuid;
    OBJECT dir; // App directory.
    OBJECT parent; // Parent app.
    OBJECT children; // Child apps observe this object.
    OBJECT mainWin; // App main window.
    OBJECT floatingWins; // List of floating windows.
    OBJECT childAppParentWin; // Root of child app window tree.
    OBJECT resList; // Resource file list.
    OBJECT resFile; // Document resource file.
    U32 reserved[2]; // Reserved.
    APP_FLAGS flags; // Flags.
} APP_METRICS, *P_APP_METRICS;

### Enabling and Disabling SAMs

In its handler for msgAppCreateMenuBar, clsApp creates several menus and menu items that are part of PenPoint’s standard user interface. These menus and items are known collectively as PenPoint’s Standard Application Menus,” or “SAMs” for short. The SAMs are identified by tags in apptag.h and are described in the PenPoint User Interface Design Reference.

In many cases, descendants of clsApp should be involved in deciding when the SAM menu items should be enabled or disabled. Sometimes a descendant should completely remove an item from the SAM.

To enable and disable the SAM items, clsApp handles msgControlProvideEnable (see control.h for a description this message). Specifically, clsApp:

- **Always enables:**
  
  `tagAppMenuPrintSetup`
  `tagAppMenuAbout`
  `tagAppMenuCheckpoint`
  `tagAppMenuRevert`
  `tagAppMenuSearch`
  `tagAppMenuSpell`

- **Enables this if the UndoManager has transactions to undo (see undo.h):**
  
  `tagAppMenuUndo`

- **Asks the appropriate manager to enable or disable:**
  
  `tagAppMenuPrint`
  `tagAppMenuSend`

- **Always disables:**
  
  `tagAppMenuSelectAll`
  any unrecognized tag

Here are some examples of how descendants might want to modify the SAMs or respond to msgControlProvideEnable:

- **Most applications should support all of the features in the SAMs. (That’s why they’re part of PenPoint’s standard UI.) But for a variety of reasons, some applications won’t support some**
Applications should handle msgControlProvideEnable and return false if there's no data in the application, true otherwise, for:

- tagAppMenuSelectAll

Selection owners should respond to msgControlProvideEnable for tagAppMenuMove, tagAppMenuCopy and tagAppMenuDelete. Here are some notes on the proper response.

- If there is no data selected, then all three items should be disabled.
- If the application data is read-only, Move and Delete should be disabled.
- In most other cases, the item should be enabled.

### Messages

#### msgNew

Creates and initializes a new document.

Takes P_APP_NEW, returns STATUS. Category: class message.

```c
typedef struct APP_NEW ONLY {
    FS_LOCATOR locator;       // Document's location in the filesystem.
    OBJECT winDev;            // Window device.
    BOOLEAN appMonitor;       // True if app monitor instance.
    U16 reserved1;            // Reserved.
    U32 reserved2[4];         // Reserved.
} APP_NEW ONLY, *P_APP_NEW ONLY;
#define appNewFields

typedef struct APP_NEW {
    appNewFields
} APP_NEW, *P_APP_NEW;
```

**Arguments**

classApp initializes the new document's instance data to default values.

You should never send msgNew directly to classApp or its descendants. Sending msgNew is not sufficient to create a viable document. The document must have its own process and directory, which msgNew does not create. To create a viable document, send msgAppMgrCreate (or msgAppMgrCopy) followed by msgAppMgrActivate to the app's application manager. (Remember that the application manager's uid is the well-known uid for the application class.)

Descendants: You should never handle msgNew directly. Instead, handle msgInit by initializing your instance data. The ancestor must be called before your msgInit handler.
### msgNewDefaults
Initializes an APP_NEW structure to default values.

Takes P_APP_NEW, returns STATUS. Category: class message.

**Message**
```
typedef struct APP_NEW {
  appNewFields
} APP_NEW, *P_APP_NEW;
```

**Arguments**
Zeroes out pArgs->app.

**Comments**
Descendants: You should handle `msgNewDefaults` by initializing your _NEW structure to default values. The ancestor must be called before your handler.

### msgFree
Destroys a document.

Takes nothing, returns STATUS.

**Comments**
The document frees its instance data, its children, its main window, and any option sheets it has created. Its final step is to kill its process, which means that flow of control never returns from this message handler.

Descendants: You should handle `msgFree` by destroying all objects and resources you have created. The ancestor must be called after your handler.

### msgFreeOK
Checks to see if a document and its children are willing to be freed.

Takes nothing, returns STATUS.

**Comments**
This message is self sent as a result of `msgDestroy` being sent to the document.

A document can be freed if it can be terminated (see above description of Document States). To determine if it can be terminated, the document self sends `msgAppTerminateOK`; if this message returns **stsOK**, the document then sends `msgFreeOK` to each active child document (those on the metrics.children list). If all of the children return **stsOK**, then the document can be terminated.

Descendants: You normally do not handle this message. Instead, handle `msgAppTerminateOK`.

**Return Value**
- **stsOK**: If the document can be terminated.
- **stsAppRefused**: If the document should not be terminated.

### msgAppActivate
Activates a document and its children.

Takes nothing, returns STATUS.

**Comments**
This message prepares an application to receive such requests as becoming available to the user (`msgAppOpen`) and searching for some data (`msgAppSearch`).

Descendants: You normally do not handle this message.
**msgAppInit**

Creates a document's default data file and main window.

Takes DIR_HANDLE, returns STATUS.

#define msgAppInit MakeMsg(clsApp, 2)

Comments

This message is sent the first time a document is activated. It performs one-time initializations.

If the main window is objNull, the document creates the main window by self sending msgAppProvideMainWin. If childAppParentWin is objNull, the document sets it to be the main window. The document also sets the main window title by self sending msgAppGetName, followed by msgAppSetName.

Descendants: You should handle this message by performing one-time initializations. This typically means creating any stateful objects that will be filed. The ancestor should be called before your handler.

**msgAppRestore**

Restores a document from its saved instance data.

Takes nothing, returns STATUS.

#define msgAppRestore MakeMsg(clsApp, 3)

Comments

The document opens its resource file (appDocStateFileName), reads its instance data, and closes the file. When it receives msgRestore, the document reads its main window from the file.

Descendants: You normally do not handle this message. Instead, you should handle msgRestore (which is sent as a result of this message).

**msgAppRestoreFrom**

Restores a document from a specified directory.

Takes DIR_HANDLE, returns STATUS.

#define msgAppRestoreFrom MakeMsg(clsApp, 4)

Comments

This message is just like msgAppRestore, except the document opens the resource file (appDocStateFileName) located in DIR_HANDLE.

Descendants: You normally do not handle this message. Instead, you should handle msgRestore (which is sent as a result of this message).

**msgAppSave**

Saves a document to its working directory.

Takes nothing, returns STATUS.

#define msgAppSave MakeMsg(clsApp, 5)

Comments

The document self sends msgAppSaveChildren to save its children. Next, the document opens its resource file (appDocStateFileName), writes its instance data, and closes the file. The document also saves its link file. When it receives msgSave, the document writes its main window to the file.

Descendants: You normally do not handle this message. Instead, you should handle msgSave to save your instance data.
**msgAppSaveTo**
Saves a document to a specified directory.
Takes DIR_HANDLE, returns STATUS.

```c
#define msgAppSaveTo MakeMsg(clsApp, 6)
```

Comments
This message is just like msgAppSave, except the document opens the resource file (appDocStateFileName) located in DIR_HANDLE.

Descendants: You normally do not handle this message. Instead, you should handle msgSave to save your instance data.

---

**msgAppSaveChildren**
Saves a document's children.
Takes nothing, returns STATUS.

```c
#define msgAppSaveChildren MakeMsg(clsApp, 7)
```

Comments
The document self sends msgAppSaveChild to save each child document.

Descendants: You normally do not handle this message.

---

**msgAppSaveChild**
Saves the specified child document.
Takes APP, returns STATUS.

```c
#define msgAppSaveChild MakeMsg(clsApp, 97)
```

Comments
The document sends msgAppSave to APP.

Descendants: You normally do not handle this message.

---

**msgAppOpen**
Opens a document's main window.
Takes P_APP_OPEN, returns STATUS.

```c
#define msgAppOpen MakeMsg(clsApp, 8)
```

Arguments
typedef struct APP_OPEN {
  OBJECT parentWin;  // Document's parent window.
  OBJECT childAppParentWin;  // out: Parent window for child apps.
  U16 printing;  // in: See printing flags.
} APP_OPEN, *P_APP_OPEN;

Comments
If the document's main window has not been sized, the document sets it to the default size. It also updates the 'parentWin' and 'childAppParentWin' fields in the application metrics. The document then sets its state to appOpened and self sends msgAppOpenChildren to open its child documents.

This message is sent to the document when it is to be made available to the user for direct interaction.

Descendants: You should handle this message by creating any non-stateful objects that are necessary to display the document's UI. You should also fill in 'childAppParentWin' - normally with the document's client window.
You typically create the menu bar in response to this message. Self send `msgAppCreateMenuBar` to create the menu bar, and then send `msgFrameSetMetrics` to your main window to insert the menu bar in the window.

If you can't open the document, you should return `staFailed`. However, if you have already displayed an error message to the user, then return `staAppOpenFailedSupressError`.

The ancestor should be called after your handler.

**msgAppClose**

Closes a document's main window.

Takes nothing, returns STATUS.

```c
#define msgAppClose MakeMsg(clsApp, 9)
```

Comments

The document extracts its main window from the window tree. It then sets the 'parentWin' field in the application metrics to `objNull` and sets its state to `appActivated`. To close its children, it self sends `msgAppCloseChildren`.

Descendants: You should handle this message by destroying any objects that you created in `msgAppOpen`. If you created the menu bar in your `msgAppOpen` handler, then you should send `msgFrameDestroyMenuBar` to your main window. The ancestor should be called before your handler.

This message is not an indication to terminate the document; it may be followed by other requests for services such as searching or re-opening.

**msgAppSetMainWin**

Specifies a document's main window.

Takes WIN, returns STATUS.

```c
#define msgAppSetMainWin MakeMsg(clsApp, 10)
```

Comments

The document updates its metrics. `mainWin` field to point to `pArgs`. It does not destroy the existing `mainWin`.

Descendants: You normally do not handle this message.

**msgAppSetChildAppParentWin**

Specifies the window that is used as the parent window for child documents.

Takes WIN, returns STATUS.

```c
#define msgAppSetChildAppParentWin MakeMsg(clsApp, 11)
```

Comments

Descendants: You normally do not handle this message.

**msgAppGetMetrics**

Passes back a copy of the application metrics.

Takes `P_APP_METRICS`, returns STATUS.

```c
#define msgAppGetMetrics MakeMsg(clsApp, 12)
```

Message Arguments

typedef struct APP_METRICS {
  UUID uuid;
  OBJECT dir;
  OBJECT parent;
} // App uuid.
  // App directory.
  // Parent app.
OBJECT children;  // Child apps observe this object.
OBJECT mainWin;   // App main window.
OBJECT floatingWins;  // List of floating windows.
OBJECT childAppParentWin;  // Root of child app window tree.
OBJECT resList;  // Resource file list.
OBJECT resFile;  // Document resource file.
U32 reserved[2];  // Reserved.
APP_FLAGS flags;  // Flags.
} APP_METRICS, *P_APP_METRICS;

Comments
Descendants: You normally do not handle this message.

msgAppDispatch
Starts message dispatching.
Takes nothing, returns STATUS.
#define msgAppDispatch MakeMsg(clsApp, 13)

Comments
Descendants: You normally do not handle this message.

msgAppRename
Renames a document.
Takes P_STRING, returns STATUS.
#define msgAppRename MakeMsg(clsApp, 14)

Comments
After msgAppRename is sent to the document, the Application Framework sends msgAppSetName to
change the document's window title.

Descendants: You normally do not handle this message. Instead, you might want to handle
msgAppSetName.

msgAppSetName
Specifies a document's displayed name (in its main window title).
Takes P_STRING, returns STATUS.
#define msgAppSetName MakeMsg(clsApp, 15)

Comments
This message does not actually rename the document; it only sets the title of the document's main
window. This message is sent to a document after it receives msgAppRename, which does rename the
document.

Descendants: You can handle this message by changing or adding to the string passed in. The ancestor
will take the new string and display it in the document's title. The ancestor must be called after your
handler.

msgAppGetName
Passes back a document's name.
Takes P_STRING, returns STATUS.
#define msgAppGetName MakeMsg(clsApp, 16)

Comments
The document passes back its name (not its main window's title). Note that P_STRING must be
nameBufLength long.

Descendants: You normally do not handle this message.
msgAppDelete
Deletes a document from the system.
Takes nothing, returns STATUS.

#define msgAppDelete MakeMsg(clsApp, 17)

Comments
The document deletes its appWin from its embeddor and sends msgAppMgrDelete to the document's class.

Descendants: You normally do not handle this message.

Return Value
stsAppRefused If metrics.flags.deletable is false.

msgAppActivateChildren
Activates a document's embedded documents.
Takes nothing, returns STATUS.

#define msgAppActivateChildren MakeMsg(clsApp, 18)

Comments
The document first activates the embedded documents that are stored in subdirectories of metrics.dir by self sending msgAppActivateChild for each child. It then self sends msgAppActivateCorkMarginChildren to activate the embedded documents that appear in the cork margin.

Descendants: You normally do not handle this message.

msgAppActivateCorkMarginChildren
Activates embedded documents that are in a document's cork margin.
Takes nothing, returns STATUS.

#define msgAppActivateCorkMarginChildren MakeMsg(clsApp, 96)

Comments
The document self sends msgAppActivateChild for each embedded document in the cork margin.

Descendants: You normally do not handle this message.

msgAppActivateChild
Instantiates and activates an embedded document.
Takes P_APP_ACTIVATE_CHILD, returns STATUS.

#define msgAppActivateChild MakeMsg(clsApp, 19)

Arguments
typedef struct APP_ACTIVATE_CHILD {
P_STRING pPath;       // Path of child relative to self.
APP uid;             // out: Child app uid.
} APP_ACTIVATE_CHILD, *P_APP_ACTIVATE_CHILD;

Comments
This message sends msgAppMgrActivate to activate the specified embedded document.

Descendants: You normally do not handle this message.

Return Value
stsAppRefused If the child appDir.attrs.flags.disabled is true (see appdir.h).
**msgAppAddFloatingWin**

Adds a window to a document's list of floating windows.

Takes WIN, returns STATUS.

```c
#define msgAppAddFloatingWin MakeMsg(clsApp, 20)
```

Comments

Descendants: You normally do not handle this message.

---

**msgAppRemoveFloatingWin**

Removes a window from a document's list of floating windows.

Takes WIN, returns STATUS.

```c
#define msgAppRemoveFloatingWin MakeMsg(clsApp, 21)
```

Comments

Descendants: You normally do not handle this message.

---

**msgAppFindFloatingWin**

Finds the floating window on a document's list of floating windows that matches the specified tag.

Takes P_APP_FIND_FLOATING_WIN, returns STATUS.

```c
#define msgAppFindFloatingWin MakeMsg(clsApp, 22)
```

**Arguments**

```c
typedef struct APP_FIND_FLOATING_WIN {
    TAG tag; // in: tag to find.
    OBJECT win; // out: matching window, or objNull if not found.
} APP_FIND_FLOATING_WIN, *P_APP_FIND_FLOATING_WIN;
```

Comments

Descendants: You normally do not handle this message.

**Return Value**

- **stsOK** If the floating window is found
- **stsNoMatch** If the floating window cannot be found

---

**msgAppGetRoot**

Passes back a document's root document (which is typically the Notebook).

Takes P_APP, returns STATUS.

```c
#define msgAppGetRoot MakeMsg(clsApp, 23)
```

Comments

Descendants: You normally do not handle this message.

---

**msgAppSetParent**

Specifies a document's parent document.

Takes APP, returns STATUS.

```c
#define msgAppSetParent MakeMsg(clsApp, 24)
```

Comments

Descendants: You normally do not handle this message.
**msgAppSetHotMode**

Turns hot mode on or off for a document.

Takes BOOLEAN, returns STATUS.

```c
#define msgAppSetHotMode MakeMsg(clsApp, 25)
```

**Comments**

Descendants: You normally do not handle this message.

---

**msgAppSetReadOnly**

Specifies a document's read only flag.

Takes BOOLEAN, returns STATUS.

```c
#define msgAppSetReadOnly MakeMsg(clsApp, 26)
```

**Comments**

Descendants: You normally do not handle this message.

---

**msgAppSetDeletable**

Specifies a document's deletable flag.

Takes BOOLEAN, returns STATUS.

```c
#define msgAppSetDeletable MakeMsg(clsApp, 27)
```

**Comments**

Descendants: You normally do not handle this message.

---

**msgAppSetMovable**

Specifies a document's movable flag. Not implemented.

Takes BOOLEAN, returns STATUS.

```c
#define msgAppSetMovable MakeMsg(clsApp, 28)
```

**See Also**

`msgAppDirSetFlags`

---

**msgAppSetCopyable**

Specifies a document's copyable flag. Not implemented.

Takes BOOLEAN, returns STATUS.

```c
#define msgAppSetCopyable MakeMsg(clsApp, 29)
```

**See Also**

`msgAppDirSetFlags`

---

**msgAppTerminate**

Terminates a document.

Takes BOOLEAN, returns STATUS.

```c
#define msgAppTerminate MakeMsg(clsApp, 30)
```

**Comments**

If true is passed in, the document is given the chance to veto the termination. It does this by self sending `msgFreeOK` to see if it is okay to free the document. If it is okay, the document saves itself by self sending `msgAppSave`, and then frees itself by self sending `msgDestroy`.

If false is passed in, the document is not given the chance to veto. The document terminates itself and all of its children unconditionally.
Descendants: You normally do not handle this message. This message is a request, not a command, to terminate. It may be sent ANY number of times while a document is active. If you need to free objects when a document is terminated, you should handle msgFree. Furthermore, if you want to add conditions under which a document should not be terminated, handle msgAppTerminateOK.

msgAppOpenChildren
Opens all of the documents on a document's metrics.children list.
Takes BOOLEAN, returns STATUS.

```c
#define msgAppOpenChildren MakeMsg(clsApp, 31)
```

Comments
If false is passed in, the document opens its child documents on screen by self sending msgAppOpenChild for each child.
If true is passed in, it opens its child documents for printing as embedded documents.
Descendants: You normally do not handle this message.

msgAppOpenChild
Opens a specific child of a document.
Takes APP.OPEN_CHILD, returns STATUS.

```c
typedef struct APP_OPEN_CHILD {
    OBJECT app;           // Document to open.
    U16 printing;         // See printing flags.
} APP_OPEN_CHILD, *P_APP_OPEN_CHILD;
```

Arguments

Comments
Opens the specified child document by creating a window for it and then sending it msgAppOpen.
Descendants: You normally do not handle this message.

msgAppCloseChildren
Closes a document's children.
Takes nothing, returns STATUS.

```c
#define msgAppCloseChildren MakeMsg(clsApp, 89)
```

Comments
The document self sends msgAppCloseChild for each of its child documents.
Descendants: You normally do not handle this message.

msgAppCloseChild
Closes a specific child of a document.
Takes APP, returns STATUS.

```c
#define msgAppCloseChild MakeMsg(clsApp, 90)
```

Comments
The document closes the specified child document by sending it msgAppClose.
Descendants: You normally do not handle this message.
msgAppGetEmbeddor
Passes back a document's direct parent in the file system hierarchy.
Takes P_APP, returns STATUS.
#define msgAppGetEmbeddor MakeMsg(clsApp, 33)
Comments
The document finds its direct parent in the filesystem and passes back a pointer to it in P_APP. If the parent is not active, P_APP is set to null.
Descendants: You normally do not handle this message.

msgAppTerminateOK
Checks if a document is willing to terminate.
Takes nothing, returns STATUS.
#define msgAppTerminateOK MakeMsg(clsApp, 34)
Comments
The document self sends this message as a result of msgAppTerminate(true). The document refuses if:
(1) the document is opened, (2) the document is in hot mode, or (3) the document or any object in the document owns the selection.
Descendants: You should handle this message if you have your own conditions under which to veto document termination. Typically you call the ancestor first. If the ancestor returns stsAppRefused, then you also return this value. However, if your ancestor returns stsOK, you check for your veto conditions and return either stsOK or stsAppRefused.
Return Value
stsOK If the document can be terminated.
stsAppRefused If the document should not be terminated.

msgAppGetEmbeddedWin
Finds the specified clsEmbeddedWin object within a document.
Takes P_APP_GET_EMBEDDED_WIN, returns STATUS.
#define msgAppGetEmbeddedWin MakeMsg(clsApp, 35)
Arguments
typedef struct APP_GET_EMBEDDED_WIN {
    UUID  uuid;    // in: embedded win's uuid.
    OBJECT win;    // out: embedded win. Set to objNull if no match.
} APP_GET_EMBEDDED_WIN, *P_APP_GET_EMBEDDED_WIN;
Comments
The document recursively enumerates its children, searching for a clsEmbeddedWin object with a matching embeddedWinMetrics.uuid (see embedwin.h).
Descendants: You should handle this message only if you are managing embedded windows that are not in the main window's window tree. Typically you call the ancestor first. If the ancestor passes back a non-null win, then you don't need to do anything. However, if the ancestor passes back objNull for the win, you should check for a clsEmbeddedWin with a matching uuid.

msgAppGetAppWin
Finds a clsAppWin object within a document.
Takes P_APP_GET_APP_WIN, returns STATUS.
#define msgAppGetAppWin MakeMsg(clsApp, 36)
typedef struct APP_GET_APP_WIN {
    UUID uuid; // in: app win's uuid.
    OBJECT win; // out: app win. Set to objNull if no match.
} APP_GET_APP_WIN, *P_APP_GET_APP_WIN;

The document recursively enumerates its children, searching for a clsAppWin object with a matching appWinMetrics.appUUID (see appwin.h).

Descendants: You should handle this message only if you are managing embedded windows that are not in the main window's window tree. Typically you call the ancestor first. If the ancestor passes back a non-null win, then you don't need to do anything. However, if the ancestor passes back objNull for the win, you should check for a clsAppWin with a matching uuid.

msgAppOwnsSelection
Tests if any object in a document owns the selection.
Takes P_APP_OWNS_SELECTION, returns STATUS.

#define msgAppOwnsSelection MakeMsg(clsApp, 37)

typedef struct APP_OWNS_SELECTION {
    BOOLEAN checkChildren; // in: check child documents, too?
    BOOLEAN ownSelection; // out: true if doc(s) own the selection.
} APP_OWNS_SELECTION, *P_APP_OWNS_SELECTION;

The document sets ownSelection to true if the selection belongs to itself or one of its children (if checkChildren is true).

Descendants: You normally do not handle this message.

msgAppOpenTo
Opens a document to a specific state.
Takes U32, returns STATUS.

#define msgAppOpenTo MakeMsg(clsApp, 38)

#define appOpenToNormal 0 // Open a doc in place.
#define appOpenToFloating 1 // Open a doc to floating.
#define appOpenToNextState 2 // Goto next state. Not Implemented.

Comments
If appOpenToNormal is passed in, the document sends msgAppOpenChild to its parent to open itself. If appOpenToFloating is passed in, the document self sends msgAppFloat to open itself.

Descendants: You normally do not handle this message.

msgAppCloseTo
Closes a document to a specific state.
Takes U32, returns STATUS.

#define msgAppCloseTo MakeMsg(clsApp, 39)

#define appCloseToNormal 0 // Close to icon.
#define appCloseToNextState 1 // Close to next state.

Comments
Short description: you probably don't need to worry about this message.

Long description: When the user taps on an embedded document icon, the document opens. If the user then double taps on the embedded document's title bar, the embedded document floats above its parent (allowing the user to resize it, without changing the layout of the parent). When the user closes the
floating document, it "closes" to its next state (i.e., open, but not floating). Closing it again closes the embedded document down to its icon.

When the user closes an embedded document, the Application Framework sends the document msgAppCloseTo, passing it appCloseToNextState. However, the Application Framework needs a mechanism to close an embedded document all the way down to its icon (e.g., when the user closes the parent document). In such cases, the Application Framework sends msgAppCloseTo to the document, passing appCloseToNormal.

Descendants: You normally do not handle this message.

**msgAppHide**

Hides an open document.

Takes nothing, returns STATUS.

```c
#define msgAppHide MakeMsg(clsApp, 40)
```

Comments

This message is used to get a document and all its associated windows off the screen as quickly as possible. It is usually followed (via ObjectPost) by msgAppClose, which is a heavier-weight message.

The document (1) sends msgWinExtract to all windows in metrics.floatingWins, (2) sends msgWinExtract to metrics.mainWin, and (3) recursively sends msgAppHide to all documents on metrics.children.

Descendants: You should handle this message if you have visible windows that are not children of the main window or in the floating window list. The ancestor should be called after your handler.

**msgAppSetFloatingRect**

Specifies a document's floating size and position.

Takes P_RECT32, returns STATUS.

```c
#define msgAppSetFloatingRect MakeMsg(clsApp, 41)
```

Comments

Descendants: You normally do not handle this message.

**msgAppSetOpenRect**

Specifies a document's open size and position.

Takes P_RECT32, returns STATUS.

```c
#define msgAppSetOpenRect MakeMsg(clsApp, 42)
```

Comments

Descendants: You normally do not handle this message.

**msgAppGetOptionSheet**

Passes back the requested option sheet of a document.

Takes P_APP_GET_OPTION_SHEET, returns STATUS.

```c
#define msgAppGetOptionSheet MakeMsg(clsApp, 91)
```

Arguments

```c
typedef struct APP_GET_OPTION_SHEET {
    TAG sheetTag; // in: tag of option sheet.
    OBJECT sheet; // out: sheet uid.
} APP_GET_OPTION_SHEET, *P_APP_GET_OPTION_SHEET;
```
If the requested option sheet has already been created, the document just passes back its uid. Otherwise, it creates the sheet by self sending msgOptionCreateSheet. If the requested sheetTag is not tagAppDocOptSheet, the document self sends msgOptionAddCards to let descendants add option cards to the newly created sheet.

Descendants: You normally do not handle this message. If you want to add other cards to the document’s option sheets, you can handle msgAppAddCards.

**msgAppGetDocOptionSheetClient**

Passes back the client for a document’s option sheets.

Takes P_OBJECT, returns STATUS.

#define msgAppGetDocOptionSheetClient MakeMsg(clsApp, 93)

The document passes back its main window’s client window.

Descendants: You normally do not handle this message.

**msgAppAddCards**

Adds cards to the specified option sheet of a document.

Takes P_OPTION_TAG, returns STATUS.

#define msgAppAddCards MakeMsg(clsApp, 100)

If the specified sheet is tagAppAboutOptSheet, the document adds the "About Document" and "About Application" option cards to the sheet. If the sheet is tagAppDocOptSheet, the document adds the "Controls," "Access" and "Comments" cards. If the sheet is tagAppPrintSetupOptSheet, the document adds the "Print Layout" card.

Descendants: You tend not to handle this message. However, you can handle it if you want to add cards to any of the document’s option sheets.

**msgAppShowOptionSheet**

Shows or hides a document’s option sheet.

Takes P_APP_SHOW_OPTION_SHEET, returns STATUS.

#define msgAppShowOptionSheet MakeMsg(clsApp, 92)

typedef struct APP_SHOW_OPTION_SHEET {
  TAG       sheetTag;  // In: Option sheet tag.
  TAG       cardTag;   // In: Option card tag to initially show, or
                      // null to show the top card.
  BOOLEAN   show;      // In: true = show, false = hide.
  OBJECT    sheet;     // Out: option sheet.
} APP_SHOW_OPTION_SHEET, *P_APP_SHOW_OPTION_SHEET;

The Application Framework sends this message to show (or hide) any of a document’s option sheets. It is sent when, for example, the user picks any of the option cards from the SAMs or draws the check gesture on a document’s title, over a selection, or over an embedded document icon.

If show is true, the document self sends msgAppGetOptionSheet to get the requested option sheet. To display the sheet, the document sends msgOptionGetCards and msgOptionShowCardAndSheet to the sheet.

If show is false, the document self sends msgAppFindFloatingWin and msgAppRemoveFloatingWin to find and then hide the requested option sheet.

Descendants: You normally do not handle this message.
**msgAppApplyEmbeddeeProps**
Applies Embedded Printing option card values to first level embeddees.
Takes OBJECT, returns STATUS.

```c
#define msgAppApplyEmbeddeeProps MakeMsg(clsApp, 98)
```

Comments
Descendants: You normally do not handle this message.

**msgAppGetBorderMetrics**
Passes back a document's border metrics.
Takes P_APP_BORDER_METRICS, returns STATUS.

```c
#define msgAppGetBorderMetrics MakeMsg(clsApp, 94)
```

// Border styles
```c
#define appBorderNone 0
#define appBorderSingle 1
#define appBorderDouble 2
#define appBorderDashed 3
```

typedef struct APP_BORDER_METRICS {
    U16 controls   : 1; // Out: true/false.
    U16 titleLine  : 1; // Out: true/false.
    U16 menuLine   : 1; // Out: true/false.
    U16 corkMargin : 1; // Out: true/false.
    U16 scrollMargins : 1; // Out: true/false.
    U16 borderStyle : 4; // Out: Border style.
    U16 reserved   : 7;
} APP_BORDER_METRICS, *P_APP_BORDER_METRICS;

Comments
Descendants: You normally do not handle this message.

**msgAppSetControls**
Turns a document's controls on or off.
Takes U32, returns STATUS.

```c
#define msgAppSetControls MakeMsg(clsApp, 47)
```

Comments
If appOff is passed in, the document turns its controls off. If appOn is passed in, the controls are turned on. If appToggle is passed in, the document will toggle the state of the controls.

Descendants: You normally do not handle this message.

**msgAppSetPrintControls**
Turns a document's screen decorations off for printing.
Takes BOOLEAN, returns STATUS.

```c
#define msgAppSetPrintControls MakeMsg(clsApp, 99)
```

Comments
The document turns its controls off so that it can be printed. It leaves user-set borders on only if the document is printing itself as an embedded document (pArgs = false).

Descendants: You normally do not handle this message.
**msgAppSetTitleLine**

Turns a document's title line on or off.

Takes U32, returns STATUS.

```c
#define msgAppSetTitleLine MakeMsg(clsApp, 44)
```

Comments

If `appOff` is passed in, the document hides its title line. If `appOn` is passed in, the title line is displayed. If `appToggle` is passed in, the document toggles whether the title line is displayed.

Descendants: You normally do not handle this message.

---

**msgAppSetMenuLine**

Turns a document's menu bar on or off.

Takes U32, returns STATUS.

```c
#define msgAppSetMenuLine MakeMsg(clsApp, 45)
```

Comments

If `appOff` is passed in, the document hides its menu bar. If `appOn` is passed in, the menu bar is displayed. If `appToggle` is passed in, the document toggles whether the menu bar is displayed.

Descendants: You normally do not handle this message.

---

**msgAppSetCorkMargin**

Turns a document's cork margin on or off.

Takes U32, returns STATUS.

```c
#define msgAppSetCorkMargin MakeMsg(clsApp, 48)
```

Comments

If `appOff` is passed in, the document hides its cork margin. If `appOn` is passed in, the cork margin is created (if it doesn't exist) and displayed. If `appToggle` is passed in, the document toggles whether the cork margin is displayed.

Descendants: You normally do not handle this message.

---

**msgAppSetScrollBars**

Turns a document's scroll bars on or off.

Takes U32, returns STATUS.

```c
#define msgAppSetScrollBars MakeMsg(clsApp, 46)
```

Comments

If `appOff` is passed in, the document hides its scroll bars. If `appOn` is passed in, the scroll bars are displayed. If `appToggle` is passed in, the document toggles whether the scroll bars are displayed.

Descendants: You normally do not handle this message.

---

**msgAppSetBorderStyle**

Specifies the border style.

Takes U32, returns STATUS.

```c
#define msgAppSetBorderStyle MakeMsg(clsApp, 95)
```

Comments

The possible values for `pArgs` are listed above in `msgAppGetBorderMetrics`.

Descendants: You normally do not handle this message.
**msgAppRevert**
Reverts to the filed copy of a document.
Takes BOOLEAN, returns STATUS.

```c
#define msgAppRevert MakeMsg(clsApp, 49)
```

Comments
The document reverts to its previously saved state. If true is passed in, the document displays a note, asking the user to confirm the action first. If false is passed in, the document just does the action.

Descendants: If you do not support revert, you should handle this message by returning *stsAppRefused*. On the other hand, if you support revert but you manage your own data files or use memory mapped files, then it may be necessary to handle this message by appropriately undoing all data modifications since the last save. The ancestor should be called before your handler.

**msgAppIsPageLevel**
Asks a document if it shows up as a page in the Notebook (as opposed to being embedded).
Takes nothing, returns STATUS.

```c
#define msgAppIsPageLevel MakeMsg(clsApp, 50)
```

Comments
Descendants: You normally do not handle this message.

Return Value
- **stsOK** If the document is page-level (i.e., its embeddor inherits from *clsContainerApp* or *clsRootContainerApp*).
- **stsNoMatch** If the document is not page-level.

**msgAppProvideMainWin**
Asks a document to provide its main window.
Takes P_OBJECT, returns STATUS.

```c
#define msgAppProvideMainWin MakeMsg(clsApp, 51)
```

Comments
This message is sent during msgAppInit. If pArgs points to objNull, the document creates a default frame of type *dsFrame* and passes the frame's uid back in pArgs.

Descendants: You should handle this message if you want to replace the default *dsFrame* main window. In such cases, you tend not to call the ancestor.

See Also
- **msgAppCreateClientWin**

**msgAppCreateLink**
Creates a link from a document to another document.
Takes P_APP_LINK, returns STATUS.

```c
#define msgAppCreateLink MakeMsg(clsApp, 52)
```

Arguments
```c
typedef struct APP_LINK {
    UUID appUUID; // UUID of the document that is linked to.
    U32 link; // Link handle.
} APP_LINK, *P_APP_LINK;
```

Comments
The uuid of the document to link to is passed in. The document passes back a link handle, which is used by msgAppGetLink to retrieve the document. The document stores the uuid in its *appDocLinkFileName* file.

Descendants: You normally do not handle this message.
**msgAppDeleteLink**

Deletes the specified link handle.

Takes P_APP_LINK, returns STATUS.

```c
#define msgAppDeleteLink MakeMsg(clsApp, 53)
```

**Message**

typedef struct APP_LINK {
    UUID appUUID; // UUID of the document that is linked to.
    U32 link; // Link handle.
} APP_LINK, *P_APP_LINK;

**Comments**

Descendants: You normally do not handle this message.

**msgAppGetLink**

Passes back a document's UUID for the specified link handle.

Takes P_APP_LINK, returns STATUS.

```c
#define msgAppGetLink MakeMsg(clsApp, 54)
```

**Message**

typedef struct APP_LINK {
    UUID appUUID; // UUID of the document that is linked to.
    U32 link; // Link handle.
} APP_LINK, *P_APP_LINK;

**Comments**

Descendants: You normally do not handle this message.

### Standard Application Menu Messages

#### msgAppCreateMenuBar

Creates the standard application menu bar.

Takes P_OBJECT, returns STATUS.

```c
#define msgAppCreateMenuBar MakeMsg(clsApp, 55)
```

**Comments**

Descendants: You should handle this message by creating the document's menu bar. If pArgs is non-null when the ancestor is called, clsApp will pre-pend the Document, Edit, and Option menus to the provided menu bar. So you should call the ancestor after you make the menu bar. After the ancestor returns, you can fix up the Document and Edit menus to remove any buttons that you don't support or to add any new buttons.

See the earlier description "Enabling and Disabling SAMs" for more details.

#### msgAppCreateClientWin

Creates a document's client window.

Takes P_OBJECT, returns STATUS.

```c
#define msgAppCreateClientWin MakeMsg(clsApp, 56)
```

**Comments**

The document creates a default client window of class clsEmbeddedWin and passes back its uid.

The Application Framework does not send this message by default. Instead, you should self send it at the appropriate time (typically during msgAppInit, since the client window is usually stateful).

Descendants: You should handle this message by creating your application-specific client window. In such cases, you tend not to call your ancestor.
msgAppSend
Sends a document.
Takes OBJECT, returns STATUS.
`#define msgAppSend MakeMsg(clsApp, 57)`

Comments
When the user taps on a button in the Send menu, the SAMs send this message to the document, passing in the SendManager. The document then self sends msgAppInvokeManager, passing on the SendManager.

Descendants: You normally do not handle this message.

msgAppPrint
Prints a document.
Takes OBJECT, returns STATUS.
`#define msgAppPrint MakeMsg(clsApp, 58)`

Comments
When the user issues the Print command (either by tapping on the Print button in the SAMs or by drawing the print gesture on the document's title line), the Application Framework sends this message to the document, passing it the PrintManager. The document then self sends msgAppInvokeManager, passing on the PrintManager.

Descendants: You normally do not handle this message.

msgAppPrintSetup
Displays a document's print setup option sheet.
Takes nothing, returns STATUS.
`#define msgAppPrintSetup MakeMsg(clsApp, 59)`

Comments
When the user taps on Print Setup, the SAMs send this message to the document. The document self sends msgAppOptionShowOptionSheet, passing it tagAppPrintSetupOptSheet.

Descendants: You normally do not handle this message.

msgAppImport
Obsolete message. Not implemented.
Takes nothing, returns STATUS.
`#define msgAppImport MakeMsg(clsApp, 60)`

See Also
msgImport

msgAppExport
Prepares to export a document as a file.
Takes OBJECT, returns STATUS.
`#define msgAppExport MakeMsg(clsApp, 61)`

Comments
The document self sends msgAppInvokeManager, passing on pArgs.

Descendants: You normally do not handle this message.
msgAppAbout
Displays a document's "About" option sheet.
Takes nothing, returns STATUS.

#define msgAppAbout MakeMsg(clsApp, 62)

Comments
When the user taps on About, the SAMs send this message to the document. The document self sends msgAppOptionShowSheet, passing it tagAppAboutOptSheet.

Descendants: You normally do not handle this message. Instead, you should handle msgOptionAddCards by adding more cards to the About option sheet. Likewise, you should handle msgOptionProvideCard by modifying or adding specific controls to the standard About cards.

msgAppHelp
Shows help for the application. Not implemented - Reserved.
Takes nothing, returns STATUS.

#define msgAppHelp MakeMsg(clsApp, 63)

Comments
Descendants: You should not handle this message. Instead, you can provide help via resource files (see the Tic-Tac-Toe sample application for an example).

msgAppUndo
Undoes the previous operation on a document.
Takes nothing, returns STATUS.

#define msgAppUndo MakeMsg(clsApp, 64)

Comments
The document sends msgUndoCurrent to theUndoManager.

Descendants: You normally do not handle this message. Instead, see UNDO.H for information on how to undo your application's commands.

msgAppMoveSel
Prepares to move a document's selection.
Takes nothing, returns STATUS.

#define msgAppMoveSel MakeMsg(clsApp, 65)

Comments
When the user issues the Move command (either by tapping on Move in the SAMs or by press-holding on a selection in the document), the Application Framework sends this message to the document. The document finds its selected object (by sending msgSelOwner to theSelectionManager) and then sends it msgSelBeginMove.

Descendants: You normally do not handle this message.

msgAppCopySel
Prepares to copy the document's selection.
Takes nothing, returns STATUS.

#define msgAppCopySel MakeMsg(clsApp, 66)
When the user issues the Copy command (either by tapping on Copy in the SAMs or by tap-press-holding on a selection in the document), the Application Framework sends this message to the document. The document finds its selected object (by sending msgSelOwner to theSelectionManager) and then sends it msgSelBeginCopy.

Descendants: You normally do not handle this message.

**msgAppDeleteSel**
Deletes a document's selection.

Takes nothing, returns STATUS.

```c
#define msgAppDeleteSel MakeMsg(clsApp, 67)
```

Comments
When the user issues the Delete command (either by tapping on Delete in the SAMs or by drawing the delete gesture, the Application Framework sends this message to the document. The document gets its selected object (by sending msgSelOwner to theSelectionManager) and then sends it msgSelDelete.

Descendants: You normally do not handle this message.

**msgAppSelOptions**
Prepares to display the options for a document's selection. Obsolete.

Takes nothing, returns STATUS.

```c
#define msgAppSelOptions MakeMsg(clsApp, 68)
```

Comments
Descendants: You should not handle this message.

**msgAppSelectAll**
Selects all of the objects in a document.

Takes nothing, returns STATUS. Category: descendant responsibility.

```c
#define msgAppSelectAll MakeMsg(clsApp, 69)
```

Comments
When the user taps on Select All in the Standard Application Menu, the document self sends this message. clsApp does not do anything in its message handler for this message.

Descendants: You should handle this message and select everything in the document. You tend not to call the ancestor.

**msgAppSearch**
Searches a document for a string.

Takes OBJECT, returns STATUS.

```c
#define msgAppSearch MakeMsg(clsApp, 70)
```

Comments
When the user issues the Find command (either by tapping on Find in SAMs or by drawing the find gesture on the document's title line), the Application Framework sends this message to the document, passing it theSeachManager. In response, the document self sends msgAppInvokeManager, passing on theSearchManager.

Descendants: You normally do not handle this message.
**msgAppSpell**

Prepares to check a document's spelling.

Takes OBJECT, returns STATUS.

```c
#define msgAppSpell MakeMsg(clsApp, 71)
```

Comments

When the user issues the Spell command (either by tapping on Spell in SAMs or by drawing the spell gesture on the document's title line), the Application Framework sends this message to the document, passing it theSpellManager. In response, the document self sends msgAppInvokeManager, passing on theSpellManager.

Descendants: You normally do not handle this message.

---

**msgAppInvokeManager**

Routes a message to a manager.

Takes OBJECT, returns STATUS.

```c
#define msgAppInvokeManager MakeMsg(clsApp, 72)
```

Comments

To route a standard application menu message to the object that provides the behavior, the document self sends msgAppInvokeManager. The argument to the message is the well-known UID of the manager that performs the operation. When the document receives msgAppInvokeManager, it sends msgAppExecute to the manager object.

Descendants: You normally do not handle this message.

---

**msgAppExecute**

Sent to the manager to execute the manager's behavior on a document.

Takes P_APP_EXECUTE, returns STATUS.

```c
#define msgAppExecute MakeMsg(clsApp, 73)
```

**Arguments**

```c
typedef struct APP_EXECUTE {
    OBJECT app;   // Requesting document.
    OBJECT sel;   // Selected object.
    U32   count;  // Number of uuids.
    UUID  uuid[1]; // UUIDs of documents to operate on.
} APP_EXECUTE, *P_APP_EXECUTE;
```

Comments

The document sends msgAppExecute to a manager when it receives msgAppInvokeManager. The manager performs some operation on the document or documents specified in the pArgs, such as printing, searching, or spell checking.

Descendants: You normally do not handle this message.

---

**msgAppExecuteGesture**

Invokes the default gesture behavior for a document's title line.

Takes P_GWIN_GESTURE, returns STATUS.

```c
#define msgAppExecuteGesture MakeMsg(clsApp, 74)
```

Comments

Descendants: You normally do not handle this message. However, if you want to handle a title line gesture differently than the default, you should handle this message. You tend not to call the ancestor.
**Notification messages**

**msgAppSetSaveOnTerminate**
Tells a document to save itself before terminating.
Takes BOOLEAN, returns STATUS.

```
define msgAppSetSaveOnTerminate MakeMsg(clsApp, 75)
```

Comments
If `msgAppSetSaveOnTerminate` has been sent before `msgAppTerminate`, the document will be sent `msgAppSave` even if it refuses to terminate. Normally, if a document vetos `msgAppTerminate`, it is not sent `msgAppSave`.

Descendants: You normally do not handle this message.

**msgAppTerminateConditionChanged**
Try to terminate a document; sent when a terminate condition changed.
Takes nothing, returns STATUS.

```
define msgAppTerminateConditionChanged MakeMsg(clsApp, 76)
```

Comments
In response to this message, the document self sends `msgAppTerminate(true)`.
This message is self sent when a terminate condition has changed. For example, the document might have given up its selection and can now be terminated.
Descendants: You normally do not handle this message. Instead, see `msgAppTerminateOK`.

**msgAppSelChanged**
Sent to a document when something in it becomes selected or deselected.
Takes BOOLEAN, returns STATUS.

```
define msgAppSelChanged MakeMsg(clsApp, 77)
```

Comments
pArgs is true when the document (or one of its embedded documents) gains the selection. pArgs is false when the selection leaves the document.
The document self sends `msgAppTerminateConditionChanged` when it no longer has the selection.
Descendants: You normally do not handle this message.

**msgAppOpened**
Sent to observers of a document when the document is opened.
Takes APP_OPENED, returns STATUS. Category: observer notification.

```
define msgAppOpened MsgNoError(MakeMsg(clsApp, 78))
```

Comments
pArgs->child is the uid of the document that has been opened.

**msgAppClosed**
Sent to observers of a document when the document is closed.
Takes APP_CLOSED, returns STATUS. Category: observer notification.

```
define msgAppClosed MsgNoError(MakeMsg(clsApp, 79))
```

Comments
pArgs->child is the uid of the document that has been closed.
**msgAppChildChanged**

Sent to observers of a document when a child document is opened or closed.

Takes P_APP_CHILD_CHANGED, returns STATUS. Category: observer notification.

```c
#define msgAppChildChanged MsgNoError(MakeMsg(clsApp, 80))
```

**Arguments**

```c
typedef struct APP_CHILD_CHANGED {
    OBJECT parent; // Parent of doc that changed.
    OBJECT child; // Doc that changed.
    UUID uuid; // UUID of doc that changed.
    MESSAGE change; // msgAppOpened or msgAppClosed.
    U32 reserved[4]; // Reserved.
} APP_CHILD_CHANGED, *P_APP_CHILD_CHANGED,
APP_OPENED, *P_APP_OPENED,
APP_CLOSED, *P_APP_CLOSED;
```

**Comments**

This message is sent to observers of a document in response to msgAppOpened and msgAppClosed.

**msgAppFloated**

Sent to observers when a document is floated or un-floated.

Takes P_APP_FLOATED, returns STATUS. Category: observer notification.

```c
#define msgAppFloated MsgNoError(MakeMsg(clsApp, 81))
```

**Arguments**

```c
typedef struct APP_FLOATED {
    OBJECT app; // Document that is floated or un-floated.
    BOOLEAN floatUp; // true=document is floated.
} APP_FLOATED, *P_APP_FLOATED;
```

**msgAppCreated**

Sent to observers of clsApp when a document is created.

Takes P_APP_CREATED, returns STATUS. Category: observer notification.

```c
#define msgAppCreated MsgNoError(MakeMsg(clsApp, 82))
```

**Arguments**

```c
typedef struct APP_CREATED {
    OBJECT rootContainer; // Root container uid.
    UUID rootContainerUUID; // Root container uid.
    UUID uuid; // Created doc’s uid.
    U32 reserved[4]; // Reserved.
} APP_CREATED, *P_APP_CREATED;
```

**msgAppDeleted**

Sent to observers of clsApp when a document is deleted.

Takes P_APP_DELETED, returns STATUS. Category: observer notification.

```c
#define msgAppDeleted MsgNoError(MakeMsg(clsApp, 83))
```

**Arguments**

```c
typedef struct APP_DELETED {
    OBJECT rootContainer; // Root container uid.
    UUID rootContainerUUID; // Root container uid.
    OBJECT app; // Deleted document. objNull if inactive.
    UUID uuid; // Deleted document’s uid.
    U32 reserved[4]; // Reserved.
} APP_DELETED, *P_APP_DELETED;
```
msgAppMoved

Sent to observers of clsApp when a document is moved.

Takes P_APP_MOVED_COPIED, returns STATUS. Category: observer notification.

```c
#define msgAppMoved      MsgNoError(MakeMsg(clsApp, 84))
// Move/copy values for moveCopyInfo argument
#define appMovedCopiedInto 0 // doc moved/copied to this root container
#define appMovedCopiedOutOf 1 // doc moved/copied from this root container
#define appMovedCopiedWithin 2 // doc moved/copied within this root container
```

Arguments

typedef struct APP_MOVED_COPIED {
  OBJECT rootContainer; // Root container uid.
  UUID rootContainerUUID; // Root container uuid.
  OBJECT app; // Moved/copied doc. objNull if inactive.
  UUID uuid; // Moved/copied document's uuid.
  U32 moveCopyInfo; // Type of move/copy.
  U32 reserved[4]; // Reserved.
} APP_MOVED_COPIED, *P_APP_MOVED_COPIED;

Comments

When a document is moved, the Application Framework notifies the observers of clsApp that a
document has moved either a) within a root container, or b) out of one root container and into another.
(It may help you to remember that root containers are typically notebooks.)

To notify the observers, the Application Framework creates a list containing the document that is being
moved and each of its embedded documents. If the document is being moved within the root container,
then for each of the documents in the list, the Application Framework sends msgAppMoved to the
observers of clsApp, specifying appMovedCopiedWithin. If the document is being moved from one
container to another, the Application Framework sends msgAppMoved twice for each document, once
specifying appMovedCopiedOutOf and once specifying msgMovedCopiedInto.

See Also

msgAppChanged

msgAppCopied

Sent to observers of clsApp when a document is copied.

Takes P_APP_MOVED_COPIED, returns STATUS. Category: observer notification.

```c
#define msgAppCopied      MsgNoError(MakeMsg(clsApp, 85))
```

Arguments

typedef struct APP_MOVED_COPIED {
  OBJECT rootContainer; // Root container uid.
  UUID rootContainerUUID; // Root container uuid.
  OBJECT app; // Moved/copied doc. objNull if inactive.
  UUID uuid; // Moved/copied document's uuid.
  U32 moveCopyInfo; // Type of move/copy.
  U32 reserved[4]; // Reserved.
} APP_MOVED_COPIED, *P_APP_MOVED_COPIED;

Comments

When a document is copied, the Application Framework notifies the observers of clsApp that a
document has been copied either a) within a root container, or b) from one root container into another.
(It may help you to remember that root containers are typically notebooks.)

To notify the observers, the Application Framework creates a list containing the document that is being
copied and each of its embedded documents. If the document is being copied within the root container,
then for each of the documents in the list, the Application Framework sends msgAppCopied to the
observers of clsApp, specifying appMovedCopiedWithin. If the document is being copied from one
container to another, the Application Framework sends msgAppCopied twice for each document, once
specifying appMovedCopiedOutOf and once specifying msgMovedCopiedInto.

See Also

msgAppChanged
msgAppChanged
Sent to observers of clsApp when a document has changed.
Takes P_APP_CHANGED, returns STATUS. Category: observer notification.

#define msgAppChanged MsgNoError(MakeMsg(clsApp, 86))
// State of a doc's bookmark (which is interpreted in the NUI as a tab)
#define appBookmarkOn 1
#define appBookmarkOff 2

typedef struct APP_CHANGED {
    OBJECT rootContainer;     // In: Root container uid.
    UUID rootContainerUUID;   // In: Root container uuid.
    UUID uid;                 // In: The uuid of the changed document.
    OBJECT bookmark;          // In: true if doc's container (i.e.,
    U16    globalSequence : 1;  // notebook) needs to be renumbered.
    U16    name : 1;           // In: true if doc's name changed
    U16    bookmark : 2;       // In: new bookmark state, if changed
    U16    create : 1;         // In: true if doc is new
    U16    deleted : 1;        // In: true if doc was deleted
    U16    move : 1;           // In: true if doc was moved
    U16    copy : 1;           // In: true if doc was copied
    U16    reserved1 : 8;      // In: if doc was moved or copied, this
    U16    moveCopyInfo;       // is set to move/copy value described
                             // in msgAppMoved.
    U32    reserved2[4];
} APP_CHANGED, *P_APP_CHANGED;

This message is sent to observers of clsApp when a document has changed in some way (e.g.,
the document has moved, has a new name, has been created, and so on).
When a document is moved or copied, this message is sent to observers of clsApp. However, it is not
sent for all of the document's embedded documents (thereby making it different from msgAppMoved
and msgAppCopied).

See Also
msgAppMoved

msgAppInstalled
Sent to observers of clsApp when an application is installed.
Takes CLASS, returns STATUS. Category: observer notification.

#define msgAppInstalled MsgNoError(MakeMsg(clsApp, 87))

Comments
pArgs is the class of the application just installed.

msgAppDeInstalled
Sent to observers of clsApp when an application is deinstalled.
Takes CLASS, returns STATUS. Category: observer notification.

#define msgAppDeInstalled MsgNoError(MakeMsg(clsApp, 88))

Comments
pArgs is the class of the application just deinstalled.
Public Functions

AppMain
Creates a document instance and starts dispatching messages to it.
Returns nothing.

Function Prototype: STATUS EXPORTED AppMain (void);
Comments: All developers should call AppMain from their main routine whenever processCount is greater than 0.

AppMonitorMain
Creates an app monitor instance and handles installing the application.
Returns nothing.

Function Prototype: STATUS EXPORTED AppMonitorMain (OBJECT, OBJECT);
Comments: All developers should call AppMonitorMain from their main routine when processCount is equal to 0. You specify the well-known uid of your application class and the well-known uid of your app monitor class. If you do not have an app monitor class, simply specify objNull for the second parameter.
APPDIR.H

This file contains the API definition for clsAppDir.

clsAppDir inherits from clsDirHandle.

Provides management for document directories.

"AppDir" stands for Application Directory Handle.

Introduction

Application directory nodes represent documents in the document hierarchy. Application directories are where documents store their resource files and any other files they use. Attributes on application directories specify useful information about each document.

clsAppDir is used to manage the various file system attributes associated with a document in PenPoint. It includes definitions of these attributes and messages to manage them. clsAppDir also provides support for enumerating embedded documents via the filesystem. This is similar to the file system’s FSReadDir facilities, but clsAppDir filters out all files and directories that are not documents.

A document can find its application directory by self sending msgAppGetMetrics. The application directory’s uid will be passed back in the dir field of the APP_METRICS structure. See app.h for more information.

Application directories are created automatically for documents during AppInit time by the Application Framework. Application classes generally should never create or destroy application directories themselves.

#ifndef APPDIR_INCLUDED
#define APPDIR_INCLUDED
#endif

Common #defines and typedefs

typedef OBJECT APP_DIR, *P_APP_DIR;

File System Attributes

These attributes are stamped on every document directory.

#define appAttrClass FSMakeFix32Attr(clsAppDir, 1)
#define appAttrSequence FSMakeFix32Attr(clsAppDir, 4)
#define appAttrNumChildren FSMakeFix32Attr(clsAppDir, 3)
#define appAttrFlags FSMakeFix64Attr(clsAppDir, 6)
#define appAttrBookmark FSMakeStrAttr(clsAppDir, 9)
#define appAttrAuthor FSMakeStrAttr(clsAppDir, 10)
#define appAttrComments FSMakeStrAttr(clsAppDir, 11)
#define appAttrClassName FSMakeStrAttr(clsAppDir, 12)
#define appAttrGlobalSequence FSMakeFix32Attr(clsAppDir, 4)
Application Directory Flags

This structure defines the application directory flags. They are stamped on a document directory with appAttrFlags. This structure is used in the flags field of APP_DIR_ATTRS.

typedef struct APP_DIR_FLAGS {
    U16 application : 1; // true = this is an application.
    U16 newlnstance : 1; // true = new app instance.
    U16 disabled : 1; // true = app is disabled, don’t activate.
    U16 bookmark : 1; // true = app has a tab
    U16 readOnly : 1; // true = app is read only.
    U16 deletable : 1; // true = app can be deleted.
    U16 movable : 1; // true = app can be moved.
    U16 copyable : 1; // true = app can be copied.
    U16 reserved1 : 8; // Reserved.
    U16 reserved2 : 16; // Reserved.
    U16 reserved3 : 16; // Reserved.
    U16 reserved4 : 16; // Reserved.
} APP_DIR_FLAGS, *P_APP_DIR_FLAGS;

Application Directory Attributes Structure

This structure is used to specify and pass back the directory attributes in one chunk.

- appClass The document’s application class (sub-class of clsApp).
- uuid The document’s uuid. Can be used in msgNew to clsDirHandle or clsAppDir to open a handle on a document directory.
- sequence The 1-based position of a document within its embeddor. If the document is in a notebook, this is the document’s position within its section.
- numChildren The total number of embedded children.

typedef struct APP_DIR_ATTRS {
    CLASS appClass; // Application class.
    UUID uuid; // Application uuid.
    U32 sequence; // Local sequence number.
    U32 numChildren; // Number of child apps (recursive).
    APP_DIR_FLAGS flags; // Flags.
} APP_DIR_ATTRS, *P_APP_DIR_ATTRS;

Messages

msgNew

Creates a new AppDir.

Takes P_FS_NEW, returns STATUS. Category: class message.

Comments

See fs.h for the FS_NEW structure definition.

clsAppDir has no method for msgNewDefaults. See fs.h for a description of clsDirHandle’s handler for msgNewDefaults.
msgAppDirGetAttrs
Passes back a document’s application directory attributes.
Takes P_APP_DIR_GET_SET_ATTRS, returns STATUS.

#define msgAppDirGetAttrs MakeMsg(clsAppDir, 1)

Arguments
typedef struct APP_DIR_GET_SET_ATTRS {
    P_STRING pPath; // in: Path relative to target directory.
    APP_DIR_ATTRS attrs; // in/out: Application directory attributes.
} APP_DIR_GET_SET_ATTRS, *P_APP_DIR_GET_SET_ATTRS;

Comments
If you are interested in only one of the attributes, use the individual msgAppDirGet... messages described below. They’re generally faster.

msgAppDirSetAttrs
Specifies a document’s application directory attributes.
Takes P_APP_DIR_GET_SET_ATTRS, returns STATUS.

#define msgAppDirSetAttrs MakeMsg(clsAppDir, 2)

Arguments
typedef struct APP_DIR_GET_SET_ATTRS {
    P_STRING pPath; // in: Path relative to target directory.
    APP_DIR_ATTRS attrs; // in/out: Application directory attributes.
} APP_DIR_GET_SET_ATTRS, *P_APP_DIR_GET_SET_ATTRS;

Comments
If you are interested in only one of the attributes, use the individual msgAppDirSet... messages described below. They’re generally faster.

msgAppDirGetFlags
Passes back a document’s application directory flags.
Takes P_APP_DIR_GET_SET_FLAGS, returns STATUS.

#define msgAppDirGetFlags MakeMsg(clsAppDir, 3)

Arguments
typedef struct APP_DIR_GET_SET_FLAGS {
    P_STRING pPath; // in: Path relative to target directory.
    APP_DIR_FLAGS flags; // in/out: Application directory control flags.
} APP_DIR_GET_SET_FLAGS, *P_APP_DIR_GET_SET_FLAGS;

msgAppDirSetFlags
Specifies a document’s application directory flags.
Takes P_APP_DIR_GET_SET_FLAGS, returns STATUS.

#define msgAppDirSetFlags MakeMsg(clsAppDir, 4)

Arguments
typedef struct APP_DIR_GET_SET_FLAGS {
    P_STRING pPath; // in: Path relative to target directory.
    APP_DIR_FLAGS flags; // in/out: Application directory control flags.
} APP_DIR_GET_SET_FLAGS, *P_APP_DIR_GET_SET_FLAGS;
**msgAppDirGetClass**

Passes back a document's application class.

Takes `P_APP_DIR_UPDATE_CLASS`, returns `STATUS`.

```c
#define msgAppDirGetClass MakeMsg(clsAppDir, 5)
```

**Message Arguments**

```c
typedef struct APP_DIR_UPDATE_CLASS {
  P_STRING pPath; // in: Path relative to target directory.
  CLASS appClass; // in/out: Application directory class.
} APP_DIR_UPDATE_CLASS, *P_APP_DIR_UPDATE_CLASS;
```

**msgAppDirSetClass**

Specifies a document's application class.

Takes `P_APP_DIR_UPDATE_CLASS`, returns `STATUS`.

```c
#define msgAppDirSetClass MakeMsg(clsAppDir, 6)
```

**Message Arguments**

```c
typedef struct APP_DIR_UPDATE_CLASS {
  P_STRING pPath; // in: Path relative to target directory.
  CLASS appClass; // in/out: Application directory class.
} APP_DIR_UPDATE_CLASS, *P_APP_DIR_UPDATE_CLASS;
```

**msgAppDirGetUUID**

Passes back an application directory's uuid.

Takes `P_APP_DIR_UPDATE_UUID`, returns `STATUS`.

```c
#define msgAppDirGetUUID MakeMsg(clsAppDir, 7)
```

**Message Arguments**

```c
typedef struct APP_DIR_UPDATE_UUID {
  P_STRING pPath; // in: Path relative to target directory.
  UUID uuid; // in/out: Application directory uuid.
} APP_DIR_UPDATE_UUID, *P_APP_DIR_UPDATE_UUID;
```

**msgAppDirSetUUID**

Specifies an application directory's uuid.

Takes `P_APP_DIR_UPDATE_UUID`, returns `STATUS`.

```c
#define msgAppDirSetUUID MakeMsg(clsAppDir, 8)
```

**Message Arguments**

```c
typedef struct APP_DIR_UPDATE_UUID {
  P_STRING pPath; // in: Path relative to target directory.
  UUID uuid; // in/out: Application directory uuid.
} APP_DIR_UPDATE_UUID, *P_APP_DIR_UPDATE_UUID;
```

**msgAppDirGetUID**

Passes back an application directory's uid.

Takes `P_APP_DIR_UPDATE_UlD`, returns `STATUS`.

```c
#define msgAppDirGetUID MakeMsg(clsAppDir, 9)
```

**Message Arguments**

```c
typedef struct APP_DIR_UPDATE_UID {
  P_STRING pPath; // in: Path relative to target directory.
  UID uid; // in/out: Application directory uid.
} APP_DIR_UPDATE_UID, *P_APP_DIR_UPDATE_UID;
```
**msgAppDirSetUID**

Specifies an application directory's uid.

Takes P_APP_DIR_UPDATE_UID, returns STATUS.

```c
#define msgAppDirSetUID MakeMsg(clsAppDir, 10)
```

**Message**

typedef struct APP_DIR_UPDATE_UID {
P_STRING pPath;    // in: Path relative to target directory.
UID uid;          // in/out: App directory uid.
} APP_DIR_UPDATE_UID, *P_APP_DIR_UPDATE_UID;

**msgAppDirGetSequence**

Passes back an application directory's sequence number.

Takes P_APP_DIR_UPDATE_SEQ, returns STATUS.

```c
#define msgAppDirGetSequence MakeMsg(clsAppDir, 11)
```

**Message**

typedef struct APP_DIR_UPDATE_SEQUENCE {
P_STRING pPath;    // in: Path relative to target directory.
U32 sequence;    // in/out: Application directory sequence.
} APP_DIR_UPDATE_SEQUENCE, *P_APP_DIR_UPDATE_SEQUENCE;

**Comments**

If the document is in a notebook, the sequence number is a 1-based position within the section.

**msgAppDirSetSequence**

Specifies an application directory's sequence number.

Takes P_APP_DIR_UPDATE_SEQUENCE, returns STATUS.

```c
#define msgAppDirSetSequence MakeMsg(clsAppDir, 12)
```

**Message**

typedef struct APP_DIR_UPDATE_SEQUENCE {
P_STRING pPath;    // in: Path relative to target directory.
U32 sequence;    // in/out: Application directory sequence.
} APP_DIR_UPDATE_SEQUENCE, *P_APP_DIR_UPDATE_SEQUENCE;

**Comments**

If the document is in a notebook, the sequence number is a 1-based position within the section.

**msgAppDirGetNumChildren**

Passes back the total number of embedded children of a document.

Takes P_APP_DIR_UPDATE_NUM_CHILDREN, returns STATUS.

```c
#define msgAppDirGetNumChildren MakeMsg(clsAppDir, 22)
```

**Arguments**

typedef struct APP_DIR_UPDATE_NUM_CHILDREN {
P_STRING pPath;    // in: Path relative to target directory.
U32 numChildren;    // in/out: App directory attr numchildren.
} APP_DIR_UPDATE_NUM_CHILDREN, *P_APP_DIR_UPDATE_NUM_CHILDREN;

**msgAppDirSetNumChildren**

Specifies the total number of embedded children of a document.

Takes P_APP_DIR_UPDATE_NUM_CHILDREN, returns STATUS.

```c
#define msgAppDirSetNumChildren MakeMsg(clsAppDir, 23)
```

**Arguments**

typedef struct APP_DIR_UPDATE_NUM_CHILDREN {
P_STRING pPath;    // in: Path relative to target directory.
U32 numChildren;    // in/out: App directory attr numchildren.
} APP_DIR_UPDATE_NUM_CHILDREN, *P_APP_DIR_UPDATE_NUM_CHILDREN;
msgAppDirGetGlobalSequence

Passes back an application directory's global sequence number.

Takes P_APP_DIR_GET_GLOBAL_SEQUENCE, returns STATUS.

#define msgAppDirGetGlobalSequence MakeMsg(clsAppDir, 21)

Arguments
typedef struct APP_DIR_GET_GLOBAL_SEQUENCE {
   P_STRING pPath;  // in: Path relative to target directory.
   U32  globalSequence;  // in/out: App directory global sequence.
} APP_DIR_GET_GLOBAL_SEQUENCE, *P_APP_DIR_GET_GLOBAL_SEQUENCE;

Comments
The global sequence number is the 1-based position of a document within its clsRootContainerApp embedder (i.e., the document's page number in the notebook).

msgAppDirGetBookmark

Passes back an document's application tab.

Takes P_APP_DIR_GET_BOOKMARK, returns STATUS.

#define msgAppDirGetBookmark MakeMsg(clsAppDir, 13)

typedef struct APP_DIR_GET_BOOKMARK {
   P_STRING pPath;  // in: Path relative to target directory.
   char label [nameBufLength];  // out: tab label.
} APP_DIR_GET_BOOKMARK, *P_APP_DIR_GET_BOOKMARK;

Comments
If the application directory has no tab (appDirFlags.bookmark==false), msgAppDirGetBookmark will return stsOK and pArgs->label will be unchanged. For this reason it is recommended that you drop a null byte into pArgs->label[0] before calling msgAppDirGetBookmark. Then, if the application directory has no tab, you will get back a null string.

msgAppDirSetBookmark

Specifies a document's application tab.

Takes P_APP_DIR_SET_BOOKMARK, returns STATUS.

#define msgAppDirSetBookmark MakeMsg(clsAppDir, 14)

Arguments
typedef struct APP_DIR_SET_BOOKMARK {
   BOOLEAN on;  // in: Turn bookmark on or off.
   P_STRING pPath;  // in: Path relative to target directory.
   char label [nameBufLength];  // in/out: tab label.
} APP_DIR_SET_BOOKMARK, *P_APP_DIR_SET_BOOKMARK;

Comments
clsAppDir sends msgAppChanged to observers of clsApp as a result of this message. See app.h for a description of msgAppChanged.

If label[0] is NULL, clsAppDir uses the default label, which is the name of the document.

msgAppDirGetNextInit

Initializes an APP_DIR_NEXT structure.

Takes P_APP_DIR_NEXT, returns STATUS.

#define msgAppDirGetNextInit MakeMsg(clsAppDir, 15)

Comments
Send this message to an application directory to prepare it for an ensuing msgAppDirGetNext loop.
**msgAppDirGetNext**

Passes back the attributes of the next application directory.

Takes P_APP_DIR_NEXT, returns STATUS.

```c
#define msgAppDirGetNext
typedef struct APP_DIR_NEXT {
    APP_DIR_ATTRS attrs; // out: attrs for next child.
    P_STRING pName; // out: name of next child.
    U32 fsFlags; // out: fs flags for next child (see fs.h)
    P UNKNOWN pFirst; // out: first app dir to examine
    P UNKNOWN pNext; // out: next app dir to examine
    P UNKNOWN handle; // out: current app dir
} APP_DIR_NEXT, *P_APP_DIR_NEXT;
```

**Arguments**

- `APP_DIR_ATTRS`: out - attrs for next child.
- `P_STRING`: out - name of next child.
- `U32`: out - fs flags for next child (see fs.h)
- `P UNKNOWN`: out - first app dir to examine
- `P UNKNOWN`: out - next app dir to examine
- `P UNKNOWN`: out - current app dir

**Comments**

Send this message to an application directory in a loop to get the appDirAttrs for each embedded document (not recursive), ordered by sequence number.

You generally do not change the values in the APP_DIR_NEXT structure between calls to `msgAppDirGetNext`. Doing so jeopardizes the traversal of the embedded documents.

---

**msgAppDirReset**

Frees resources after a series of `msgAppDirGetNext` messages.

Takes P_APP_DIR_NEXT, returns STATUS.

```c
#define msgAppDirReset
typedef struct APP_DIR_NEXT {
    APP_DIR_ATTRS attrs; // out: attrs for next child.
    P_STRING pName; // out: name of next child.
    U32 fsFlags; // out: fs flags for next child (see fs.h)
    P UNKNOWN pFirst; // out: first app dir to examine
    P UNKNOWN pNext; // out: next app dir to examine
    P UNKNOWN handle; // out: current app dir
} APP_DIR_NEXT, *P_APP_DIR_NEXT;
```

**Arguments**

- `APP_DIR_ATTRS`: out - attrs for next child.
- `P_STRING`: out - name of next child.
- `U32`: out - fs flags for next child (see fs.h)
- `P UNKNOWN`: out - first app dir to examine
- `P UNKNOWN`: out - next app dir to examine
- `P UNKNOWN`: out - current app dir

**Comments**

You must send this message to the application directory after the `msgAppDirGetNext` loop has completed. Failing to do so can cause internally allocated memory not to be deallocated.

---

**msgAppDirSeqToName**

Passes back the name of the embedded document with a specified sequence number.

Takes P_APP_DIR_SEQ_TO_NAME, returns STATUS.

```c
#define msgAppDirSeqToName
typedef struct APP_DIR_SEQ_TO_NAME {
    U32 sequence; // in: Sequence number.
    P_STRING pName; // out: Buffer for name.
} APP_DIR_SEQ_TO_NAME, *P_APP_DIR_SEQ_TO_NAME;
```

**Arguments**

- `U32`: in - Sequence number.
- `P_STRING`: out - Buffer for name.

**Message**

You must send this message to the application directory after the `msgAppDirGetNext` loop has completed. Failing to do so can cause internally allocated memory not to be deallocated.

---

**msgAppDirGetDirectNumChildren**

Passes back the number of directly embedded documents (not recursive).

Takes P_U32, returns STATUS.

```c
#define msgAppDirGetDirectNumChildren
```

**Arguments**

- `U32`: out - Number of directly embedded documents (not recursive).
msgAppDirGetTotalNumChildren

Passes back the total number of embedded documents (recursive).

Takes P_U32, returns STATUS.

#define msgAppDirGetTotalNumChildren MakeMsg(clsAppDir, 20)
This file contains the API definition for clsAppMgr.

clsAppMgr inherits from clsClass.

Provides support for application classes and document management.

"AppMgr" stands for Application Manager.

Introduction

When you create a new application class (i.e., install an application), rather than sending msgNew to clsClass you send msgNew to clsAppMgr. This allows you to specify properties of the application class, and also to specify in advance some default properties of the documents (i.e., instances) of the application class.

There is one instance of clsAppMgr for each installed application class. This object is given the well-known uid of the application class. The application manager class implements document management messages and stores information about the installed application class in its instance data.

```
#ifndef APPMGR_INCLUDED
#define APPMGR_INCLUDED

#include <fs.h>
#include <geo.h>
```

Common #defines and typedefs

```c
typedef OBJECT APPMGR, *P_APPMGR;
```

AppMgr Flags

Various settings for the installed application class.

stationery: If true, an instance of the application will be placed in the Stationery Notebook when the application is installed. The instance will have default parameters. You can also create customized stationery instances using the STATNRY subdirectory. See appmon.h for more details.

accessory: If true, an instance of the application will be placed in the Accessories Palette. The instance will have default parameters. You can also create customized accessories instances using the ACESSRY subdirectory. See appmon.h for more details.

hotMode: If true, instances of the application are created in hot mode by default. Note that you can change a document's hot mode flag at msgInit time (or at any other time) using msgAppSetHotMode. See app.h for more details.

allowEmbedding: If true, instances of the application allow child applications to be embedded within them. This parameter cannot be modified on a per-document basis.

confirmDelete: If true, PenPoint will ask for user confirmation before deleting any instance of the application. This parameter cannot be modified on a per-document basis.

deinstallable: If false, users will be prevented from deinstalling the application class.
systemApp: If true, users will not see the application on the list of choices for importing documents.

lowMemoryApp: If false, users will be prevented from activating instances of the application when the system is low on memory.

fullEnvironment: If true, instance 0 of the application will have a full environment, including a resource list and floating window list. If false, these two items are destroyed, saving memory. In general, if your application does no processing in instance 0 (i.e., it simply calls AppMonitorMainO), you should set fullEnvironment to false to save unneeded memory.

typedef struct APP_MGR_FLAGS {
    U16 stationery  : 1; // Put in stationery notebook.
    U16 accessory   : 1; // Put in accessory palette.
    U16 hotMode    : 1; // Create docs in hot mode.
    U16 allowEmbedding : 1; // Allow child embedded apps.
    U16 confirmDelete : 1; // Confirm document deletes.
    U16 deinstallable : 1; // App class deinstallable.
    U16 systemApp   : 1; // Disable imports into this app.
    U16 lowMemoryApp: 1; // Allow activation under low memory.
    U16 fullEnvironment : 1; // Initialize instance 0 environment.
    U16 reserved1  : 7; // Reserved.
    U16 reserved2  : 16; // Reserved.
} APP_MGR_FLAGS, *P_APP_MGR_FLAGS;

AppMgr Metrics and NEW Structure

Public instance data for an installed application class. Also the new structure for creating a new installed application class.

typedef struct APP_MGR_METRICS {
    // All fields are passed back from msgAppMgrGetMetrics.
    // For msgNew: in=specified, out=passed back, na=not applicable (don't care).
    OBJECT dir; // na: App monitor dir.
    OBJECT appMonitor; // na: App monitor object.
    OBJECT iconBitmap; // na: Icon bitmap.
    OBJECT smallIconBitmap; // na: Small icon bitmap.
    OBJECT appWinClass; // in: always clsAppWin.
    RECT32 defaultRect; // in: Default rectangle (in points).
    char name[bufLength]; // na: Application name.
    char version[bufLength]; // na: Version.
    char company[bufLength]; // in: Company name.
    char defaultDocName[bufLength]; // in/out: Default document name.
    char copyright; // in: Copyright notice.
    OS_PROG_HANDLE programHandle; // out: Program handle.
    U32 reserved[4]; // na: Reserved.
    APP_MGR_FLAGS flags; // in: Described above.
} APP_MGR_METRICS, *P_APP_MGR_METRICS,
APP_MGR_NEW_ONLY, *P_APP_MGR_NEW_ONLY;

msgNew

Install a new application class.

Takes P_APP_MGR_NEW, returns STATUS. Category: class message.

#define appMgrNewFields
    classNewFields
    APP_MGR_NEW_ONLY
    appMgr;
typedef struct APP_MGR_NEW {
    appMgrNewFields
} APP_MGR_NEW, *P_APP_MGR_NEW;

The fields you commonly set are:

pArgs->object.uid  your application class's uid
pArgs->cls.pMsg  your application class's method table
pArgs->cls.ancestor your application class's ancestor (usually clsApp)
pArgs->cls.size  size of a document's instance data
pArgs->cls.newArgsSize  size of the _NEW struct for the app class
pArgs->appMgr.defaultRect  rectangle to open doc to when floating
pArgs->appMgr.company  your company's name
pArgs->appMgr.defaultDocName  name of new documents of this application
pArgs->appMgr.copyright  copyright notice
pArgs->appMgr.flags (see description of flags above)

clsAppMgr objects cannot be locked, so clsAppMgr forces pArgs->object.key to 0.

msgNewDefaults

Initializes APP_MGR_NEW structure to default values.

Takes P_APP_MGR_NEW, returns STATUS. Category: class message.

typedef struct APP_MGR_NEW {
    appMgrNewFields
} APP_MGR_NEW, *P_APP_MGR_NEW;

Zeroes out pArgs->appMgr and sets

pArgs->object.cap = objCapCall | objCapSend | objCapScavenge;

pArgs->appMgr.flags.stationery = true;
pArgs->appMgr.flags.accessory = false;
pArgs->appMgr.flags.allowEmbedding = true;
pArgs->appMgr.flags.confirmDelete = true;
pArgs->appMgr.flags.deinstallable = true;
pArgs->appMgr.flags.systemApp = false;
pArgs->appMgr.flags.hotMode = false;
pArgs->appMgr.appWinClass = clsAppWin;

// Default rect: 300 x 300 points, centered in theRootWindow
WIN_METRICS wm;
ObjCallRet(msgWinGetMetrics, theRootWindow, &wm, s);
pArgs->appMgr.defaultRect.size.w = 300;
pArgs->appMgr.defaultRect.size.h = 300;
pArgs->appMgr.defaultRect.origin.x = (wm.bounds.size.w/2) -
                                   (pArgs->appMgr.defaultRect.size.w/2);
pArgs->appMgr.defaultRect.origin.y = (wm.bounds.size.h/2) -
                                   (pArgs->appMgr.defaultRect.size.h/2);
**msgAppMgrGetMetrics**

Passes back the AppMgr metrics.

Takes P_APP_MGR_METRICS, returns STATUS. Category: class message.

```c
#define msgAppMgrGetMetrics MakeMsg(clsAppMgr, 1)
```

```c
typedef struct APP_MGR_METRICS {
    // All fields are passed back from msgAppMgrGetMetrics.
    // For msgNew: in=specified, out=passed back, na=not applicable (don't care).
    OBJECT dir; // na: App monitor dir.
    OBJECT appMonitor; // na: App monitor object.
    OBJECT iconBitmap; // na: Icon bitmap.
    OBJECT smallIconBitmap; // na: Small icon bitmap.
    OBJECT appWinClass; // in: always clsAppWin.
    RECT32 defaultRect; // in: Default rectangle
    // (in points).
    char name[nameBufLength]; // na: Application name.
    char version[nameBufLength]; // na: Version.
    char company[nameBufLength]; // in: Company name.
    char defaultDocName[nameBufLength]; // in/out: Default
    // document name.
    P_STRING copyright; // in: Copyright notice.
    OS_PROG_HANDLE programHandle; // out: Program handle.
    U32 reserved[4]; // na: Reserved.
    APP_MGR_FLAGS flags; // in: Described above.
} APP_MGR_METRICS, *P_APP_MGR_METRICS,
```

**msgAppMgrCreate**

Creates a directory entry for a new document.

Takes P_APP_MGR_CREATE, returns STATUS.

```c
#define msgAppMgrCreateMakeMsg(clsAppMgr, 2)
```

```c
typedef struct APP_MGR_CREATE {
    // Parent doc. uid must be of clsAppDir.
    FS_LOCATOR locator; // in: Name of new app.
    P_STRING pName; // in/out: Name of new app.
    U32 sequence; // Sequence of new app in parent app.
    BOOLEAN renumber; // true=update global sequence #s.
    U32 reserved[2]; // reserved.
} APP_MGR_CREATE, *P_APP_MGR_CREATE;
```

This message transitions a document from the Non-Existent state to the Created state.

clsAppMgr creates a new file system directory entry for the new document, using the name im pName.
clsAppMgr also stamps the new directory with the application's class.

If pName is pNull, clsAppMgr creates a unique name, based on the application name. If pName is not
pNull, it points to a client-allocated buffer that must be nameBufLength bytes long.

After msgAppMgrCreate, the document will appear in the appropriate table of contents or icon
window. But the application instance itself will not be created until msgAppMgrActivate, which
transitions the document from the Created state to the Activated state.

**Return Value**

stsFSNodeNotFound Invalid pArgs->locator
**msgAppMgrActivate**

Activates a document.

Takes P_APP_MGR_ACTIVATE, returns STATUS.

```c
#define msgAppMgrActivate MakeMsg(clsAppMgr, 3)
```

**Arguments**

```c
typedef struct APP_MGR_ACTIVATE {
    OBJECT winDev;  // Window device to activate app on.
    FS LOCATOR locator; // Location of doc to activate.
    OBJECT parent; // Parent doc uid.
    OBJECT uid; // out: activated doc uid.
} APP_MGR_ACTIVATE, *P_APP_MGR_ACTIVATE;
```

**Comments**

This message transitions a document from the Created or Dormant state to the Activated state.

clsAppMgr creates a new process for the document, and a new instance of the application class in the new process. The Application Framework will then send the new application instance msgAppInit if the document was in the Created state, or msgAppRestore if the document was in the Dormant state.

**Return Value**

- **stsAppMgrLowMemNoActivate** Document could not be activated due to low memory conditions.
- **stsFSNodeNotFound** Invalid pArgs->locator.

---

**msgAppMgrMove**

Moves a document to a new location.

Takes P_APP_MGR_MOVE_COPY, returns STATUS.

```c
#define msgAppMgrMove MakeMsg(clsAppMgr, 4)
```

**Arguments**

```c
typedef struct APP_MGR_MOVE_COPY_STYLE {
    U16 showConfirm : 1;  // show confirmation UI
    U16 showProgress : 1;  // show progress UI
    U16 reserved : 14;    // reserved.
    U16 reserved2 : 16;   // reserved.
} APP_MGR_MOVE_COPY_STYLE, *P_APP_MGR_MOVE_COPY_STYLE;

typedef struct APP_MGR_MOVE_COPY {
    FS LOCATOR locator; // Source document location.
    OBJECT source;     // Source object.
    OBJECT dest;       // Destination object.
    XY32 xy;           // x,y location in dest object.
    CHAR name[nameBufLength]; // in/out New doc name;
    BOOLEAN renumber; // true=update global sequence #s.
    APP_MGR_MOVE_COPY_STYLE style; // Move/copy style.
    OBJECT appWin;     // out: move/copied appwin.
} APP_MGR_MOVE_COPY, *P_APP_MGR_MOVE_COPY;
```

**Comments**

clsAppMgr will display the appropriate UI to show the progress of any time-consuming moves.

If the move fails due to low memory, user cancellation, etc., msgAppMgrMove will nevertheless return a value >= stsOK. The user will have been notified of the condition via standard error messaging facilities.

---

**msgAppMgrCopy**

Copies a document to a new location.

Takes P_APP_MGR_MOVE_COPY, returns STATUS.

```c
#define msgAppMgrCopy MakeMsg(clsAppMgr, 5)
```
typedef struct APP_MGR_MOVE_COPY {
    FS_LOCATOR locator;       // Source document location.
    OBJECT source;            // Source object.
    OBJECT dest;              // Destination object.
    XY32 xy;                  // x,y location in dest object.
    CHAR name[nameBufLength]; // in/out: New doc name;
    BOOLEAN renumber;         // true=update global sequence $s.
    APP_MGR_MOVE_COPY_STYLE style; // Move/copy style.
    OBJECT appWin;            // out: move/copied appwin.
} APP_MGR_MOVE_COPY, *P_APP_MGR_MOVE_COPY;

clsAppMgr will display the appropriate UI to show the progress of any time-consuming copies.

If the copy fails due to low memory, user cancellation, etc., msgAppMgrCopy will nevertheless return a
value >= stsOK. The user will have been notified of the condition via standard error messaging facilities.

msgAppMgrFSMove
Low-level move message used internally by msgAppMgrMove.
Takes P_APP_MGR_FS_MOVE_COPY, returns STATUS. Category: internal use only.
#define msgAppMgrFSMove MakeMsg(clsAppMgr, 17)

typedef struct APP_MGR_FS_MOVE_COPY {
    FS_LOCATOR source;       // Source doc location.
    FS_LOCATOR dest;         // Location of new parent doc.
    U32 sequence;            // Sequence of new doc in parent doc.
    CHAR name[nameBufLength]; // in/out: Name of new doc.
    U32 reserved[2];         // reserved.
} APP_MGR_FS_MOVE_COPY, *P_APP_MGR_FS_MOVE_COPY;

msgAppMgrFSCopy
Low-level copy message used internally by msgAppMgrCopy.
Takes P_APP_MGR_FS_MOVE_COPY, returns STATUS. Category: internal use only.
#define msgAppMgrFSCopy MakeMsg(clsAppMgr, 18)

typedef struct APP_MGR_FS_MOVE_COPY {
    FS_LOCATOR source;       // Source doc location.
    FS_LOCATOR dest;         // Location of new parent doc.
    U32 sequence;            // Sequence of new doc in parent doc.
    CHAR name[nameBufLength]; // in/out: Name of new doc.
    U32 reserved[2];         // reserved.
} APP_MGR_FS_MOVE_COPY, *P_APP_MGR_FS_MOVE_COPY;

msgAppMgrDelete
Deletes a document.
Takes P_APP_MGR_DELETE, returns STATUS.
#define msgAppMgrDelete MakeMsg(clsAppMgr, 6)

typedef struct APP_MGR_DELETE {
    FS_LOCATOR locator;     // Document to delete.
    BOOLEAN renumber;       // true=update global sequence $s.
    U32 reserved[2];        // reserved.
} APP_MGR_DELETE, *P_APP_MGR_DELETE;

This message transitions a document from the Created or Dormant state to the Non-Existent state. The
document is deleted along with all of its directory nodes, embedded documents, document processes,
and so on.
msgAppMgrRename

Renames a document.
Takes P_APP_MGR_RENAME, returns STATUS.

```c
#define msgAppMgrRename MakeMsg(clsAppMgr, 7)
```

Arguments

typedef struct APP_MGR_RENAME {
  FS_LOCATOR locator; // Location of app to rename.
  P_STRING pName; // in/out: New app name.
  U32 reserved[2]; // reserved.
} APP_MGR_RENAME, *P_APP_MGR_RENAME;

Comments

pName must point to a buffer nameBufLength long.

Return Value

stsAppBadName Invalid new name.
stsAppDuplicateName Name already in use.

msgAppMgrShutdown

Unconditionally shuts down an application instance and all children.
Takes P_FS_LOCATOR, returns STATUS.

```c
#define msgAppMgrShutdown MakeMsg(clsAppMgr, 8)
```

Comments

This message transitions a document from the Activated or Opened state to the Dormant state. The document is not given the opportunity to veto the shutdown. The document is sent msgAppSave before the shutdown, so it can file its data.

msgAppMgrGetRoot

Passes back the root application (clsRootContainerApp) of a tree of applications.
Takes P_APP_MGR_GET_ROOT, returns STATUS.

```c
#define msgAppMgrGetRoot MakeMsg(clsAppMgr, 9)
```

Arguments

typedef struct APP_MGR_GET_ROOT {
  FS_LOCATOR locator; // Location of app.
  char path[fsPathBufLength]; // out: Path to root.
  UUID uuid; // out: Root uuid;
  OBJECT app; // out: Root app. objNull if inactive.
  U32 reserved[2]; // reserved.
} APP_MGR_GET_ROOT, *P_APP_MGR_GET_ROOT;

msgAppMgrSetIconBitmap

Specifies the large application icon bitmap.
Takes OBJECT, returns STATUS.

```c
#define msgAppMgrSetIconBitmap MakeMsg(clsAppMgr, 10)
```

msgAppMgrSetSmallIconBitmap

Specifies the small application icon bitmap.
Takes OBJECT, returns STATUS.

```c
#define msgAppMgrSetSmallIconBitmap MakeMsg(clsAppMgr, 11)
```
**msgAppMgrRevert**
Reverts a document to its most recently filed copy.
Takes P_FS_LOCATOR, returns STATUS.

```
#define msgAppMgrRevert MakeMsg(clsAppMgr, 12)
```

**msgAppMgrRenumber**
Renumbers an application hierarchy.
Takes P_FS_LOCATOR, returns STATUS.

```
#define msgAppMgrRenumber MakeMsg(clsAppMgr, 13)
```

**msgAppMgrDumpSubtree**
Dumps the attributes of a subtree of documents.
Takes P_FS_LOCATOR, returns STATUS.

```
#define msgAppMgrDumpSubtree MakeMsg(clsAppMgr, 14)
```

**msgAppMgrGetResList**
Creates a resource list, given an application UUID.
Takes P_APP_MGR_GET_RES_LIST, returns STATUS.

```
#define msgAppMgrGetResList MakeMsg(clsAppMgr, 15)
```

**Arguments**
```c
typedef struct APP_MGR_GET_RES_LIST {
    UUID    appUUID;    // App uuid.
    OBJECT  resList;    // in/out: resource file list.
} APP_MGR_GET_RES_LIST, *P_APP_MGR_GET_RES_LIST;
```

**Comments**
The resource list will contain the document resource file, the application resource file, the preference resource file, and the system resource file. `resList` should be set to `objNull` or a well-known uid.
This file contains the API definition for clsAppMonitor.

clsAppMonitor inherits from clsApp.

Provides the standard behavior for an application's monitor object.

You create an application monitor when you call AppMonitorMain from your main routine, when processCount is zero. An application monitor drives application installation and helps with deinstallation. It also controls displaying global application options, maintaining global state, and importing files.

You should subclass clsAppMonitor if your application needs to do a more sophisticated installation (such as installing shared dictionaries or data files), to support file import, to set and save global application configurations, and to provide file converters. See the section below on Subclassing.

clsAppMonitor's Lifecycle

Every application has a single instance of its application monitor class alive as long as the application is installed. The app monitor object is owned by the application's processCount 0 process. Clients can get the uid of the app monitor object by sending msgAppMgrGetMetrics to an application's class.

clsAppMonitor is a descendant of clsApp. It makes use of the standard Application Framework lifecycle messages to perform some of its functions:

msgAppInit Install the application.

msgAppRestore Reinitialize the application after a warm-boot.

msgAppOpenTo Display global application option sheet.

msgAppCloseTo Take down global application option sheet.

Note: msgAppTerminate must *never* be sent directly to the app monitor. Use msgAMTerminateOK and msgAMTerminate instead.

Application Installation

Application installation is performed as follows:

1. Somebody sends msgIMInstall to theInstalledApps. theInstalledApps creates an application directory in the selected volume under \penpoint\sys\app, copies the application's resource file into the application directory, and installs the application's code. See appimgr.h for details.

2. When the code is installed, the operating system creates the application's first process (processCount = 0) and begins execution of the main() routine. The application installs its classes and calls AppMonitorMain(). AppMonitorMain never returns; it creates the app monitor object and goes into an object dispatch loop (see clsmgr.h).

3. The Application Framework sends msgAppInit to the app monitor. This initiates the app monitor's installation sequence.
4. The app monitor self sends msgAMLoadInitDll. This causes an optional initialization .dll to be run and then be unloaded.

5. The app monitor self sends msgAMPopupOptions. If a descendant wants to pop up the app monitor global option sheet, it must handle this message and set pArgs to true, then pass it on to its ancestor. This will cause the option sheet protocol (msgOptionAddCards, etc) to be sent to the app monitor.

6. The app monitor self sends msgAMLoadMisc. This causes any files that the application has in the MISC directory to be copied into the app directory in the selected volume.

7. The app monitor self sends msgLoadAuxNotebooks, which causes msgLoadStationery and msgLoadHelp to be sent to self. msgLoadStationery causes all the stationery and accessory templates that do not have an anmAttrNoLoad attribute on them to be loaded into the machine. Stationery is stored in the STATNRY directory; Accessories are stored in the ACCESSRY directory. msgLoadHelp causes all Help Notebook documents and templates that do not have the anmAttrNoLoad attribute set to be loaded into the Help Notebook.

8. The app monitor self sends msgAMLoadFormatConverters and msgAMLoadOptionalDlls. These messages are currently not implemented by clsAppMonitor; descendants can deal with them if desired. There might be default superclass behavior in the future.

---

**Stationery, Accessory, and Help Documents**

Stationery and Accessory documents can either be saved document instances (typically copied out to a distribution disk with the Connections Notebook), or plain directories containing files that the application knows about.

Help documents can be directories containing ASCII or RTF files, or PenPoint documents. These items must be located in the application’s installation directory in subdirectories called STATNRY, ACCESSRY, and HELP.

---

**Subclassing clsAppMonitor**

The app monitor is an excellent place to add global application control and synchronization functions, since it is always around and easily accessible. For instance, if an application wants to access some application-specific shared data (such as a list of worldwide telephone country codes), the app monitor for the application could manage this data and provide an API to access it.

Applications can have a global application option sheet automatically displayed when the application is installed by handling msgAMPopupOptions. A special resource is written into the application’s resource file after this occurs, inhibiting subsequent popups if the resource file is copied to the application’s installation directory. clsAppMon does not provide any default cards; you must provide at least one if you handle msgAMPopupOptions.

If you display a user interface from your app monitor you will probably have to turn on the fullEnvironment app manager flag when you create your main application class. If this flag is false then the app monitor will run in a stripped down process environment. This saves a substantial amount of memory, but does not have process-local resources such as the ProcessResList.

If you subclass clsAppMonitor, you must specify your descendant’s class name when you call AppMonitorMain. The first parameter to this routine is the global well-known name of your application class. The second is the global well-known name of your descendant of clsAppMonitor. If you do not
subclass clsAppMonitor, pass objNull for the second parameter. The AppMonitorMain routine will know to create a default application monitor.

```c
#ifndef APPMON_INCLUDED
#define APPMON_INCLUDED
#ifndef FS_INCLUDED
#include <fs.h>
#endif
#ifndef OPTION_INCLUDED
#include <option.h>
#endif
```

### Common #defines and typedefs
This attribute represents the last modified date that a piece of stationery had when it was installed on the machine.

```c
#define amAttrDateTimeLoaded FSMakeFix32Attr(clsAppMonitor, 2)
```

### Application Framework Messages

#### msgAppInit
Installs the application.

Takes DIR_HANDLE, returns STATUS.

**Comments**
This message is sent once and only once by the system, when the application is first installed from disk. The app monitor initializes its instance data, runs the installation protocol (msgAMLoadInitDLL, msgAMLoadStationery, msgAMLoadMisc, etc), adds this application to system lists, and signals the installation process to continue running.

Descendants: You can handle this message to perform any first-time initialization. The ancestor must be called before your handler.

#### msgAppRestore
Reinitializes the application after a warm-boot.

Takes nothing, returns STATUS.

**Comments**
This message is sent by the system when a warm-boot occurs. The app monitor initializes its instance data and signals the system warm-boot process to proceed.

Descendants: You can handle this message and perform any first-time initialization. The ancestor must be called before your handler.

#### msgAppOpen
Displays the global configuration option sheet.

Takes P_APP_OPEN, returns STATUS.

**Comments**
This message is self-sent by msgAMPopupOptions. It can also be sent by anyone else. The app monitor displays the application configuration option sheet (tagAppDocOptSheet).

Descendants: You normally do not handle this message. To provide an option sheet, see msgAMPopupOptions.
msgAppClose
Removes the global configuration option sheet.
Takes nothing, returns STATUS.
Comments
This message is self-sent by msgAMPopupOptions. It can also be sent by anyone else.
Descendants: You normally do not handle this message.

Import Messages

msgImportQuery
Determines if a file can be imported by the application.
Takes P_IMPORT_QUERY, returns STATUS.
Comments
The app monitor forwards msgImportQuery to its class as a class message. If it isn’t handled there, the
app monitor sends back "No" to all import requests. In the future there will be support to run through
any of the file translators that the application has loaded.
Descendants: You normally do not handle this message.
See Also
import.h

msgImport
Imports a file.
Takes P_IMPORT_DOC, returns STATUS.
Comments
The app monitor first creates a new document object and activates it. It then forwards msgImport to
the document. Next, it sends msgAppMgrShutdown to both save the document and shut it down.
Descendants: You normally do not handle this message.
See Also
import.h

App Monitor Messages

msgAMGetMetrics
Gets the app monitor’s metrics.
Takes P_AM_METRICS, returns STATUS.
#define msgAMGetMetrics
MakeMsg(clsAppMonitor, 1)
Arguments
typedef struct AM_METRICS {
  CLASS appClass; // Main application class.
  OBJECT handle; // This app's handle in theInstalledApps.
  U32 unused2;
  U32 unused3;
  U32 unused4;
  U16 unused;
} AM_METRICS, *P_AM_METRICS;
Comments
Descendants: You normally do not handle this message.
msgAMGetInstallDir
Creates a directory handle on the application's installation directory.
Takes P_OBJECT, returns STATUS.

`#define msgAMGetInstallDir MakeMsg(clsAppMonitor, 2)`

Comments
The app monitor creates a clsDirHandle object which references the location on external media that the application was installed from. If the external volume is not connected, the user is asked to attach it.

If this application 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.

msgAMLoadInitDll
Loads, runs, and unloads an optional dll initialization routine.
Takes OBJECT, returns STATUS.

`#define msgAMLoadInitDll MakeMsg(clsAppMonitor, 4)`

Comments
The app monitor looks for an init.dll file in the application's directory (which is specified in pArgs). If it is found, the DllMain routine for this dll is run. The dll is then unloaded.

Descendants: You normally do not handle this message.

Return Value
stsOK  Either the dll initialization was not found or it was found and run successfully.

msgAMLoadMisc
Load the application's miscellaneous files.
Takes nothing, returns STATUS.

`#define msgAMLoadMisc MakeMsg(clsAppMonitor, 5)`

Comments
If a directory called MISC exists, the app monitor copies this directory into the in-memory application directory.

Descendants: You normally do not handle this message. However, you can create the MISC directory and place in it files that all of your documents need to use. For example, your documents may need to reference a file that contains all of the postal/zip codes for a country.

Return Value
stsOK  Either the MISC directory was not found or was found and copied successfully.

msgAMLoadStationery
Loads stationery and accessory templates.
Takes nothing, returns STATUS.

`#define msgAMLoadStationery MakeMsg(clsAppMonitor, 6)`
The app monitor looks for stationery in a directory named STATNRY and accessories in a directory named ACCESSRY in the app's directory. It copies any templates that are not marked with the noLoad attribute from these directories to the Stationery and Accessories notebooks.

A template is a subdirectory with a either a complete, saved document or any kind of file that the application can read.

If appMgrMetrics.flags.stationery is true, the app monitor creates a default piece of stationery (an empty document of its application type). Similarly, if appMgrMetrics.flags.accessory is true, the app monitor places an empty document in the Accessories notebook.

Descendants: You normally do not handle this message.

msgAMRemoveStationery

Removes all the stationery and accessory templates for this application.

Takes nothing, returns STATUS.

#define msgAMRemoveStationery MakeMsg(clsAppMonitor, 7)

The app monitor removes the stationery notebook section for this application, which removes the stationery loaded in msgAMLoadStationery and any user-defined stationery. It then removes all of this application's documents from the Accessories notebook (thereby removing templates loaded in msgAMLoadStationery and any documents that the user placed there).

Descendants: You normally do not handle this message.

msgAMLoadHelp

Loads the application's help into the Help Notebook.

Takes nothing, returns STATUS.

#define msgAMLoadHelp MakeMsg(clsAppMonitor, 8)

The app monitor looks for a HELP subdirectory in the application's directory. If HELP exists, the app monitor copies all of the help templates that are not marked with the noLoad attribute to the Help Notebook. Help templates can be directories with ASCII, RTF or saved MiniText documents in them.

Descendants: You normally do not handle this message.

msgAMRemoveHelp

Removes all Help Notebook items for this application.

Takes nothing, returns STATUS.

#define msgAMRemoveHelp MakeMsg(clsAppMonitor, 9)

The app monitor removes all of this application's items from the Help Notebook.

Descendants: You normally do not handle this message.

msgAMPopupOptions

Pops up a global option sheet the first time the app is installed.

Takes P_BOOLEAN, returns STATUS.

#define msgAMPopupOptions MakeMsg(clsAppMonitor, 17)
If `pArgs` is false, the app monitor does not do anything. If it is true, the app monitor pops up the global option sheet, then writes a resource in the application’s resource file which inhibits subsequent popups.

Descendants: If you want to allow the user to configure (or check the configuration of) the application as it is being installed, you need to handle this message. In your handler, you should set `pArgs` to true and then call the ancestor. You also need to create an option sheet resource with a tag of `tagAppDocOptSheet` (in your application’s `msgAppAddCards` handler).

You can have the option sheet to always pop up (even after the first time the user installs the application) by not calling the ancestor and popping up the option sheet yourself with:

```c
ObjCallRet(msgAppOpenTo, self, (P_ARGS) appOpenToFloating, s);
```

### msgAMLoadAuxNotebooks

Loads items into auxiliary notebooks.

Takes nothing, returns STATUS.

```c
#define msgAMLoadAuxNotebooks MakeMsg(clsAppMonitor, 14)
```

Comments: The app monitor self sends `msgAMLoadStationery` and `msgAmLoadHelp` to load the application’s stationery, accessory, and help templates.

Descendants: You normally do not handle this message.

### msgAMLoadFormatConverters

Loading file format converter .dlls.

Takes nothing, returns STATUS.

```c
#define msgAMLoadFormatConverters MakeMsg(clsAppMonitor, 10)
```

Comments: Currently, the app monitor does not do anything in response to this message. It will do something in the future.

Descendants: You normally do not handle this message.

### msgAMUnloadFormatConverters

Unloads file format converter .dlls.

Takes nothing, returns STATUS.

```c
#define msgAMUnloadFormatConverters MakeMsg(clsAppMonitor, 11)
```

Comments: Currently, the app monitor does not do anything in response to this message. It will do something in the future.

Descendants: You normally do not handle this message.

### msgAMLoadOptionalDlls

Loads an application’s optional .dlls.

Takes nothing, returns STATUS.

```c
#define msgAMLoadOptionalDlls MakeMsg(clsAppMonitor, 12)
```

Comments: Currently, the app monitor does not do anything in response to this message. It will do something in the future.

Descendants: You normally do not handle this message.
msgAMUnloadOptionalDlls
Unloads an application's optional .dlls.
Takes nothing, returns STATUS.
#define msgAMUnloadOptionalDlls MakeMsg(clsAppMonitor, 13)

Currently, the app monitor does not do anything in response to this message. It will do something in the future.

Descendants: You normally do not handle this message.

msgAMTerminateOK
Asks if this application is willing to terminate.
Takes P_OBJECT, returns STATUS.
#define msgAMTerminateOK MakeMsg(clsAppMonitor, 20)

Deinstallation is a two phase process. All applications and services that are to be deinstalled together get the chance to veto. This message is sent to an application monitor to see if it wishes to veto.

By default, the app monitor unconditionally terminates all of its application's instances. To do so, it sends msgAppMgrShutdown to its application class for each of its active documents.

Descendants: If you want to be given the chance to terminate the application, you should handle this message. In your handler, if you decide that you want to terminate, you simply pass the message on to your ancestor.

You can veto the termination by returning anything other than stsOK and by not passing the message on to your ancestor. If you veto, you must set pArgs to the uid of the object that was responsible for the veto, which is typically self.

See Also
msgAMTerminate

msgAMTerminate
Terminates this application.
Takes nothing, returns STATUS.
#define msgAMTerminate MakeMsg(clsAppMonitor, 21)

Deinstallation is a two phase process. All applications and services that are to deinstalled together get the chance to veto. This message is sent to an application monitor after everyone has agreed to the deinstallation.

This message unconditionally terminates the application in the final phase of deinstallation. The app monitor self sends msgAMRemoveStationery and msgAMRemoveHelp, and then calls OSTaskTerminate to kill the application's processCount 0 task.

Descendants: You should handle this message to remove anything you have loaded. The ancestor must be called after your handler.

See Also
msgAMTerminateOK
msgAMTerminateVetoed
Sent when the application termination sequence is vetoed.
Takes P_AM_TERMINATE_VETOED, returns STATUS.

#define msgAMTerminateVetoed MakeMsg(clsAppMonitor, 22)

typedef struct AM_TERMINATE_VETOED {
    OBJECT vetoer;    // Object or class that vetoed the deinstallation.
    STATUS status;    // Veto status.
} AM_TERMINATE_VETOED, *P_AM_TERMINATE_VETOED;

Comments
When one of the applications or services that are deinstalled together vetoes termination, the Application Framework sends this message to those applications and services.

pArgs->vetoer gives the uid of the object or class that vetoed the deinstallation. pArgs->status gives the return status of the veto. The app monitor does not do anything in response to this message.

Descendants: You can handle this message if you wish. If you handled msgAMTerminateOK, and changed anything because you thought you were about to be terminated, you should handle this message to change things back to the way they were.

See Also
msgAMTerminateOK

Tags

#define tagAMFirstTime MakeTag(clsAppMonitor, 2)
APPTAG.H

This file contains constant tags used by the Application Framework.

There are three kinds of tags in this file:

- Resource tags
- Window tags
- Quick help tags

Resource tags are used to construct resource identifiers (resID's) that identify well-known resources in the system resource file. Developers can use these tags to read a copy of any of these resources from their document's resList (see app.h and resfile.h).

Window tags are used as arguments to `msgWinFindTag` to locate well-known windows. For example, all the standard application menus (SAMS) are tagged so they can be programatically located and changed or removed by an application.

Quick help tags are used for two purposes: (1) to construct the resource identifier used to read a quick-help string from the document's resList, and (2) stored in clsGWin as the quick-help tag (see gwin.h).

All the resource ids in this file are constructed by (see resfile.h):

```
MakeWknResId(resId, appResId, tag);
```

To write an object resource:

```
write.resId = tagAppObject;
write.mode = resWriteObjectOnce;
write.object = objectToWrite;
ObjCallRet(msgResWriteObject, file, &write, s);
```

To read an object resource:

```
ObjectCall(msgNewDefaults, clsObject, &read.new);
read.resId = tagAppObject;
read.mode = resReadObjectOnce;
ObjCallRet(msgResReadObject, resList, &read, s);
newObject = read.new.uid;
```

```#ifndef APPTAG_INCLUDED
#define APPTAG_INCLUDED
#endif
```

### Resource Tags

#### clsApp Resource Identifiers

Used to construct Application Framework resID's (see above).

```#define appResId
// next: 194```
Tags used by StdMsg.

```c
#define tagAppDeleteRequest MakeDialogTag(clsAppMgr, 0)
#define tagAppDeleteSectRequest MakeDialogTag(clsAppMgr, 1)
#define tagAppRevertRequest MakeDialogTag(clsAppMgr, 2)
#define tagAppSystemShutdownRequest MakeDialogTag(clsAppMgr, 3)
#define tagAppSystemSoftShutdownRequest MakeDialogTag(clsAppMgr, 4)
```

Miscellaneous tags.

```c
#define tagAppObject MakeTag(clsApp, 138)
#define tagAppClass MakeTag(clsApp, 118)
#define tagAppQHAppClass MakeTag(clsApp, 155)
#define tagAppTitleBar MakeTag(clsApp, 119)
#define tagAppMoveIconMarquee MakeTag(clsApp, 135)
#define tagAppCopyIconMarquee MakeTag(clsApp, 136)
#define tagAppPrintMetrics MakeTag(clsApp, 139)
#define tagAppMenuImport MakeTag(clsApp, 148)
#define tagAppMenuExport MakeTag(clsApp, 149)
```

These identify each item in the SAMS menu bar.

```c
#define tagAppMenuBar MakeTag(clsApp, 1)
#define tagAppMenuDocument MakeTag(clsApp, 2)
#define tagAppMenuEdit MakeTag(clsApp, 3)
#define tagAppMenuOptions MakeTag(clsApp, 4)
#define tagAppMenuCreate MakeTag(clsApp, 156)
```

These identify each item in the Document menu.

```c
#define tagAppMenuCheckpoint MakeTag(clsApp, 5)
#define tagAppMenuRevert MakeTag(clsApp, 6)
#define tagAppMenuPrint MakeTag(clsApp, 7)
#define tagAppMenuPrintSetup MakeTag(clsApp, 8)
#define tagAppMenuSend MakeTag(clsApp, 9)
#define tagAppMenuAbout MakeTag(clsApp, 10)
```

These identify each item in the Edit menu.

```c
#define tagAppMenuUndo MakeTag(clsApp, 11)
#define tagAppMenuSelectAll MakeTag(clsApp, 12)
#define tagAppMenuMove MakeTag(clsApp, 13)
#define tagAppMenuCopy MakeTag(clsApp, 14)
#define tagAppMenuDelete MakeTag(clsApp, 124)
#define tagAppMenuSearch MakeTag(clsApp, 125)
#define tagAppMenuSpell MakeTag(clsApp, 126)
```

These identify SAMS option sheets.

```c
#define tagAppAboutOptSheet MakeTag(clsApp, 120)
#define tagAppDocOptSheet MakeTag(clsApp, 121)
#define tagAppPrintSetupOptSheet MakeTag(clsApp, 122)
#define tagAppIconOptSheet MakeTag(clsApp, 123)
```

These identify each card in the Document option sheet.

```c
#define tagAppOptControlsCard MakeTag(clsApp, 142)
#define tagAppOptAccessCard MakeTag(clsApp, 143)
#define tagAppOptCommentsCard MakeTag(clsApp, 144)
#define tagAppOptIconCard MakeTag(clsApp, 147)
#define tagAppOptGotoButtonCard MakeTag(clsApp, 154)
#define tagAppOptIconWinCard MakeTag(clsApp, 172)
```

These identify each card in the About option sheet.

```c
#define tagAppOptInfoCard MakeTag(clsApp, 140)
#define tagAppOptAboutCard MakeTag(clsApp, 141)
```
These identify each card in the Print Setup option sheet.

#define tagAppOptPrintCard MakeTag(clsApp, 145)
#define tagAppOptHeadersCard MakeTag(clsApp, 146)
#define tagAppOptEmbeddieCard MakeTag(clsApp, 173)

These identify each item in the Borders & Controls card.

#define tagAppOptCtrls MakeTag(clsApp, 127)
#define tagAppOptCtrlsLabel MakeTag(clsApp, 128)
#define tagAppOptCtrlsOn MakeTag(clsApp, 129)
#define tagAppOptCtrlsOff MakeTag(clsApp, 130)
#define tagAppOptCtrlStyle MakeTag(clsApp, 131)
#define tagAppOptCtrlStyleLabel MakeTag(clsApp, 132)
#define tagAppOptCtrlTitleBar MakeTag(clsApp, 133)
#define tagAppOptCtrlMenuBar MakeTag(clsApp, 134)
#define tagAppOptCtrlScrollBars MakeTag(clsApp, 26)
#define tagAppOptCtrlCorkMargin MakeTag(clsApp, 27)
#define tagAppOptBorderStyle MakeTag(clsApp, 157)
#define tagAppOptBorderStyleLabel MakeTag(clsApp, 158)
#define tagAppOptBorderSingle MakeTag(clsApp, 159)
#define tagAppOptBorderDouble MakeTag(clsApp, 162)
#define tagAppOptBorderDashed MakeTag(clsApp, 161)
#define tagAppOptBorderNone MakeTag(clsApp, 160)

These identify each item in the Access card.

#define tagAppOptDelete MakeTag(clsApp, 28)
#define tagAppOptDeleteLabel MakeTag(clsApp, 29)
#define tagAppOptDeleteOn MakeTag(clsApp, 30)
#define tagAppOptDeleteOff MakeTag(clsApp, 31)
#define tagAppOptReadOnly MakeTag(clsApp, 32)
#define tagAppOptReadOnlyLabel MakeTag(clsApp, 33)
#define tagAppOptReadOnlyOn MakeTag(clsApp, 34)
#define tagAppOptReadOnlyOff MakeTag(clsApp, 35)
#define tagAppOptHotMode MakeTag(clsApp, 36)
#define tagAppOptHotModeLabel MakeTag(clsApp, 37)
#define tagAppOptHotModeOn MakeTag(clsApp, 38)
#define tagAppOptHotModeOff MakeTag(clsApp, 39)

These identify each item in the Comments card.

#define tagAppOptCommentsTable MakeTag(clsApp, 191)
#define tagAppOptTitle MakeTag(clsApp, 40)
#define tagAppOptTitleLabel MakeTag(clsApp, 41)
#define tagAppOptAuthor MakeTag(clsApp, 42)
#define tagAppOptAuthorLabel MakeTag(clsApp, 43)
#define tagAppOptComments MakeTag(clsApp, 44)
#define tagAppOptCommentsSWin MakeTag(clsApp, 190)
#define tagAppOptCommentsLabel MakeTag(clsApp, 45)

These identify each item in the About/Document card.

#define tagAppOptCreated MakeTag(clsApp, 46)
#define tagAppOptCreatedLabel MakeTag(clsApp, 47)
#define tagAppOptModified MakeTag(clsApp, 48)
#define tagAppOptModifiedLabel MakeTag(clsApp, 49)
#define tagAppOptFileSize MakeTag(clsApp, 50)
#define tagAppOptFileSizeLabel MakeTag(clsApp, 51)
#define tagAppOptActiveSize MakeTag(clsApp, 52)
#define tagAppOptActiveSizeLabel MakeTag(clsApp, 53)

These identify each item in the About/Application card.

#define tagAppOptApp MakeTag(clsApp, 54)
#define tagAppOptAppLabel MakeTag(clsApp, 55)
#define tagAppOptVersion MakeTag(clsApp, 56)
#define tagAppOptVersionLabel MakeTag(clsApp, 57)
These identify each item in the Icon Window Layout card.

#define tagAppIconWinLayout MakeTag(clsApp, 163)
#define tagAppIconWinLayoutLabel MakeTag(clsApp, 164)
#define tagAppIconWinTTToB MakeTag(clsApp, 165)
#define tagAppIconWinBTToT MakeTag(clsApp, 166)
#define tagAppIconWinUnconstrained MakeTag(clsApp, 167)
#define tagAppIconWinStyle MakeTag(clsApp, 168)
#define tagAppIconWinStyleLabel MakeTag(clsApp, 169)
#define tagAppIconWinKeepSame MakeTag(clsApp, 170)
#define tagAppIconWinOpenInProgress MakeTag(clsApp, 171)

These identify each item in the Print Setup cards.

#define tagAppPaperSize MakeTag(clsApp, 66)
#define tagAppPaperSizeLabel MakeTag(clsApp, 67)
#define tagAppPaperWidth MakeTag(clsApp, 68)
#define tagAppPaperHeight MakeTag(clsApp, 69)
#define tagAppTopMargin MakeTag(clsApp, 70)
#define tagAppTopMarginLabel MakeTag(clsApp, 71)
#define tagAppBottomMargin MakeTag(clsApp, 72)
#define tagAppBottomMarginLabel MakeTag(clsApp, 73)
#define tagAppLeftMargin MakeTag(clsApp, 74)
#define tagAppLeftMarginLabel MakeTag(clsApp, 75)
#define tagAppRightMargin MakeTag(clsApp, 76)
#define tagAppRightMarginLabel MakeTag(clsApp, 77)
#define tagAppLeftHeader MakeTag(clsApp, 78)
#define tagAppLeftHeaderLabel MakeTag(clsApp, 79)
#define tagAppCenterHeader MakeTag(clsApp, 80)
#define tagAppCenterHeaderLabel MakeTag(clsApp, 81)
#define tagAppRightHeader MakeTag(clsApp, 82)
#define tagAppRightHeaderLabel MakeTag(clsApp, 83)
#define tagAppLeftFooter MakeTag(clsApp, 84)
#define tagAppLeftFooterLabel MakeTag(clsApp, 85)
#define tagAppCenterFooter MakeTag(clsApp, 86)
#define tagAppCenterFooterLabel MakeTag(clsApp, 87)
#define tagAppRightFooter MakeTag(clsApp, 88)
#define tagAppRightFooterLabel MakeTag(clsApp, 89)
#define tagAppEmbedVisible MakeTag(clsApp, 90)
#define tagAppEmbedVisibleLabel MakeTag(clsApp, 91)
#define tagAppOrientation MakeTag(clsApp, 92)
#define tagAppOrientationLabel MakeTag(clsApp, 93)
#define tagAppHeaderMargin MakeTag(clsApp, 94)
#define tagAppHeaderMarginLabel MakeTag(clsApp, 95)
#define tagAppFooterMargin MakeTag(clsApp, 96)
#define tagAppFooterMarginLabel MakeTag(clsApp, 97)
#define tagAppHeaderFont MakeTag(clsApp, 98)
#define tagAppHeaderFontLabel MakeTag(clsApp, 99)
#define tagAppHeaderSize MakeTag(clsApp, 100)
#define tagAppHeaderSizeLabel MakeTag(clsApp, 101)
#define tagAppFirstPage MakeTag(clsApp, 102)
#define tagAppFirstPageLabel MakeTag(clsApp, 103)
#define tagAppOtherLabel MakeTag(clsApp, 104)
#define tagAppEmbedLoc MakeTag(clsApp, 174)
#define tagAppEmbedLocLabel MakeTag(clsApp, 175)
#define tagAppEmbedApplyTo MakeTag(clsApp, 176)
#define tagAppHeaderMarginOtherButton MakeTag(clsApp, 177)
#define tagAppHeaderMarginOtherField
#define tagAppFooterMarginOtherButton
#define tagAppFooterMarginOtherField
#define tagAppTopMarginOtherButton
#define tagAppTopMarginOtherField
#define tagAppBottomMarginOtherButton
#define tagAppBottomMarginOtherField
#define tagAppLeftMarginOtherButton
#define tagAppLeftMarginOtherField
#define tagAppRightMarginOtherButton
#define tagAppRightMarginOtherField
#define tagAppEmbedApplyToLabel
These identify each item in the Icon option card.
#define tagApplconTitle
#define tagApplconTitleLabel
#define tagApplconOpen
#define tagApplconOpenLabel
#define tagApplconOpenInPlace
#define tagApplconOpenFloating
#define tagApplconType
#define tagApplconTypeLabel
#define tagApplconTypePictAndTitle
#define tagApplconTypePictOnly
#define tagApplconTypeSmallPictAndTitle
#define tagApplconTypeSmlPictOverTitle
#define tagApplconTypeSmallPictOnly
These identify each item in the Goto Button option card.
#define tagAppGotoButtonTitle
#define tagAppGotoButtonTitleLabel
#define tagAppGotoButtonTargetDoc
#define tagAppGotoButtonTargetDocLabel
#define tagAppGotoButtonBorderLabel
#define tagAppGotoButtonBorder
#define tagAppGotoButtonSquare
#define tagAppGotoButtonRound
#define tagAppGotoButtonHRound
#define tagAppGotoButtonNone
These identify various bitmaps.
#define tagAppIconBitmap
#define tagAppSmallIconBitmap
#define tagAppDefaultDocIconBitmap
#define tagAppDefaultDocSmallIconBitmap
#define tagAppMoveIconBitmap
#define tagAppCopyIconBitmap
#define tagAppLinkIconBitmap
#define tagAppClosedFolderBitmap
#define tagAppClosedFolderSmallBitmap
#define tagAppOpenFolderBitmap
#define tagAppOpenFolderSmallBitmap
Tags used during the creation of a document to get default values for some fields from the application resource file.
#define tagAppDefaultDocName
#define tagAppDisplayedAppName
This file contains the API definition for clsAppWin.

clsAppWin inherits from clsCustomLayout.

Provides support for embedded applications.

"AppWin" stands for Application Window.

Introduction

clsAppWin is an embedded window that manages an embedded document. It shrink-wraps around a clsIcon object to display an icon to the user, like those on the bookshelf or embedded in a document. When an icon with style awOpenInPlace is tapped, the application window destroys the icon and opens the associated document into itself. The application window then shrink-wraps around the document's main window.

Application Windows live in the process space and are filed with the embeddor document.

An application window reads its icon bitmap from metrics.resList of OStThisApp() in response to msgIconProvideBitmap (see icon.h). It uses the following resID (see apptag.h):

```cpp
MakeWknResIdX(read.resId, appResId, tagAppIconBitmap);
```

This bitmap is usually found in the app.res file of the application class for the associated document. The document can override this bitmap by filing a resource with the above resId into its doc.res file.

AppWins can also store their own private bitmaps. Use msgAppWinSetIconBitmap to give an application window a bitmap. This bitmap object will be filed by the application window. If an application window has its own bitmap object, it will not read from the resList.

```cpp
#ifndef APPWININCLUDED
#define APPWININCLUDED
#include <clayout.h>
#endif
```

Common #defines and typedefs

typedef OBJECT APP_WIN, *P_APP_WIN;

Application Window States

These are the valid states for an application window.

```cpp
#define awClosed 0
#define awOpenedFloating 1
#define awOpenedInPlace 2
#define awOpenedInPlaceFloating 3
```
Application Window Open Styles

These are the valid styles for directing an application window how to open.

#define awOpenInPlace 0
#define awOpenFloating 1

Application Window Icon Types

These are the valid icon types.

#define awPictAndTitle 0
#define awPictOnly 1
#define awSmallPictAndTitle 2
#define awSmallPictOnly 3
#define awSmallPictOverTitle 4

Application Window Style Structure

This structure defines the various application window styles.

typedef struct APP_WIN_STYLE {
    U16 open 2;     // Open style.
    U16 type 4;     // Icon type.
    U16 openStyleLock 1;  // True = cannot change open style.
    U16 privatel 1;  // Reserved.
    U16 private2 1;  // Reserved.
    U16 reserved 7;  // Reserved.
} APP_WIN_STYLE, *P_APP_WIN_STYLE;

Messages

msgNew

Creates a new Application Window.

Takes P_APP_WIN_NEW, returns STATUS. Category: class message.

typedef struct APP_WIN_NEW ONLY {
    UUID appUUID;  // App uuid.
    APP_WIN_STYLE style;  // Application Window style.
    U16 state;     // Application Window state.
    char label[nameBufLength];  // Icon label.
    U32 reserved[4];  // Reserved.
} APP_WIN_NEW ONLY, *P_APP_WIN_NEW ONLY;
#define appWinNewFields \
    customLayoutNewFields \
    APP_WIN_NEW ONLY appWin;

typedef struct APP_WIN_NEW {
    appWinNewFields \[
    APP_WIN_NEW ONLY appWin;
} APP_WIN_NEW, *P_APP_WIN_NEW;

msgNewDefaults

Initializes the APP_WIN_NEW structure to default values.

Takes P_APP_WIN_NEW, returns STATUS. Category: class message.

typedef struct APP_WIN_NEW {
    appWinNewFields \[
} APP_WIN_NEW, *P_APP_WIN_NEW;
Comments

Zeroes out pArgs->appWin and sets

```c
pArgs->win.flags.style |= wsCaptureGeometry
| wsSendGeometry
| wsShrinkWrapWidth
| wsShrinkWrapHeight;
pArgs->win.flags.input |= inputHoldTimeout;
pArgs->embeddedWin.style.embeddee = true;
pArgs->embeddedWin.style.moveable = true;
pArgs->embeddedWin.style.copyable = true;
pArgs->border.style.previewAlter = bsAlterNone;
pArgs->border.style.selectedAlter = bsAlterNone;
pArgs->appWin.style.open = awOpenInPlace;
pArgs->appWin.style.type = awSmallPictAndTitle;
pArgs->appWin.state = awClosed;
```

**msgAppWinGetMetrics**

Passes back an application window's metrics.

Takes P_APP_WIN_METRICS, returns STATUS.

```c
#define msgAppWinGetMetrics MakeMsg(clsAppWin, 1)

typedef struct APP_WIN_METRICS {
    UUID appUUID;        // Application uuid.
    OBJCET icon;         // Application Window icon.
    OBJCET iconBitmap;   // Icon bitmap.
    OBJCET smalllconBitmap; // Small icon bitmap.
    OBJCET appClass;     // Application class.
    APP_WIN_STYLE style; // Application Window style.
    U16 state;           // Application Window state.
    char label[nameBufLength]; // Icon label.
    U32 reserved[4];     // Reserved.
} APP_WIN_METRICS, *P_APP_WIN_METRICS;
```

**msgAppWinGetState**

Passes back an application window's state.

Takes P_U16, returns STATUS.

```c
#define msgAppWinGetState MakeMsg(clsAppWin, 2)
```

Comments

Possible values are described in Application Window States, above.

**msgAppWinSetState**

Specifies an application window's state.

Takes U16, returns STATUS.

```c
#define msgAppWinSetState MakeMsg(clsAppWin, 3)
```

Comments

Possible values are described in Application Window States, above.

**msgAppWinGetStyle**

Passes back an application window's style.

Takes P_APP_WIN_STYLE, returns STATUS.

```c
#define msgAppWinGetStyle MakeMsg(clsAppWin, 4)
```
typedef struct APP_WIN_STYLE {
    U16 open : 2;   // Open style.
    U16 type : 4;   // Icon type.
    U16 openStyleLock : 1; // True = cannot change open style.
    U16 private1 : 1; // Reserved.
    U16 private2 : 1; // Reserved.
    U16 reserved : 7; // Reserved.
} APP_WIN_STYLE, *P_APP_WIN_STYLE;

msgAppWinSetStyle
Specifies an application window's style.
Takes APP_WIN_STYLE, returns STATUS.
#define msgAppWinSetStyle MakeMsg(clsAppWin, 5)

typedef struct APP_WIN_STYLE {
    U16 open : 2;   // Open style.
    U16 type : 4;   // Icon type.
    U16 openStyleLock : 1; // True = cannot change open style.
    U16 private1 : 1; // Reserved.
    U16 private2 : 1; // Reserved.
    U16 reserved : 7; // Reserved.
} APP_WIN_STYLE, *P_APP_WIN_STYLE;

msgAppWinSetLabel
Specifies an application window's label.
Takes P_STRING, returns STATUS.
#define msgAppWinSetLabel MakeMsg(clsAppWin, 6)

msgAppWinSetIconBitmap
Specifies an application window's large icon bitmap.
Takes BITMAP, returns STATUS.
#define msgAppWinSetIconBitmap MakeMsg(clsAppWin, 7)

msgAppWinSetSmallIconBitmap
Specifies an application window's small icon bitmap.
Takes BITMAP, returns STATUS.
#define msgAppWinSetSmallIconBitmap MakeMsg(clsAppWin, 8)

msgAppWinOpen
Opens the document associated with an application window.
Takes nothing, returns STATUS.
#define msgAppWinOpen MakeMsg(clsAppWin, 9)

msgAppWinClose
Closes the document associated with an application window.
Takes nothing, returns STATUS.
#define msgAppWinClose MakeMsg(clsAppWin, 10)
**msgAppWinDelete**

Deletes an application window.

Takes BOOLEAN, returns STATUS.

```c
#define msgAppWinDelete MakeMsg(clsAppWin, 11)
```

Comments

If pArgs is true, msgAppWinDelete also deletes the associated document. If pArgs is false, msgAppWinDelete does not delete the document.

**msgAppWinSetUUID**

Specifies the UUID of the document to which an application window is linked.

Takes P_UUID, returns STATUS.

```c
#define msgAppWinSetUUID MakeMsg(clsAppWin, 12)
```

**msgAppWinCreateIcon**

Creates an application window's icon.

Takes P_UUID, returns STATUS.

```c
#define msgAppWinCreateIcon MakeMsg(clsAppWin, 13)
```

**msgAppWinDestroyIcon**

Destroys an application window's icon.

Takes P_UUID, returns STATUS.

```c
#define msgAppWinDestroyIcon MakeMsg(clsAppWin, 14)
```

**msgAppWinStyleChanged**

Notification that an application window style changed.

Takes OBJECT, returns STATUS.

```c
#define msgAppWinStyleChanged MakeMsg(clsAppWin, 15)
```

Comments

Application windows send this message to their observers whenever they receive msgAppWinSetStyle. Note that application icon option cards will send msgAppWinSetStyle to application windows whenever they cause the application window's icon style to change.

**msgAppWinEditName**

Pops up an edit pad to allow the user to rename the document associated with an application window.

Takes nothing, returns STATUS.

```c
#define msgAppWinEditName MakeMsg(clsAppWin, 16)
```
This file contains the API definition for clsCorkBoardWin.

clsCorkBoardWin inherits from clsIconWin.
"cbwin" stands for Cork Board Window.

**Introduction**

A cork board window is an icon window associated with a document. The cork board window puts embedded documents in a subdirectory of the document. This frees the document's application from having to manage the embedded windows and documents in the cork board window. The PenPoint Application Framework uses clsCorkBoardWin to implement the "cork margin" that all documents have by default.

Clients should rarely (if ever) need to create cork board windows themselves since the Application Framework has a clean UI and API for enabling the cork margin. clsApp creates a cork board window as the command bar of the document's main window (assuming the main window is a frame).

See Also

- app.h for messages to enable the cork margin of an application.

```c
#ifndef CBWIN_INCLUDED
#define CBWIN_INCLUDED
#ifndef ICONWIN_INCLUDED
#define ICONWIN_INCLUDED
#include <iconwin.h>
#endif
#endif
```

**Common \#defines and typedefs**

```c
typedef OBJECT CORKBOARD_WIN, *P_CORKBOARD_WIN;
```

**Quick Help Tags**

```c
#define qhCorkBoardWin MakeTag(clsCorkBoardWin, 1)
```

**Messages**

**msgNew**

Creates a cork board window.

Takes P_CORKBOARD_WIN_NEW, returns STATUS. Category: class message.

```c
typedef struct CORKBOARD_WIN_NEW_ONLY {
  U32 reserved1[4];
  U16 reserved2:16;
} CORKBOARD_WIN_NEW_ONLY, *P_CORKBOARD_WIN_NEW_ONLY;
#define corkboardWinNewFields
  iconWinNewFields
  CORKBOARD_WIN_NEW_ONLY corkboardWin;
```
typedef struct CORKBOARD_WIN_NEW {
    corkboardWinNewFields
} CORKBOARD_WIN_NEW, *P_CORKBOARD_WIN_NEW;

msgNewDefaults

Initializes the CORKBOARD_WIN_NEW structure to default values.

Takes P_CORKBOARD_WIN_NEW, returns STATUS. Category: class message.

typedef struct CORKBOARD_WIN_NEW {
    corkboardWinNewFields
} CORKBOARD_WIN_NEW, *P_CORKBOARD_WIN_NEW;

Zeroes out pArgs->corkboardWin and sets:

    pArgs->win.flags.style = wsShrinkWrapWidth;
    pArgs->win.flags.style = wsShrinkWrapHeight;
    pArgs->embeddedWin.style.quickMove = false;
    pArgs->border.style.topMargin = bsMarginSmall;
    pArgs->border.style.bottomMargin = bsMarginSmall;
    pArgs->border.style.leftMargin = bsMarginSmall;
    pArgs->border.style.rightMargin = bsMarginSmall;
    pArgs->iconWin.style.iconType = awSmallPictAndTitle;
    pArgs->iconWin.style.propagateIconType = true;
    pArgs->iconWin.style.allowOpenInPlace = false;
    pArgs->iconWin.style.constrainedLayout = true;

Messages from other classes

msgEmbeddedWinGetDest

Passes back the destination for embedded win move or copy.

Takes P_EMBEDDED_WIN_GET_DEST, returns STATUS.

clsCorkBoardWin responds by forcing the embedded document to be put in the embedding document's cork board subdirectory (appCorkboardDirName), creating this directory if it does not exist.

app.h definition of appCorkboardDirName string.
This file contains the app-level API for clsPrn.

clsPrn inherits from clsOBXService.

Very few developers would or should deal with instances of clsPrn. Its clients would be those writing print-wrapper applications or printer drivers. Both kinds of clients would need far more information than what could be described in a header file.

WARNING: the clsPrn API is likely to change in the future.

Much more functionality is in clsPrn but things not documented here are GO-internal.

```c
#ifndef CLSPRN_INCLUDED
#define CLSPRN_INCLUDED
#endif

#ifndef OSHEAP_INCLUDED
#include <osheap.h>
#endif

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

#ifndef OBXSVC_INCLUDED
#include <obxsvc.h>
#endif

#ifndef GEO_INCLUDED
#include <geo.h>
#endif

#pragma pack(1)

Common #defines and typedefs

Popular paper types

#define prnPaperLetter  0 // all printers
#define prnPaperLegal  1 // Pcl, Postscript
#define prnPaperExec  2 // Pcl
#define prnPaperA4  3 // Pcl, Postscript
#define prnPaperCom10  4 // Pcl
#define prnPaperMonarc  5 // Pcl
#define prnPaperC5  6 // Pcl
#define prnPaperDL  7 // Pcl
#define prnPaperB5  8 // Postscript
#define prnPaperLetterSmall  9 // Postscript
#define prnPaperA4Small 10 // Postscript
#define prnPaperTypeMax 10
#define prnPaperUserDefined Oxffff
```
Paper metrics

typedef struct PAPER_CONFIG {   // Paper configuration
    U16 type;                     // out: one of paper--- above
    U16 width, height;           // out: paper dimensions in mm
    U16 landscape;               // out:
    U16 nCopies;                 // out: # of copies to print
} PAPER_CONFIG, *P_PAPER_CONFIG;

Common header for all printer objects in its FS node

typedef struct PRN_FS_HDR {    // versioning
    U16 majorVersion,
        minorVersion;
    PAPER_CONFIG paper;
    U32 portMetricsFPos;
    U16 portMetricsSz;
} PRN_FS_HDR, *P_PRN_FS_HDR;

Error Messages

#define stsPrnStreamError MakeStatus(clsPrn, 1)
#define stsPrnNoStream MakeStatus(clsPrn, 2)
#define stsPrnUserAbort MakeStatus(clsPrn, 3)
#define stsPrnFntError MakeStatus(clsPrn, 4)

Dialog Messages

#define tagPrnManualFeedDialog MakeDialogTag(clsPrn, 0)

Quick Help Id's

#define tagQhPrnOptions MakeTag(clsPrn, 12)
#define tagQhPrnModel MakeTag(clsPrn, 13)
#define tagQhPrnPort MakeTag(clsPrn, 14)
// Epson driver specific
#define tagQhEpModelSheet MakeTag(clsEpson, 10)
#define tagQhEpModelList MakeTag(clsEpson, 11)
#define tagQhEpPaperFeed MakeTag(clsEpson, 15)
// Pcl driver specific
#define tagQhPclModelSheet MakeTag(clsPcl, 10)
#define tagQhPclModelList MakeTag(clsPcl, 11)
#define tagQhPclPaperFeed MakeTag(clsPcl, 15)
#define tagQhPclBinding MakeTag(clsPcl, 16)

msgNew

Creates a new printer object under the auspices of clsService.

Takes P_PRN_NEW, returns STATUS. Category: class message.

Arguments
typedef struct PRN_NEW_ONLY {
    U16 model;                     // in: model of printer (subclass defined)
    U16 fsNodeIsNew;               // out: first instantiation of object
    U16 filedDataSz;               // in: # of bytes to read/write from/to fs node
    P_PRN_FS_HDR pFileData;        // in: pointer to read/write filed data
} PRN_NEW_ONLY, *P_PRN_NEW_ONLY;
#define prnNewFields
    obxServiceNewFields
    PRN_NEWONLY prn;
typedef struct PRN_NEW {
    prnNewFields
} PRN_NEW, *P_PRN_NEW;
Device and Page Controls

msgPrnGetPaperConfig
Get the currently selected paper type, metrics and orientation.

Takes P_PAPER_CONFIG, returns STATUS. Category: class message.

```c
#define msgPrnGetPaperConfig MakeMsg(clsPrn, 2)
```

```
typedef struct PAPER_CONFIG {
    // Paper configuration
    U16 type; // out: one of paper--- above
    U16 width, height; // out: paper dimensions in mm
    U16 landscape; // out:
    U16 nCopies; // out: # of copies to print
} PAPER_CONFIG, *P_PAPER_CONFIG;
```

msgPrnSetPaperConfig
Set the currently selected paper type, metrics and orientation.

Takes P_PAPER_CONFIG, returns STATUS. Category: class message.

```c
#define msgPrnSetPaperConfig MakeMsg(clsPrn, 3)
```

```
typedef struct PAPER_CONFIG {
    // Paper configuration
    U16 type; // out: one of paper--- above
    U16 width, height; // out: paper dimensions in mm
    U16 landscape; // out:
    U16 nCopies; // out: # of copies to print
} PAPER_CONFIG, *P_PAPER_CONFIG;
```

msgPrnGetMetrics
Query a printer's device metrics.

Takes P_PRN_METRICS, returns STATUS. Category: descendant responsibility.

```c
#define msgPrnGetMetrics MakeMsg(clsPrn, 12)
```

```
typedef struct PRN_METRICS {
    // out: printer type (prnType---)
    U8 prnType;
    // out: capability bits
    U8 cap;
    // for dot matrix printers, this should be the pin size (8 or 24)
    U16 minBandSz;
    // out: pixel densities:
    U16 devPPMX,
        devPPMY;
    // out: unit is pixels/meter
    U16 nPlanes;
    // out: Number of planes of the device
    // Number of colors of the device (note: this number does not
    // necessarily equal (1 << devPlanes) because of halftoning
    U16 nColors;
    // out:
    // currently selected paper metrics in pixels
    U16 width, height;
    // out: printable area size
    U16 left, right, top, bottom;
    // out: unprintable margins
} PRN_METRICS, *P_PRN_METRICS;
```

#define prnTypeBm 0 // dot matrix printers
#define prnTypePcl 1 // HP laserjets
#define prnTypePscript 2 // Postscript

```c
#define prnDLBitmap 0x80 // can download bitmap font
#define prnDLOutline 0x40 // can download outline font
#define prnAutoRotate 0x20 // can print in rotated mode
#define prnAutoCopies 0x10 // can print multiple copies of a page
```
### msgPrnStartDoc

Prepare to start a new document.

Takes nothing, returns STATUS. Category: descendant responsibility.

```c
#define msgPrnStartDoc MakeMsg(clsPrn, 13)
```

### msgPrnEndDoc

End the currently printing document.

Takes nothing, returns STATUS. Category: descendant responsibility.

```c
#define msgPrnEndDoc MakeMsg(clsPrn, 14)
```

### msgPrnBeginPage

Prepare to start a new page.

Takes nothing, returns STATUS. Category: descendant responsibility.

```c
#define msgPrnBeginPage MakeMsg(clsPrn, 15)
```

### msgPrnShowPage

Output the current page.

Takes optional U16, returns STATUS. Category: descendant responsibility.

```c
#define msgPrnShowPage MakeMsg(clsPrn, 16)
#define prnNextSide 0 // if current side is front, print at back
#define prnFrontSide 1 // print on the front side
#define prnBackSide 2 // print on the back side
```

**Comments**
P_ARG is ignored if the printer can only do single-sided printing. P_ARG is a number specifying page duplexing for printers that can do double-sided printing.

### msgPrnSetCopyCount

Set the copy count.

Takes U32, returns STATUS. Category: descendant responsibility.

```c
#define msgPrnSetCopyCount MakeMsg(clsPrn, 17)
```

**Comments**
Valid only for devices with prnAutoCopies set in the metrics spec (msgPrnGetMetrics).

### msgPrnSetRotation

Tell device to operate in 0 or 90 degree mode.

Takes BOOLEAN, returns STATUS. Category: descendant responsibility.

```c
#define msgPrnSetRotation MakeMsg(clsPrn, 18)
```

**Comments**
Note: Change rotation only at the beginning of a new page.
For printers with the \texttt{prnAutoRotate} capability, sending this message in the middle of page formatting will cause undefined behavior of the printer. The co-ordinate system of the device will be rotated automatically.

For printers \textbf{without} the \texttt{prnAutoRotate} capability, this message will only affect the metrics returned by the \texttt{msgPrnGetMetrics} call. The co-ordinate system of the device remains unaffected.

\texttt{msgPrnStartDoc} will always put the device back into the non-rotated mode.

\section*{msgPrnEnumModels}

Enumerate the models that this class supports.

\begin{verbatim}
#define msgPrnEnumModels MakeMsg(clsPrn, 22)

typedef struct {
    U16 model;            // Out: model id (class defined)
    RES_ID iconResIdNormal; // Out: resId of model's normal icon
    RES_ID iconResIdSmall;  // Out: resId of model's small icon
    CHAR name[nameBufLength]; // Out: name of model
} PRN_MODEL, *P_PRN_MODEL;

typedef struct {
    max,                      // in = size of pModel[] array
    count;                    // in = # to return in array
    P_PRN_MODEL pModel;       // if count > max then memory may be allocated
    next;                     // out = # of valid entries in array
} P_PRN_ENUM_MODELS, *P_P_PRN_ENUM_MODELS;
\end{verbatim}

\section*{msgPrnGetModel}

Passes back the receiver's model.

\begin{verbatim}
#define msgPrnGetModel MakeMsg(clsPrn, 23)

typedef struct {
    U16 model;            // Out: model id (class defined)
    RES_ID iconResIdNormal; // Out: resId of model's normal icon
    RES_ID iconResIdSmall;  // Out: resId of model's small icon
    CHAR name[nameBufLength]; // Out: name of model
} PRN_MODEL, *P_PRN_MODEL;
\end{verbatim}

\section*{msgPrnMoveTo}

Move the printer's 'cursor' to the specified point.

\begin{verbatim}
#define msgPrnMoveTo MakeMsg(clsPrn, 19)
\end{verbatim}
msgPrnGetLptFontMetrics

Get the metrics/information of a given hardware font.

Takes P_PRN_TEXTOUT, returns STATUS. Category: descendant responsibility.

#define msgPrnGetLptFontMetrics MakeMsg(clsPrn, 20)

msgPrnLptTextOut

Output a line of text starting from where the printer was 'msgPrnMoveTo' last.

Takes P_PRN_TEXTOUT, returns STATUS. Category: descendant responsibility.

#define msgPrnLptTextOut MakeMsg(clsPrn, 21)

typedef struct PRN_TEXTOUT {
    U16 nChars; // in: number of characters to output
    P_CHAR pStr; // in: where the string is. Output will
                  // be terminated if a NULL is encountered,
                  // regardless of nChars.
    // additional font attributes subject to printer’s capabilities
    U16 fontSz; // in: big, medium or small
    U16 width, height; // out: character metrics in pixels
    // transformable attributes:
    // in: (msgPrnLptTextOut) specifies requested font attributes
    //    BOOLEAN underline, bold, italic;
    U16 underline, bold, italic; // out: (msgPrnGetFontMetrics) tells client what the selected
                                 //    // font is capable of
                                 //    BOOLEAN canUnderline, canBold, canItalic;
    U16 canUnderline, canBold, canItalic;
} PRN_TEXTOUT, *P_PRN_TEXTOUT;

#define prnLPTSmall 0 // fontSz field above
#define prnLPTMedium 1
#define prnLPTBig 2
#pragma pack()
EMBEDWIN.H

This file contains the API definition for clsEmbeddedWin. clsEmbeddedWin inherits from clsGWin. Embedded windows provide default functionality for embedding windows, move/copy, selection ownership and input target interaction.

Other Important Files

ewnew.h contains the API definition for creating embeddedWins. Of particular interest there are definitions for:

- embedded window style (EMBEDDED_WIN_STYLE)
- embedded window metrics (EMBEDDED_WIN_METRICS)
- new structs (EMBEDDED_WIN_NEW_ONLY, EMBEDDED_WIN_NEW)
- selection types (ewSelect...)
- move/copy modes

Road Map

Typical subclasses self send:

- msgEmbeddedWinBeginMove
- msgEmbeddedWinBeginCopy
- several messages defined in xfer.h and sel.h

Typical subclasses handle:

- several messages defined in xfer.h and sel.h

Subclasses that support traversal (see mark.h) probably handle:

- msgEmbeddedWinShowChild

Subclasses that manage child windows as part of their data (e.g. text editors) probably handle:

- msgEmbeddedWinInsertChild
- msgEmbeddedWinExtractChild
- msgEmbeddedWinPositionChild

Subclasses that file information other than instance data (e.g. reference buttons) probably handle:

- msgEmbeddedWinDestroy
- msgEmbeddedWinGetDest
- msgEmbeddedWinForwardedGetDest
Subclasses that implement sophisticated printing behavior probably handle:

- msgEmbeddedWinGetPrintInfo

### Embedding

When an `embeddedWin` has `style.embeddor` true, it can embed all `embeddedWins` with `style.embeddee` true. It can also have `embeddedWins` moved or copied into it. Examples of embeddors are (1) cork margins, (2) bookshelves, and (3) the main window of most applications. An `embeddedWin` with `style.embeddor` true also responds to the "link" gesture (`xgsDblCircle` in `xgesture.h`) by creating a goto button in the window.

When an `embeddedWin` has `style.embeddee` true, the `embeddedWin` can be embedded, moved and copied. Examples of embeddees are (1) icons for an application (2) `appWins` around an application's frame (see `appwin.h`) and (3) goto buttons (see `goto.h`).

### Move/Copy Behavior

The header files `sel.h` and `xfer.h` describe PenPoint's move/copy mechanism. You need to understand PenPoint's general move/copy mechanism before you'll be able to understand `embeddedWin`'s specific use of it.

`clsEmbeddedWin` defines a data transfer type, `xferEmbeddedWin`, and a corresponding data transfer protocol. These can be used to move and copy `embeddedWins`.

Unlike most PenPoint data transfer protocols, the `xferEmbeddedWin` protocol is primarily a "push" protocol — the destination sends a message to the source instructing the source to move/copy itself into the destination.

If the source and destination agree to move data using `xferEmbeddedWin`, the following steps are taken. (This discussion assumes that the destination's `style.quickMove` is true; see section "Move Optimizations" for more information.)

- The destination `embeddedWin` sends `msgEmbeddedWinMove` to the source `embeddedWin` to have the source move itself into the destination at `pArgs->xy`.
- In response, the source self sends `msgEmbeddedWinMoveCopyOK`. If the resulting `moveOK` is false, the source returns `stsEWSelRefusedMove`.
- If the destination's parent window is the same as self's parent window, then the source `embeddedWin" moves" itself by sending `msgEmbeddedWinPositionChild` to the destination.
- If self and the destination are in the same process, then the source `embeddedWin" moves" itself by sending `msgEmbeddedWinExtractChild` to its parent, and then sending `msgEmbeddedWinInsertChild` to the destination.
- If self and the destination are in different processes, then the source `embeddedWin" moves" itself by (1) using `msgCopy` to create a copy of itself that is owned by the destination's process, and (2) sending `msgEmbeddedWinInsertChild` to the destination. Finally the original source `embeddedWin` posts `msgEmbeddedWinDestroy` to itself.

Copying data goes through the following steps:

- The destination `embeddedWin` sends `msgEmbeddedWinCopy` to the source `embeddedWin` to have the source copy itself into the destination at `pArgs->xy`. 
In response, the source self sends `msgEmbeddedWinMoveCopyOK`. If the resulting `copyOK` is false, the source returns `stsEWSelRefusedCopy`.

The source `embeddedWin` "copies" itself by (1) using `msgCopy` to create a copy of itself that is owned by the destination's process, and (2) sending `msgEmbeddedWinInsertChild` to the destination.

### Selection Interaction

The header file `sel.h` describes PenPoint's selection mechanism. You need to understand PenPoint's general selection mechanism before you can understand `embeddedWin`'s specific use of it.

`clsEmbeddedWin` provides default selection management for itself and its subclasses.

Some objects should take selection ownership via `msgSelSetOwner` and some should take ownership via `msgSelSetOwnerPreserve`. (See `sel.h` for complete information, but here's one example: objects in pop-up dialog boxes, such as option sheets, should typically take ownership via `msgSelSetOwnerPreserve`.)

Rather than having each subclass or instance compute which way to take the selection, `embeddedWin` creators can give an `embeddedWin` a `style.selection` value which tells the `embeddedWin` which message to use to take selection ownership.

Subclasses of `clsEmbeddedWin` should self send `msgSelSelect` to take selection ownership rather than sending `msgSelSetOwner` or `msgSelSetOwnerPreserve` directly to the `SelectionManager`.

In response to `msgSelSelect`, an `embeddedWin` does the following:

- If `style.selection` is `ewSelect`, the `embeddedWin` sends `msgSelSetOwner` to the `SelectionManager` with `self` as the value of `pArgs`.
- If `style.selection` is `ewSelectPreserve`, the `embeddedWin` sends `msgSelSetOwnerPreserve` to the `SelectionManager` with `self` as the value of `pArgs`.
- If `style.selection` is `ewSelectUnknown` (the default), the `embeddedWin` searches up the window hierarchy looking for the first window that (1) is an `embeddedWin` and (2) has a `style.selection` other than `ewSelectUnknown`. The value of that window's `style.selection` is used. If no ancestor sets this bit, or no ancestor is an `embeddedWin`, the `embeddedWin` takes the selection via `msgSelSetOwner`.

In addition to selection ownership message, an `embeddedWin` provides default responses to several other messages defined in `sel.h`. Details of each response are described with the specific messages later in this file.

### Input Target Interaction

One of PenPoint's UI guidelines is that, in most cases, the selection owner should also be the input target. The input target receives keyboard events from the input system. (See `sel.h` and `input.h` for more information.)

While PenPoint as a whole does not enforce a link between selection ownership and the input target, `clsEmbeddedWin` does. As part of its response to `msgSelSelect` and `msgSelPromote`, an `embeddedWin` makes itself the input target.
Enabling Move/Copy of the Entire Window

If you want an entire embeddedWin to be moveable or copyable as a window, then you should set style.moveable and style.copyable to true. Also, you should turn on the inputHoldTimeout flag of the window’s input flags.

```c
pArgs->win.flags.input |= inputHoldTimeout;
```

Move Optimizations

By default, an embeddedWin’s style.quickMove is true, and the section "Move/Copy Behavior" correctly describes what happens during a move. But a client or subclass can set style.quickMove false, and thereby defeat the "same parent" and "same process" optimizations.

```c
#ifndef EMBEDWIN INCLUDED
#define EMBEDWIN INCLUDED
#endif
#ifndef EMBEDWIN NEW INCLUDED
#include <ewnew.h>
#endif
#ifndef FS_INCLUDED
#include <fs.h>
#endif
#ifndef PRINT_INCLUDED
#include <print.h>
#endif
```

Common #defines and typedefs

typedef OBJECT EMBEDDED_WIN, *P_EMBEDDED_WIN;

Status Codes

```c
#define stsEWNoSelection MakeStatus(clsEmbeddedWin, 1)
#define stsEWSelRefusedMove MakeStatus(clsEmbeddedWin, 2)
#define stsEWSelRefusedCopy MakeStatus(clsEmbeddedWin, 3)
#define stsEWSelRefusedLink MakeStatus(clsEmbeddedWin, 4)
#define stsEWUnrecognizedFormat MakeStatus(clsEmbeddedWin, 5)
#define stsEWMoveToInvalidLocation MakeStatus(clsEmbeddedWin, 6)
#define stsEWCopyToInvalidLocation MakeStatus(clsEmbeddedWin, 7)
#define stsEWNotEmbeddee MakeStatus(clsEmbeddedWin, 8)
#define stsEWRefusedDelete MakeStatus(clsEmbeddedWin, 9)
```

xferEmbeddedWin is the data transfer type clsEmbeddedWin uses to move or copy embeddedWins.

```c
#define xferEmbeddedWin MakeTag(clsEmbeddedWin, 1)
```

Messages

msgEmbeddedWinGetMetrics

Passes back an embeddedWin’s metrics.

Takes P_EMBEDDED_WIN_METRICS, returns STATUS.

```c
#define msgEmbeddedWinGetMetrics MakeMsg(clsEmbeddedWin, 1)
```

Comments

pArgs->uuid is set if and only if style.embeddee is true.

See ewnew.h for the definition of P_EMBEDDED_WIN_METRICS.
msgEmbeddedWinGetStyle
Passes back an embeddedWin's style.
Takes P_EMBEDDED_WIN_STYLE, returns STATUS.
#define msgEmbeddedWinGetStyle MakeMsg(clsEmbeddedWin, 2)

Comments
See ewnew.h for the definition of P_EMBEDDED_WIN_STYLE.

msgEmbeddedWinSetStyle
Specifies an embeddedWin's style.
Takes P_EMBEDDED_WIN_STYLE, returns STATUS.
#define msgEmbeddedWinSetStyle MakeMsg(clsEmbeddedWin, 3)

Comments
If pArgs->embeddee is true and the embeddedWin's uuid is nil, a uuid is created for the window.
Clients must not alter the value of style.moveCopyMode.
See ewnew.h for the definition of P_EMBEDDED_WIN_STYLE.

Move/Copy Protocol Messages

msgEmbeddedWinBeginMove
Places an embeddedWin in move mode.
Takes P_EMBEDDED_WIN_BEGIN_MOVE_COPY, returns STATUS.
#define msgEmbeddedWinBeginMove MakeMsg(clsEmbeddedWin, 4)

Arguments
typedef struct EMBEDDED_WIN_BEGIN_MOVE_COPY {
    XY32 xy;  // x,y in source to begin move/copy.
    RECT32 bounds;  // Bounding box of area to move/copy.
    U32 reserved[4];  // Reserved.
} EMBEDDED_WIN_BEGIN_MOVE_COPY, *P_EMBEDDED_WIN_BEGIN_MOVE_COPY;

Comments
An embeddedWin self sends this message to get itself into move mode. This message is usually self sent by an embeddedWin as part of the response to msgSelBeginMove if style.moveable is set.
clsEmbeddedWin responds by creating a move icon (an instance of clsMoveCopyIcon). If pArgs->bounds is a visible rectangle, the move icon is created with an image of what's displayed in the pArgs->bounds rectangle in the embeddedWin. Otherwise a default move icon is displayed centered at pArgs->xy. The client of the icon is self. Also style.moveCopyMode becomes ewMoveMode.

Return Value
stsRequestDenied The window is already in either ewMoveMode or ewCopyMode

See Also
msgSelBeginMove

msgEmbeddedWinBeginCopy
Places an embeddedWin in copy mode.
Takes P_EMBEDDED_WIN_BEGIN_MOVE_COPY, returns STATUS.
#define msgEmbeddedWinBeginCopy MakeMsg(clsEmbeddedWin, 5)

Message
typedef struct EMBEDDED_WIN_BEGIN_MOVE_COPY {
    XY32 xy;  // x,y in source to begin move/copy.
    RECT32 bounds;  // Bounding box of area to move/copy.
    U32 reserved[4];  // Reserved.
} EMBEDDED_WIN_BEGIN_MOVE_COPY, *P_EMBEDDED_WIN_BEGIN_MOVE_COPY;
An embeddedWin self sends this message to get itself into copy mode. This message is usually self sent by an embeddedWin as part of the response to msgSelBeginCopy if style.copyable is set.

clsEmbeddedWin responds by creating a copy icon (an instance of clsMoveCopyIcon). If pArgs->bounds is a visible rectangle, the copy icon is created with an image of what's displayed in the pArgs->bounds rectangle in the embeddedWin. Otherwise a default copy icon is displayed centered at pArgs->xy. The client of the icon is self. Also style.moveCopyMode becomes ewCopyMode.

stsRequestDenied The window is already in either ewMoveMode or ewCopyMode.

See Also
msgSelBeginCopy

**msgEmbeddedWinMove**
Moves an embeddedWin to the destination.

Takes P_EMBEDDED_WIN_MOVE_COPY, returns STATUS.

```c
#define msgEmbeddedWinMove MakeMsg(clsEmbeddedWin, 6)
```

**Arguments**

typedef struct EMBEDDED_WIN_MOVE_COPY {
  XY32 xy;          // x,y location in dest.
  OBJECT dest;     // Destination object.
  TAG format;      // Data transfer format. Must be xferEmbeddedWin.
  OBJECT uid;      // out: moved/copied object.
  U32 reserved[2]; // Reserved.
} EMBEDDED_WIN_MOVE_COPY, *P_EMBEDDED_WIN_MOVE_COPY;

**Comments**

A destination embeddedWin sends this message to a source embeddedWin to have the source embeddedWin move itself to the destination.

See the section "Move/Copy Behavior" for more information.

**Return Value**

stsEWSelRefusedMove The send of msgEmbeddedWinMoveCopyOK returned FALSE for moveOK.

stsEWMoveToInvalidDestination window could not be moved to pArgs->dest.

**msgEmbeddedWinProvideIcon**

Asks an embeddedWin to provide the move/copy icon.

Takes P_EMBEDDED_WIN_PROVIDE_ICON, returns STATUS.

```c
#define msgEmbeddedWinProvideIcon MakeMsg(clsEmbeddedWin, 23)
```

**Arguments**

typedef struct EMBEDDED_WIN_PROVIDE_ICON {
  MESSAGE msg;      // msgEmbeddedWinMove or msgEmbeddedWinCopy.
  XY32 xy;          // x,y in source to begin move/copy.
  RECT32 bounds;   // Bounding box of area to move/copy.
  OBJECT icon;     // out: the icon.
  U32 reserved[4]; // Reserved.
} EMBEDDED_WIN_PROVIDE_ICON, *P_EMBEDDED_WIN_PROVIDE_ICON;

**Comments**

An embeddedWin's default response is as follows:

- if pArgs->bounds.size.w and pArgs->bounds.size.h are both greater than zero, then a marquee style icon is created using a "snapshot" of the screen image contained in pArgs->bounds.
- Otherwise, a default move or copy icon is created.
msgEmbeddedWinCopy
Copies an embeddedWin to the destination.
Takes P_EMBEDDED_WIN_MOVE_COPY, returns STATUS.

#define msgEmbeddedWinCopy MakeMsg(clsEmbeddedWin, 7)
typedef struct EMBEDDED_WIN_MOVE_COPY {
    XY32 xy;          // x,y location in dest.
    OBJECT dest;      // Destination object.
    TAG format;       // Data transfer format. Must be xferEmbeddedWin.
    OBJECT uid;       // out: moved/copied object.
    U32 reserved[2];  // Reserved.
} EMBEDDED_WIN_MOVE_COPY, *P_EMBEDDED_WIN_MOVE_COPY;

A destination embeddedWin sends this message to a source embeddedWin to have the source
embeddedWin copy itself to the destination.
See the section "Move/Copy Behavior" for more information.

Return Value
stsEWSelRefusedCopy The send of msgEmbeddedWinMoveCopyOK returned FALSE for copyOK.
stsEWCopyToInvalidLocation window could not be copied to pArgs->dest.

msgEmbeddedWinMoveCopyOK
Asks whether it is OK to move or copy an embeddedWin to a destination.
Takes P_EMBEDDED_WIN_MOVE_COPY_OK, returns STATUS.

#define msgEmbeddedWinMoveCopyOK MakeMsg(clsEmbeddedWin, 8)
typedef struct EMBEDDED_WIN_MOVE_COPY_OK {
    BOOLEAN moveOK;  // out: true if ok to move.
    BOOLEAN copyOK;  // out: true if ok to copy.
    EMBEDDED_WIN_MOVE_COPY target; // move/copy struct.
} EMBEDDED_WIN_MOVE_COPY_OK, *P_EMBEDDED_WIN_MOVE_COPY_OK;

A source embeddedWin self sends this message to check that it is OK to move or copy itself to the
destination. The default response to this message is to fill in pArgs->moveOK with style.moveable and
pArgs->copyOK with style.copyable.
See the section "Move/Copy Behavior" for more information.

Return Value
stsEWUnrecognizedFormat target.format was not xferEmbeddedWin.
stsEWNotEmbeddee embeddedWin is not an embeddee.
See Also
msgEmbeddedWinMove

msgEmbeddedWinGetPenOffset
Passes back the pen offset during move or copy.
Takes P_XY32, returns STATUS.

#define msgEmbeddedWinGetPenOffset MakeMsg(clsEmbeddedWin, 9)
Comments
This message allows the destination of a move or copy to determine the actual pen position relative to
the lower-left hand corner of the move/copy icon.
When the user lifts the pen, msgSelBeginMove passes the x,y position of the icon, not the pen.
msgEmbeddedWinGetDest

Get the destination for embeddedWin move or copy.

Takes P_EMBEDDED_WIN_GET_DEST, returns STATUS.

```c
#define msgEmbeddedWinGetDest MakeMsg(clsEmbeddedWin, 10)
#define ewPropCopyDest MakeTag(clsEmbeddedWin, 1) // Private.
```

**Arguments**

```c
typedef struct EMBEDDED_WIN_GET_DEST {
  XY32 xy;          // x,y location in self.
  FS_LOCATOR locator;  // out: Destination parent app.
  U16 sequence;      // out: Sequence in parent.
  char path[fsPathBufLength]; // Path buffer for locator.
  OBJECT source;    // Object to be moved/copied.
  U32 reserved[3];  // Reserved.
} EMBEDDED_WIN_GET_DEST, *P_EMBEDDED_WIN_GET_DEST;
```

**Comments**

Some source embeddedWins move or copy more than themselves in response to msgEmbeddedWinMove or msgEmbeddedWinCopy. Some also transfer filed information. (For instance, reference buttons have to move filed information about the destination of the button.) The source sends msgEmbeddedWinGetDest to the destination to get the file system location that the destination wants the source to use for this filed information.

An embeddedWin's default response is to (1) set pArgs->locator to OSThisAppO's locator, (2) set pArgs->sequence to 1, and (3) set pArgs->path to the empty string. Then if style.embedForward is true, msgEmbeddedWinForwardedGetDest is sent to self's parent window.

Corkboard Windows (clsCorkBoardWin; see cbwin.h) are an example of a class that that has a non-default response to this message. When an embeddedWin is copied to a cork margin, it may represent a document, and the source is likely to copy not only the window but also the document files to the destination. The cork margin cannot allow the source to copy these files into the directory of the cork margin's containing application since then the files would look like they're in the parent application -- the wrong place! So in response to msgEmbeddedWinGetDest, a corkboard window appends an extra directory level to is ancestor's response to msgEmbeddedWinGetDest.

msgEmbeddedWinForwardedGetDest

Get the destination for embeddedWin move or copy.

Takes P_EMBEDDED_WIN_GET_DEST, returns STATUS.

```c
#define msgEmbeddedWinForwardedGetDest MakeMsg(clsEmbeddedWin, 22)
```

**Arguments**

```c
typedef struct EMBEDDED_WIN_GET_DEST {
  XY32 xy;          // x,y location in self.
  FS_LOCATOR locator;  // out: Destination parent app.
  U16 sequence;      // out: Sequence in parent.
  char path[fsPathBufLength]; // Path buffer for locator.
  OBJECT source;    // Object to be moved/copied.
  U32 reserved[3];  // Reserved.
} EMBEDDED_WIN_GET_DEST, *P_EMBEDDED_WIN_GET_DEST;
```

**Comments**

If a child embeddedWin's style.embedForward is true, then the child sends msgEmbeddedWinForwardedGetDest to the parent to allow the parent to override all or part of the child's response to msgEmbeddedWinGetDest.

An embeddedWin's default response to this message is identical to the default response to msgEmbeddedWinGetDest.
msgEmbeddedWinInsertChild
Asks an embeddedWin to insert a child window.
Takes P_EMBEDDED_WIN_INSERT_CHILD, returns STATUS.

#define msgEmbeddedWinInsertChild MakeMsg(clsEmbeddedWin, 11)

typedef struct EMBEDDED_WIN_INSERT_CHILD {
  XY32 xy; // x,y location in destination.
  OBJECT win; // Window to insert/extract/position.
  OBJECT source; // Requestor.
  U32 reserved[4]; // Reserved.
} EMBEDDED_WIN_INSERT_CHILD, *P_EMBEDDED_WIN_INSERT_CHILD,
EMBEDDED_WIN_EXTRACT_CHILD, *P_EMBEDDED_WIN_EXTRACT_CHILD,
EMBEDDED_WIN_POSITION_CHILD, *P_EMBEDDED_WIN_POSITION_CHILD;

c1sEmbeddedWin's default response is as follows; this is illustrated in the sample code below.
• send msgEmbeddedWinGetPenOffset to pArgs->source
• offset pArgs->xy by the value passed back by msgEmbeddedWinGetPenOffset
• send msgWinInsert to pArgs->win with self as the parent.

XY32 xy;
WIN_METRICS wm;
ObjSendUpdateRet(msgEmbeddedWinGetPenOffset, pArgs->source, &xy, SizeOf(xy));
ObjSendUpdateRet(msgWinGetMetrics, pArgs->win, &wm, SizeOf(wm), s);
wm.bounds.origin.x = pArgs->xy.x - xy.x;
wm.bounds.origin.y = pArgs->xy.y - xy.y;
ObjSendRet(msgWinDelta, pArgs->win, &wm, SizeOf(wm), s);
wm.options = wsPOSTOP;
wm.parent = self;
ObjSendRet(msgWinInsert, pArgs->win, &wm, SizeOf(wm), s);

This message may be sent during a move/copy operation; see the section "Move/Copy Behavior" for more information.

msgEmbeddedWinExtractChild
Asks an embeddedWin to extract a child window.
Takes P_EMBEDDED_WIN_EXTRACT_CHILD, returns STATUS.

#define msgEmbeddedWinExtractChild MakeMsg(clsEmbeddedWin, 12)
c1sEmbeddedWin's default response is to ObjectSend msgWinExtract to pArgs->win.
This message may be sent during a move/copy operation; see the section "Move/Copy Behavior" for more information.

msgEmbeddedWinPositionChild
Asks an embeddedWin to reposition a child window.
Takes P_EMBEDDED_WIN_POSITION_CHILD, returns STATUS.

#define msgEmbeddedWinPositionChild MakeMsg(clsEmbeddedWin, 13)
c1sEmbeddedWin's default response is as follows; this is illustrated in the sample code below.
• send msgEmbeddedWinGetPenOffset to pArgs->source
• offset pArgs->xy by the value passed back by msgEmbeddedWinGetPenOffset
• self send msgWinDelta.
XY32 xy;
WIN_METRICS wm;
ObjSendUpdateRet(msgEmbeddedWinGetPenOffset, pArgs->source, &xy, 
SizeOf(xy), s);
ObjSendUpdateRet(msgWinGetMetrics, pArgs->win, &wm, 
SizeOf(wm), s);
wm.bounds.origin.x = pArgs->xy.x - xy.x;
wm.bounds.origin.y = pArgs->xy.y - xy.y;
ObjSendRet(msgWinDelta, pArgs->win, &wm, 
SizeOf(wm), s);

This message may be sent during a move/copy operation; see the section "Move/Copy Behavior" for more information.

### Linking Related Messages

#### msgEmbeddedWinShowChild
Display a given area of an embeddedWin to the user

Takes P_EMBEDDED_WIN_SHOW_CHILD, returns STATUS.

```c
#define msgEmbeddedWinShowChild MakeMsg(clsEmbeddedWin, 14)
```

**Arguments**

```c
typedef struct EMBEDDED_WIN_SHOW_CHILD {
    WIN child;          // the child directly below
    UUID childUUID;     // its UUID
    RECT32 area;        // area to show
    WIN areaWin;        // window that the area is relative to
} EMBEDDED_WIN_SHOW_CHILD, *P_EMBEDDED_WIN_SHOW_CHILD;
```

**Comments**

Clients send this message to ask an embeddedWin to show the rectangle pArgs->area to the user, scrolling if necessary.

Note that pArgs->area is relative to pArgs->areaWin. Therefore handling this message may involve transforming pArgs->area to be relative to self. This can be accomplished as follows:

```c
WIN_METRICS wm;
wmsg.bounds = pArgs->area;
wmsg.parent = self;
ObjCallJmp(msgWinTransformBounds, pArgs->areaWin, &wm, s, Error);
```

In many cases, subclasses need do nothing; clsScrollWin's response to this message takes care of it all.

However, if a subclass does its own scrolling, manages embeddees (for example, by not having them inserted when off-screen) or uses something other than window coordinates to scroll a scroll window, then it needs to respond to this message in the following manner:

- ensure that child is inserted and deltad to the correct place (possibly scrolling it into view if needed)
- transform the rect to the child (remember: it may be in some nested window)
- scroll as needed to get that rect into view.
- call ancestor.

clsEmbeddedWin's default response is to set pArgs->child to self, set pArgs->childUUID to self's UUID and ObjectSend the message to its parent.
Other Messages

**msgEmbeddedWinSetUUID**
Specifies an embeddedWin's uuid.
Takes P_UUID, returns STATUS.

```c
define msgEmbeddedWinSetUUID MakeMsg(clsEmbeddedWin, 19)
```

Comments:
Gives an embeddedWin a UUID, if style.embeddee is true.

**msgEmbeddedWinDestroy**
Permanently destroys an embeddedWin.
Takes OBJ_KEY, returns STATUS.

```c
define msgEmbeddedWinDestroy MakeMsg(clsEmbeddedWin, 20)
```

Comments:
This message is sent to an embeddedWin in response to msgSelDelete, or as the last step of msgEmbeddedWinMove. This message is different from msgDestroy in that this message is sent when the embeddedWin is being permanently destroyed and will never be restored. (msgDestroy is sent when the embeddedWin is being destroyed but may be restored later.) Any subclasses that file data to maintain information as part of their embedding behavior should free that data in response to this message. They should not free that data in response to msgDestroy.

clsEmbeddedWin's default response is as follows:
- if style.deleteable is false, return stsEWRefusedDelete.
- Send msgEmbeddedWinDestroy to any child embeddedWins that are in the same task.
- Self send msgDestroy.

**msgEmbeddedWinGetPrintInfo**
Passes back an embeddedWin's print information.
Takes P_EMBEDDEE_PRINT_INFO, returns STATUS.

```c
define msgEmbeddedWinGetPrintlnfo MakeMsg(clsEmbeddedWin, 21)
```

Comments:
This message gives subclasses an opportunity to support more advanced printing of embeddedWins.
clsEmbeddedWin's default response is to set all fields in *pArgs to 0.

Messages Defined in clsmgr.h

**msgFree**
Defined in clsmgr.h.
Takes OBJ_KEY, returns STATUS.

**msgSave**
Defined in clsmgr.h.
Takes P_OBJ_SAVE, returns STATUS.

Comments:
clsEmbeddedWin saves the embeddedWin's style and UUID.
**msgRestore**
Defined in clsmgr.h.
Takes P_OBJ_RESTORE, returns STATUS.
Comments clsEmbeddedWin restores the embeddedWin's style and UUID.

**Messages Defined in xfer.h and sel.h**

**msgXferList**
Defined in xfer.h.
Takes OBJECT, returns STATUS.
Comments This message is sent to an object to ask it to provide the list of data transfer types it can provide.
clsEmbeddedWin's default response is to add the transfer type xferEmbeddedWin to the end of the list.

**msgSelMoveSelection**
Defined in sel.h.
Takes P_XY32, returns STATUS.
Comments This message is sent to an object to ask it to move the selection to itself.
See the section "Move/Copy Behavior" for more information.

Return Value
stsRequestForward  embeddedWin is not an embedder.
stsEWSelRefusedMove destination embeddedWin refused the move.
stsEWNoSelection No selection exists in the system.

**msgSelCopySelection**
Defined in sel.h.
Takes P_XY32, returns STATUS.
Comments This message is sent to an object to ask it to copy the selection to itself.
See the section "Move/Copy Behavior" for more information.

Return Value
stsRequestForward  embeddedWin is not an embedder.
stsEWSelRefusedCopy destination embeddedWin refused the copy.
stsEWNoSelection No selection exists in the system.

**msgSelRememberSelection**
Defined in sel.h.
Takes P_XY32, returns STATUS.
Comments Self sent by an embeddedWin in response to the Circle-Tap gesture.
clsEmbeddedWin's default response is to
• create a reference button
• insert the button by self sending msgEmbeddedWinInsertChild.
Return Value
stsRequestForward The window is not an embeddor.
stsEWNoSelection No selection exists.

msgSelSelect
Defined in sel.h.
Takes nothing, returns STATUS.

Comments
See the section "Selection Interaction" for a description of an embeddedWin's response to msgSelSelect.

msgSelPromote
Defined in sel.h.
Takes nothing, returns STATUS.

Comments
clsEmbeddedWin's default response is to become the input target by calling InputSetTarget (see input.h) with self as the target.

msgSelYield
Defined in sel.h.
Takes BOOLEAN, returns STATUS.

Comments
clsEmbeddedWin's default response is to return stsOK.

msgSelIsSelected
Defined in sel.h.
Takes nothing, returns BOOLEAN.

Return Value
stsOK Self is the selection owner.
other Self is not the selection owner. (Note that self may be the preserved selection owner.)

msgSelDelete
Defined in sel.h.
Takes U32, returns STATUS.

Comments
See sel.h for a complete description of when this message is sent. Typically, an embeddedWin receives this message because the destination of the move is deleting the source.

embeddedWin's default response is to self send msgEmbeddedWinDestroy.

msgSelBeginMove
Defined in sel.h.
Takes nothing, returns STATUS.

Comments
See sel.h for a complete description of when this message is sent.

clsEmbeddedWin's default response is to self send msgEmbeddedWinMove.

Return Value
stsRequestDenied The embeddedWin is already in move or copy mode.
msgSelBeginCopy
Defined in sel.h.
Takes nothing, returns STATUS.
Comments
See sel.h for a complete description of when this message is sent.
clsEmbeddedWin's default response is to self send msgEmbeddedWinCopy.

Return Value
stsRequestDenied the embeddedWin is already in move or copy mode.

Other Messages

msgIconProvideBitmap
Defined in icon.h.
Takes P_ICON_PROVIDE_BITMAP, returns STATUS.
Comments
An embeddedWin receives this message from a move/copy icon (since the embeddedWin is the icon's client.)
clsEmbeddedWin's default response is to forward the message to OSThisApp().

msgMoveCopyIconDone
Defined in mcicon.h.
Takes OBJECT, returns STATUS.
Comments
An embeddedWin receives this message when a move/copy icon completes. (The move/copy icon completes when it dropped on some destination window.)
clsEmbeddedWin's default response is to send msgSelMoveSelection or msgSelCopySelection (as appropriate) to the destination window.

msgMoveCopyIconCancel
Defined in mcicon.h.
Takes OBJECT, returns STATUS.
Comments
An embeddedWin receives this message when a move/copy icon is canceled. clsEmbeddedWin's default response is to take itself out of move/copy mode (by setting self's style.moveCopyMode to ewMoveCopyModeOff).

msgTrackProvideMetrics
Defined in track.h.
Takes P_TRACK_METRICS, returns STATUS.
Comments
An embeddedWin receives this message from the move/copy icon's tracker. (The tracker can be recognized as the move/copy icon's tracker because pArgs->tag will be tagMoveCopyIconTrack.)
Subclasses can handle this message by repositioning the tracker (and therefore the move/copy icon) relative to the pen. This is done by modifying pArgs->initRect. Typically you do not call the ancestor in such cases. For instance, PenPoint's text component "jumps" the icon so that the pen is at the vertical center of the left edge of the icon by using code similar to the following:
MsgHandlerArgType(SomeViewTrackProvideMetrics, P_TRACK_METRICS)
{
    if (pArgs->tag == tagMoveCopyIconTrack) {
        pArgs->initRect.origin = pArgs->origXY;
        pArgs->initRect.origin.y -= pArgs->initRect.size.h/2;
        return stsOK;
    } else {
        return ObjectCallAncestorCtx(ctx);
    }
    ...
}
EWNEW.H

This file contains the API definition for creating embedded windows.
See embedwin.h for information. Essentially all of the documentation is in embedwin.h.

```c
#ifndef EWNEW_INCLUDED
#define EWNEW_INCLUDED
#endif

#ifndef GWIN_INCLUDED
#include <gwin.h>
#endif

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

Common #defines and typedefs

**EmbeddedWin Selection Types**

Use one of these values in an `embeddedWin`'s `style.selection`.
See the section "Selection Interaction" in embedwin.h for a description of each of these values.

```c
#define ewSelectUnknown 0 // take selection based on parent
#define ewSelect 1 // take selection via `msgSelSetOwner`
#define ewSelectPreserve 2 // take selection via `msgSelSetOwnerPreserve`
```

**Move/Copy Modes**

These are the possible values for `style.moveCopyMode`. The mode is set while an `embeddedWin` is involved in a move/copy. Clients and subclasses must NOT alter the value of this field.

```c
#define ewMoveCopyModeOff 0
#define ewMoveMode 1
#define ewCopyMode 2
```

**Embedded Window Style**

```c
typedef struct EMBEDDED_WIN_STYLE {
  U16 embedder : 1; // Allow embedding. Causes response to 
    // `msgGWinGesture`, `msgEmbeddedWinMove`, 
    // and `msgEmbeddedWinCopy`. See section 
    // "Embedding."

  U16 embeddee : 1; // Can be embedded. Causes `embeddedWin` to 
    // generate UUID. See section "Embedding."

  U16 selection : 2; // Selection style. Most clients use 
    // `ewSelectUnknown`. See section "Selection 
    // Interaction."

  U16 moveable : 1; // `embeddedWin` is moveable. Responds 
    // to `msgSelBeginMove` by self sending 
    // `msgEmbeddedWinBeginMove`.

  U16 copyable : 1; // `embeddedWin` is copyable. Responds 
    // to `msgSelBeginCopy` by self sending
```
// msgEmbeddedWinBeginCopy.
U16 moveCopyMode : 2; // Current move/copy mode. Clients must not
                   // set this field.
U16 deletable     : 1; // Destroy in response to msgEWDestroy?
U16 moveCopyContainer : 1; // Private
U16 embedForward  : 1; // See comments with msgEmbeddedWinGetDest
                       // and msgEmbeddedWinForwardedGetDest.
U16 quickMove     : 1; // Use optimizations when moving windows
                       // within a common parent or within a common
                       // process. True by default. See section
                       // "Move Optimizations."
                       // msgEmbeddedWinExtract/InsertChild.
U16 reserved      : 4; // Reserved for future use.
U16 reserved2     : 16; // Reserved for future use.
} EMBEDDED_WIN_STYLE, *P_EMBEDDED_WIN_STYLE;

Embedded Window Metrics

Passed back from msgEmbeddedWinGetMetrics.

typedef struct EMBEDDED_WIN_METRICS {
    UUID uuid;            // Defined if style.embeddee is true.
    EMBEDDED_WIN_STYLE style;
} EMBEDDED_WIN_METRICS, *P_EMBEDDED_WIN_METRICS;

msgNew

Creates a new embeddedWin object.

Takes P_EMBEDDED_WIN_NEW, returns STATUS. Category: class message.

Arguments
typedef struct EMBEDDED_WIN_NEW_ONLY {
    UUID uuid;
    EMBEDDED_WIN_STYLE style;
    U32 reserved[4];
} EMBEDDED_WIN_NEW_ONLY, *P_EMBEDDED_WIN_NEW_ONLY;
#define embeddedWinNewFields
    gWinNewFields
    EmbeddedWinNewFields
    EmbeddedWinNewFields
} EMBEDDED_WIN_NEW, *P_EMBEDDED_WIN_NEW;

Comments
If style.embeddor is true, objCapCreate is set in object.cap. If the passed in uuid is nil, and the object is
an embeddee, a uuid is created.

msgNewDefaults

Initializes the EMBEDDED_WIN_NEW structure to default values.

Takes P_EMBEDDED_WIN_NEW, returns STATUS. Category: class message.

Message
typedef struct EMBEDDED_WIN_NEW {
    EmbeddedWinNewFields
} EMBEDDED_WIN_NEW, *P_EMBEDDED_WIN_NEW;

Arguments

Comments
Zeros out pNew->embeddedWin and then executes the following:

    win.flags.style |= wsSendFile
    MakeNilUUID(pArgs->embeddedWin.uuid);
    pArgs->embeddedWin.style.deletable = true;
    pArgs->embeddedWin.style.quickMove = true;

This file contains the API definition for clsGotoButton.

clsGotoButton inherits from clsButton.

Provides links to other documents.

A Goto Button is a Button associated with a Mark object. When the Goto Button is tapped, the data pointed to by the Mark are brought into view. Note that Goto Buttons are called Reference Buttons in the PenPoint User Interface.

```c
ifndef GOTO_INCLUDED
#define GOTO_INCLUDED
ifndef BUTTON_INCLUDED
#include <button.h>
#endif
ifndef MARK_INCLUDED
#include <mark.h>
#endif
typedef OBJECT GOTO_BUTTON, *P_GOTO_BUTTON;
define qhGotoButton MakeTag(clsGotoButton, 1)
```

**msgNew**

Creates a new Goto Button object.

Takes P_GOTO_BUTTON_NEW, returns STATUS. Category: class message.

**Arguments**

typedef struct GOTO_BUTTON_NEWONLY {
    MARK mark; // the mark of the button, or objNull
    MARK markNew; // New structure used to create a mark
    U32 reserved[2]; // Reserved.
} GOTO_BUTTON_NEWONLY, *P_GOTO_BUTTON_NEWONLY;

define gotoButtonNewFields \
    buttonNewFields \
    GOTO_BUTTON_NEWONLY gotoButton;

typedef struct GOTO_BUTTON_NEW {
    gotoButtonNewFields \
    GOTO_BUTTON_NEWONLY gotoButton;
} GOTO_BUTTON_NEW, *P_GOTO_BUTTON_NEW;

**Comments**

You can pass in the exact mark object that you want the Goto Button to use, or simply set up the markNew structure and let the Goto Button create its own mark.

**msgNewDefaults**

Initializes a GOTO_BUTTON_NEW structure.

Takes P_GOTO_BUTTON_NEW, returns STATUS. Category: class message.

**Message**

typedef struct GOTO_BUTTON_NEW {
    gotoButtonNewFields \
} GOTO_BUTTON_NEW, *P_GOTO_BUTTON_NEW;
clsGoto sets up the structure so that the Goto Button will create its own mark for the selection. The mark will be document relative because of the setting of the markForSelection and markDocRelative flags in the markNew structure.

Zeroes out `pArgs->gotoButton` and sets

```c
pArgs->win.flags.input = inputHoldTimeout;
pArgs->gWin.helpld = qhGotoButton;
pArgs->embeddedWin.style.embedded = true;
pArgs->embeddedWin.style.moveable = true;
pArgs->embeddedWin.style.copyable = true;
pArgs->embeddedWin.style.selection = ewSelect;
pArgs->label.scale = lsScaleMedium;
```

ObjectCall(msgNewDefaults, clsMark, &(pArgs->gotoButton.markNew));

```c
pArgs->gotoButton.markNew.mark.flags = markForSelection
| markDocRelative
| markRelaxActivate;
```

### msgGotoButtonGetMark

Passes back the mark object being used by a Goto Button.

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

```c
#define msgGotoButtonGetMark MakeMsg(clsGotoButton, 1)
```

### msgGotoButtonGotoLink

Jumps to the mark being used by a Goto Button.

Takes `BOOLEAN`, returns `STATUS`.

```c
#define msgGotoButtonGotoLink MakeMsg(clsGotoButton, 3)
```

If `pArgs` is true, turn to the document; if `pArgs` is false, float the document.

### msgGotoButtonDeleteLink

Deletes a Goto Button link.

Takes nothing, returns `STATUS`.

```c
#define msgGotoButtonDeleteLink MakeMsg(clsGotoButton, 4)
```

### msgGotoButtonLinkToSelection

Links a Goto Button to the selection.

Takes nothing, returns `STATUS`.

```c
#define msgGotoButtonLinkToSelection MakeMsg(clsGotoButton, 5)
```

### msgGotoButtonEditLabel

Allows the user to edit a Goto Button's label.

Takes nothing, returns `STATUS`.

```c
#define msgGotoButtonEditLabel MakeMsg(clsGotoButton, 7)
```
msgGotoButtonPressed
Sent to observers when a Goto Button has been executed.
Takes OBJECT, returns STATUS.
#define msgGotoButtonPressed MakeMsg(clsGotoButton, 6)
Comments
pArgs is the button that was pressed.

msgGotoButtonResetLabel
Causes a Goto Button to reset its label based on the thing it points to.
Takes nothing, returns STATUS.
#define msgGotoButtonResetLabel MakeMsg(clsGotoButton, 2)
Comments
This message is self sent at object new time, and generally should never be resent as it destroys any editing the user has done.

msgGotoButtonGetLabel
Sent to the component containing the marked selection when the Goto Button's label is reset.
Takes P_GOTO_BUTTON_GET_LABEL, returns STATUS.
#define msgGotoButtonGetLabel MakeMsg(clsGotoButton, 8)
Arguments
typedef struct GOTO_BUTTON_GET_LABEL {
    MARK_MSG_HEADER header;
    U32 buflen;
    P_CHAR pBuf;
} GOTO_BUTTON_GET_LABEL, *P_GOTO_BUTTON_GET_LABEL;
Comments
Components that support Goto Buttons should fill in the buffer (*pBuf) with the label to use. If you don't support this message, then clsGoto will try msgSRGetChars to get the characters pointed to by the mark.

msgGotoButtonRePosition
Sent to the component containing the mark to request possible re-positioning for the Goto Button.
Takes P_MARK_MESSAGE, returns STATUS.
#define msgGotoButtonRePosition MakeMsg(clsGotoButton, 9)
Comments
This message is sent when a component wants the Goto Button to end up pointing at a child and not itself. To do this the component returns stsMarkEnterChild to this message, and then the Goto Button will send msgMarkGetChild to re-position the mark at the child (but not actually enter it). Most clients that support Goto Buttons ignore this message.
This file contains the API definition for clsIconWin.

clsIconWin inherits from clsTkTable.

Icon windows are windows that contain a number of embedded windows.

Icon windows can manage their children icons to give them uniform appearance and layout. Icon windows can also force their icons to open floating, and they can deny the ability of users to set icon options. Examples of icon window subclasses are the Bookshelf, Accessories, and the Cork Margin.

```c
#ifndef ICONWIN_INCLUDED
#define ICONWIN_INCLUDED

#ifndef TKTABLE_INCLUDED
#include <tktable.h>
#endif

Common #defines and typedefs

typedef OBJECT ICON_WIN, *P_ICON_WIN;

Icon Window Style

This structure defines the various icon window styles. The fields are as follows:

- iconType  What appearance to give the icons. The values for this field are defined in appwin.h, and are things like awPictAndTitle (large icon over label), awPictOnly (large icon), awSmallPictAndTitle (small icon to the left of a label), awSmallPictOnly (small icon), and awSmallPictOverTitle (small icon over label).
- propagateIconType  If this field is true, all icons in the icon window are forced to have the same iconType. If the user changes one icon's iconType, all icons in the icon window will change to match.
- allowOpenInPlace  If this field is true, documents will be able to open inside the icon window. If false, documents will always open floating.
- constrainedLayout  If this field is true, the icon window will arrange icons into neat rows and columns. An icon dragged to a new location in the icon window will be "snapped" into place. If this field is false, icons are left where they're dropped.
- showOptions  If this field is true, users may make the check gesture over an icon in the icon window to bring up the icon option sheet. If it is false, the check gesture will be rejected.

typedef struct ICON_WIN_STYLE {
    U16 iconType 4;  // Use AppWin icon types (see appwin.h).
    U16 propagateIconType 1;  // True = all icons in win are same type.
    U16 allowOpenInPlace 1;  // False = always open floating.
    U16 constrainedLayout 1;  // True = line up icons in rows & columns.
    U16 showOptions 1;  // True = allow option sheet display.
    U16 reserved 8;  // Reserved.
} ICON_WIN_STYLE, *P_ICON_WIN_STYLE;
```
**Message**

**msgNew**

Creates a new icon window.

Takes P_ICON_WIN_NEW, returns STATUS. Category: class message.

```c
typedef struct ICON_WIN_NEWONLY {
    ICON_STYLE style;
    U32 reserved[4];
} ICON_WIN_NEWONLY, *P_ICON_WIN_NEWONLY;
define iconWinNewFields \	kTableNewFields \ICON_WIN_NEWONLY iconWin;
typedef struct ICON_WIN_NEW {
    iconWinNewFields
} ICON_WIN_NEW, *P_ICON_WIN_NEW;
```

**msgNewDefaults**

Initializes the ICON_WIN_NEW structure to default values.

Takes P_ICON_WIN_NEW, returns STATUS. Category: class message.

```c
typedef struct ICON_WIN_NEW {
    iconWinNewFields
} ICON_WIN_NEW, *P_ICON_WIN_NEW;
```

**Comments**

Zeroes out pArgs->iconWin and sets

- `pArgs->win.flags.style` &= ~wsShrinkWrapWidth;
- `pArgs->win.flags.style` &= ~wsShrinkWrapHeight;
- `pArgs->win.flags.style` |= wsCaptureGeometry;
- `pArgs->win.flags.style` |= wsClipChildren;
- `pArgs->win.flags.style` |= wsHeightFromWidth;
- `pArgs->win.style.grabDown` = false;
- `pArgs->embeddedWin.style.embeddor` = true;
- `pArgs->tableLayout.style.tblXAlignment` = tlAlignLeft;
- `pArgs->tableLayout.style.tblYAlignment` = tlAlignBottom;
- `pArgs->tableLayout.style.childXAlignment` = tlAlignCenter;
- `pArgs->tableLayout.style.childYAlignment` = tlAlignCenter;
- `pArgs->tableLayout.style.growChildWidth` = false;
- `pArgs->tableLayout.style.growChildHeight` = false;
- `pArgs->tableLayout.style.placement` = tlPlaceRowMajor;
- `pArgs->tableLayout.style.senseOrientation` = false;
- `pArgs->tableLayout.style.reverseX` = false;
- `pArgs->tableLayout.style.reverseY` = true;
- `pArgs->tableLayout.numRows.constraint` = tlInfinite;
- `pArgs->tableLayout.numCols.constraint` = tlMaxFit;
- `pArgs->tableLayout.rowHeight.constraint` = tlGroupMax;
- `pArgs->tableLayout.rowHeight.gap` = 2;
- `pArgs->tableLayout.colWidth.constraint` = tlGroupMax;
- `pArgs->tableLayout.colWidth.gap` = 3;
- `pArgs->iconWin.style.iconType` = awSmallPictAndTitle;
- `pArgs->iconWin.style.propagateIconType` = false;
- `pArgs->iconWin.style.allowOpenInPlace` = false;
- `pArgs->iconWin.style.constrainedLayout` = true;

If the environment variable "ICONWIN.SHOWOPTIONS" is set, `pArgs->iconWin.style.showOptions` is set to true, otherwise false.
msgIconWinGetMetrics
Passes back an icon window's metrics.
Takes P_ICON_WIN_METRICS, returns STATUS.
#define msgIconWinGetMetrics MakeMsg(clsIconWin, 1)

typedef struct ICON_WIN_METRICS {
   ICON_WIN_STYLE style;
   U32 reserved[4];
} ICON_WIN_METRICS, *P_ICON_WIN_METRICS;

Comments
Since the icon window metrics structure currently contains no information other than the icon window style, use msgIconWinGetStyle instead of this message.

msgIconWinGetStyle
Passes back an icon window's style.
Takes P_ICON_WIN_STYLE, returns STATUS.
#define msgIconWinGetStyle MakeMsg(clsIconWin, 2)

typedef struct ICON_WIN_STYLE {
   U16 iconType : 4; // Use AppWin icon types (see appwin.h).
   U16 propagateIconType : 1; // True = all icons in win are same type.
   U16 allowOpenInPlace : 1; // False = always open floating.
   U16 constrainedLayout : 1; // True = line up icons in rows & columns.
   U16 showOptions : 1; // True = allow option sheet display.
   U16 reserved : 8; // Reserved.
} ICON_WIN_STYLE, *P_ICON_WIN_STYLE;

msgIconWinSetStyle
Specifies an icon window's style.
Takes P_ICON_WIN_STYLE, returns STATUS.
#define msgIconWinSetStyle MakeMsg(clsIconWin, 3)

typedef struct ICON_WIN_STYLE {
   U16 iconType : 4; // Use AppWin icon types (see appwin.h).
   U16 propagateIconType : 1; // True = all icons in win are same type.
   U16 allowOpenInPlace : 1; // False = always open floating.
   U16 constrainedLayout : 1; // True = line up icons in rows & columns.
   U16 showOptions : 1; // True = allow option sheet display.
   U16 reserved : 8; // Reserved.
} ICON_WIN_STYLE, *P_ICON_WIN_STYLE;
MARK.H

clsMark

clsMark inherits from clsObject.

clsMark provides a path to interact with data in other, possibly nested, components. It is used by Goto Buttons, Search/Replace, and Spell.

See Also
goto.h, sr.h

Overview

PenPoint allows parts of the system, such as search and replace, spell checking and reference buttons, to operate and refer to data in any document or nested embedded documents. This generic reference to data is done with clsMark. In order for clsMark to work, applications must support the client side of clsMark's protocol and the client side of the various system protocols (which are described elsewhere, specifically in sr.h and goto.h).

What follows describes in more detail how clsMark relates to the rest of PenPoint. You may wish to skip ahead to the Example and/or Quick Start sections and refer back to here later.

In PenPoint, the data that make up a document are held in one or more 'components'. Typically these components are descendants of clsEmbeddedWin. Alternatively, your descendant of clsApp might hold all the data. In either case the individual pieces of data (individual words in a text component, shapes in a drawing component, etc.) are only accessible via the component object. No other object knows how the data is actually stored and the data are not usually accessible as objects outside the component.

There are times, however, when these individual data items need to be manipulated from outside of your application. Goto buttons, for example, allow a user to create a link to such a data item, and later turn back to it. Search & Replace, which is driven by a PenPoint supplied manager, needs to access successive pieces of text in both your application and documents embedded within you.

An instance of clsMark, (from now on, simply 'a mark',) is a reference to a data item in a component. We call this data item the 'target' as it is this data item that a Goto button or Search is really interested in, not the component that contains it. The object that uses the mark is called the 'holder' of the mark. A mark may be persistent or temporary. In the former case, once established a mark will remain valid across document shut-downs and re-boots. In the latter, the mark is valid only so long as the component remains active.

In order to support marks a component must create a mapping between the two U32s, or 'token', that a mark holds and its data items. For example, a data base might use this to hold a record number. Remember that the mark might persist beyond a single operation. Therefore, a text editor would NOT use these U32s to be a character position. This is because if a mark is created for a word, and then text is deleted before the word, the desired action is for the mark to still refer to the word which has now moved in character position. Remember: once a mark has been created for a piece of data, there is no way for the component to update the token it has given for it.
An Example

The process of using a mark is best illustrated by the Search & Replace mechanism in PenPoint:
Search/Replace, Spelling, etc. use the mark mechanism to traverse the contents of applications. All applications that allow themselves to be searched, spelled or printed support the component half of this protocol. Implementors of new functionality similar to Search/Replace, Spell, or Print must implement the driver half of the protocol.

When the user selects Find from the Edit menu, the Search Manager responds by displaying an option sheet and by creating a mark which initially points to the document the user is working in.

As the user requests find and replace operations, the Search Manager calls the mark with msgMarkDeliver with arguments specifying the clsSR messages it wants sent to the component. In turn, the mark sends those messages, along with its own messages to the component and, if requested, each nested component. It is these messages that a client must implement. (The clsMark messages are described in this file, the clsSR messages are in sr.h.)

After the component performs the request find and/or replace, the status is passed back all the way to the Search Manager which lets the user know. Note that the messages to hilite and select the found text are also passed from the Search Manager to the component this way.

Quick Start

How to be a Client that Supports Marks

1) You must decide how to refer to the data items in the component via tokens. There are several considerations: How will you treat marks that survive save & restore? How will the mark be affected by edit operations? What is the ordering of data items (even if the data items have no intrinsic ordering, you will still need a way to enumerate over them in some serial order)? Do you inherit markable data from your ancestor that you don’t take care of.

2) Support the basic messages:
   • msgMarkCreateToken
   • msgMarkDeleteToken (if necessary)
   • msgMarkGetDataAncestor

3) For a component that can be traversed, support the following. These are typically very easy to implement, and all markable components should support them.
   • msgMarkPositionAtEdge
   • msgMarkPositionAtToken
   • msgMarkCompareTokens

4) If the component has a graphical view of the data, support the following. This allow Goto Buttons to work (first three messages) and the Search & Replace and Spell gestures to work (last message).
   • msgMarkShowTarget
   • msgMarkSelectTarget (if it can hold the selection)
   • msgMarkPositionAtSelection (if it can hold the selection)
   • msgMarkPositionAtGesture (if it can target gestures to data)
5) If the component has any text as data, support the following. These support the text side of both Search & Replace and Spell. They are also used by reference buttons. See sr.h for a description of these messages.

- msgSRNextChars
- msgSRGetChars
- msgSRPositionChars
- msgSRReplaceChars (if replacement is possible)

6) If your component manages its own embedees, support:

- msgMarkPositionAtChild
- msgMarkNextChild
- msgMarkGetChild

7) If your component is not a descendant of clsEmbeddedWin or clsApp then it must support the following messages:

- msgMarkGetParent
- msgMarkGetUUIDs

**How to be a Driver that Uses Marks**

1) Send msgNewDefaults and msgNew to clsMark. This creates the initial mark and sets up the component for the mark.

2) Send the appropriate msgMarkPosition... message. This sets the mark at the place where you want it. You are free to define new kinds of positioning messages, so long the components you work with support them. As a back-up, you should always be prepared to deal with stsMsgNotUnderstood as a response from a message sent to a component. In that case, do the default action (try msgMarkPositionAtEdge).

3) If you need to manipulate the mark, send messages via msgMarkDeliver, msgMarkDeliverPos, and/or msgMarkDeliverNext. These will instruct the component to take the appropriate action on the target and the mark. Again, be prepared to deal with stsMsgNotUnderstood. Try to use standardized messages, such as msgSRNextChars, when your specific ones fail. Remember: an embedded document may not know the protocol the enclosing document does and vice versa.

4) You can file and unfile the mark as you would with any other object. The mark will remain connected to the target. Note that once a mark has been filed it is now permanent; this will likely consume resources at the component that has the target.

5) Send msgDestroy to the mark when you are done with it.

```c
#ifndef MARK_INCLUDED
#define MARK_INCLUDED 1
#endif
#ifndef GO_INCLUDED
#include <go.h>
#endif
#ifndef UUID_INCLUDED
#include <uuid.h>
#endif
#ifndef GWIN_INCLUDED
#include <gwin.h>
#endif
```
## Statuses

```c
#define stsMarkNoUUIDs MakeStatus(clsMark, 1)
#define stsMarkRedirectMark MakeStatus(clsMark, 2)
#define stsMarkNoWin MakeStatus(clsMark, 3)
#define stsMarkNoComponent MakeStatus(clsMark, 4)
#define stsMarkComponentsDiffer MakeWarning(clsMark, 10)
#define stsMarkTokensEqual MakeWarning(clsMark, 11)
#define stsMarkTokenAfter MakeWarning(clsMark, 12)
#define stsMarkTokenBefore MakeWarning(clsMark, 13)
#define stsMarkEnterChild MakeWarning(clsMark, 20)
#define stsMarkRetry MakeWarning(clsMark, 21)
#define stsMarkSkipChild MakeWarning(clsMark, 22)
#define stsMarkNotActive MakeWarning(clsMark, 23)
```

### Common #defines and Types

```c
typedef OBJECT MARK, *P_MARK;

typedef struct MARK_TOKEN {
    CLASS classLevel; // which class level is the data at
    U32 index; // index to the data item
    U32 index2; // secondary index if needed
} MARK_TOKEN, *P_MARK_TOKEN;

typedef struct MARK_COMPONENT {
    UUID appUUID;
    UUID compUUID;
    VID compVID;
} MARK_COMPONENT, *P_MARK_COMPONENT;

typedef U32 MARK_FLAGS, *P_MARK_FLAGS;

define markDocRelative flag1 // if saved, document relative
#define markForSelection flag2 // make mark for the selection
#define markAlwaysDelete flag3 // if you manage the destination
#define markRelaxActivate flag4 // don't always activate

typedef U16 MARK_MSG_FLAGS, *P_MARK_MSG_FLAGS;
```
These flags are only valid with `msgMarkDeliverPos` & `msgMarkDeliverNext`

```c
#define markMsgNormal 0 // standard message send
#define markMsgTry 1    // one in a sequence of possible messages
#define markMsgLastTry 2 // last in a sequence of messages
#define markMsgMode 3    // 'x' with flags to extract flag field
```

These flags are only valid with `msgMarkDeliverNext`

```c
#define markBackward flag8   // direction of movement is reversed
#define markEnterNone 0       // enter no children
#define markEnterAll  flag9   // enter all children
#define markEnterOpen flag10  // enter only open children
#define markEnterMode  (flag9|flag10) // 'x' with flags to extract Enter // field
#define markExitUp flag11     // at end, move up to parents
```

// default flag settings:
```c
#define markDefaultMsgFlags 0
#define markDefaultPosMsgFlags markMsgNormal
#define markDefaultNextMsgFlags (markMsgNormal | markEnterOpen | markExitUp)
```

`MARK_MSG_HEADER` must be the start of the argument structure for any message delivered via `msgMarkDeliver`, `msgMarkDeliverPos`, or `msgMarkDeliverNext`. It allows `clsMark` to insert the token information into the message arguments to indicate which part of the component is to be operated on.

```c
typedef struct MARK_MSG_HEADER {
    MARK_TOKEN token;  // Supplied by mark: the token
    MESSAGE msg;       // In: the message to send
    SIZEOF lenArgs;   // In: length of the whole structure
    MARK_MSG_FLAGS flags; // In: flags as appropriate
} MARK_MSG_HEADER, *P_MARK_MSG_HEADER;

typedef struct MARK_MESSAGE {
    MARK_MSG_HEADER header;
} MARK_MESSAGE, *P_MARK_MESSAGE;
typedef U16 MARK_LOCATION;
```

The following location codes are only valid for `msgMarkPositionAtGesture`. These may be or'd together and in with the above codes...

```c
#define markLocWhole 0
#define markLocBeginning 1
#define markLocEnd 2
```

Important: all message handlers for messages sent via `msgMarkDeliver`, `msgMarkDeliverPos`, or `msgMarkDeliverNext`, must have the following as its first statement. Replace "clsYourClassHere" with the uid of your class.

```c
MarkHandlerForClass(clsYourClassHere);
#define MarkHandlerForClass(cls) \
    if (WKNValue(((P_MARK_TOKEN)pArgs)->classLevel) != WKNValue(cls)) \
        return ObjectCallAncestor(msg, self, pArgs, ctx);
```
Messages

msgNew

Creates a new mark, initialized to the given component (if any).

Takes P_MARK_NEW, returns STATUS. Category: class message.

Arguments

typedef struct MARK_NEW_ONLY {
    MARK_FLAGS flags;
    MARK_COMPONENT component;
    U16 reserved[2];
} MARK_NEW_ONLY, * P_MARK_NEW_ONLY;
#define markNewFields \objectNewFields \MARK_NEW_ONLY mark;
typedef struct MARK_NEW {
    markNewFields
} MARK_NEW, * P_MARK_NEW;

The fields you might typically set are pArgs->mark.flags: or in markForSelection to refer to the selection object, or in markDocRelative if you ever plan on saving the mark object pArgs->mark.component.compUID: the object to refer to (not needed if you set markForSelection above)

msgNewDefaults

Initializes the MARK_NEW structure to default values.

Takes P_MARK_NEW, returns STATUS. Category: class message.

Arguments

typedef struct MARK_NEW {
    markNewFields
} MARK_NEW, * P_MARK_NEW;

Comments

Zeroes out pNew->mark. Specifically, this includes:

MakeNilUUID(pArgs->mark.component.appUUID);
MakeNilUUID(pArgs->mark.component.compUUID);
pArgs->mark.component.compUID = objNull;

msgMarkDeliver

Delivers a message to the target that does not move the token.

Takes P_MARK_MESSAGE, returns STATUS.

Arguments

#define msgMarkDeliver MakeMsg(clsMark, 1)
typedef struct MARK_MESSAGE {
    MARK_MSG_HEADER header;
} MARK_MESSAGE, * P_MARK_MESSAGE;

Comments

The message in pArgs->header.msg is sent to the component after the mark fills in the token field. Note that the pArgs for the sent message are the same as the pArgs that are passed in to msgMarkDeliver. Various messages that are sent to components have extra fields tacked on to this structure. Therefore, all messages delivered with msgMarkDeliver MUST have a pArgs structure that starts with same fields as MARK_DELIVER. Furthermore, the lenArgs field must be set to the size of the WHOLE structure.
msgMarkDeliverPos
Delivers a message to the target that moves the token but does not change the component.
Takes P_MARK_MESSAGE, returns STATUS.

#define msgMarkDeliverPos MakeMsg(clsMark, 2)

typedef struct MARK_MESSAGE {
    MARK_MSG_HEADER header;
} MARK_MESSAGE, *P_MARK_MESSAGE;

This is just like msgMarkDeliver, only it is used to deliver a message that will potentially reposition the mark elsewhere in the component. It is chiefly used with the msgMarkPosition... messages.
The additional flags argument is used to determine how the holder wants to interpret the response from the client. Normally you use markMsgNormal, which automatically deals with certain client response codes that the holder doesn’t need to be aware of.

For example, if a holder wants to use msgMarkPositionAtEdge the code would be:

    MARK_POSITION_EDGE edgeArgs;
    edgeArgs.msg = msgMarkPositionAtEdge;
    edgeArgs.lenArgs = SizeOf(MARK_POSITION_EDGE);
    edgeArgs.flags = markMsgNormal;
    edgeArgs.location = markLocBeginning;
    ObjCallRet(msgMarkDeliverPos, aMark, &edgeArgs);

However, if the holder wishes to try a different positioning message if the first one fails, then the holder must use the flag setting markMsgTry on all except the last message which uses markMsgLastTry. Furthermore, these must be in a while loop and repeated if stsMarkRetry is ever returned.

For example, if a holder would like to use the (hypothetical) message msgPositionAtVowel, and if that fails use msgPositionAtLetter, and if that fails try msgPositionAtCharacter; then it the code would be:

    POS_VOWEL posVowel;
    POS_LETTER posLetter;
    POS_CHAR posChar;
    STATUS s;
    while (true) {
        posVowel.msg = msgPositionAtVowel;
        posVowel.lenArgs = SizeOf(POS_VOWEL);
        posVowel.flags = markMsgTry;
        s = ObjectCall(msgMarkDeliverPos, aMark, &posVowel);
        if (s == stsMarkRetry) continue;
        if (s != stsMsgNotUnderstood) break; // some error occurred
        posLetter.msg = msgPositionAtLetter;
        posLetter.lenArgs = SizeOf(POS_LETTER);
        posLetter.flags = markMsgTry;
        s = ObjectCall(msgMarkDeliverPos, aMark, &posLetter);
        if (s == stsMarkRetry) continue;
        if (s != stsMsgNotUnderstood) break; // some error occurred
        posChar.msg = msgPositionAtCharacter;
        posChar.lenArgs = SizeOf(POS_CHAR);
        posChar.flags = markMsgLastTry;
        s = ObjectCall(msgMarkDeliverPos, aMark, &posChar);
        if (s == stsMarkRetry) continue;
        if (s != stsMsgNotUnderstood) break; // some error occurred
        // do what you do if none were understood
    }
While this code is a little complicated, it allows the holder to deal with a variety of components that may know different messages. The while loop and stsMarkRetry are necessary for the handling of inherited component data and behavior. (Specifically, while the mark takes care of most of the chore of moving from level to level in a components class hierarchy, only the holder knows the sequence of messages to try at each level, so the stsMarkRetry acts as a sentinel to the holder to retry the full sequence again.)

**msgMarkDeliverNext**
Delivers a message to the target that moves the token and sometimes (but not always) changes the component.

Takes P_MARK_MESSAGE, returns STATUS.

```c
#define msgMarkDeliverNext MakeMsg(clsMark, 3)
```

**Message Arguments**

```c
typedef struct MARK_MESSAGE {
    MARK_MSG_HEADER header;
} MARK_MESSAGE, *P_MARK_MESSAGE;
```

This is the same as `msgMarkDeliverPos`, only it is used when the repositioning of the token may result moving to a new component. This may happen in messages like `msgSRNextChars` where the next string to search is in an embedded component.

The flags field is used the same way as in `msgMarkDeliverPos`. The flags field also carries some additional flags: These indicate which direction the movement is in, and what to do about embedded components and what to do at the end of a component.

All components that respond to messages sent via `msgMarkDeliverNext` are responsible for two things:

1. They must check the `markBackward` flag to determine the direction of motion.
2. If they encounter a child window as the next item, regardless of what the message is looking for, then the token needs to be set to refer to that child and `stsMarkEnterChild` needs to be returned.

**msgMarkSend**
Sends a message to a component with no further processing.

Takes P_MARK_SEND, returns STATUS.

```c
#define msgMarkSend MakeMsg(clsMark, 9)
```

**Arguments**

```c
typedef struct MARK_SEND {
    MESSAGE msg;     // the message to send
    P_ARGS pArgs;    // pointer to the arguments
    SIZEOF lenArgs;  // length of those arguments
} MARK_SEND, *P_MARK_SEND;
```

Sends a message to the component. Note that this allows you to send any arbitrary message. However, unlike the `msgMarkDeliver` messages, `msgMarkSend` doesn't copy the token value of the mark into the argument structure passed to the component. Hence, no indication of what the target is goes with the message. This is rarely what you want.

The rule is: any message designed to be used with marks should use one of the `msgMarkDeliver` forms. Any message NOT designed to work with marks (and thus has no specific target) should use `msgMarkSend`. 
**msgMarkSetComponent**

Sets the mark to refer to the given component.

Takes P_MARK_COMPONENT, returns STATUS.

#define msgMarkSetComponent MakeMsg(clsMark, 4)

typedef struct MARK_COMPONENT {
    UUID appUUID;
    UUID compUUID;
    UID compUID;
} MARK_COMPONENT, * P_MARK_COMPONENT;

Comments

You set the fields of the MARK_COMPONENT one of three ways (zeroing the unused fields):

- Set pArgs->compUID to refer to a specific component object
- Set pArgs->appUUID to refer to an application object by UUID
- Set pArgs->appUUID and pArgs->compUUID to refer to a component in an application by UUIDs

This will delete the previous mark, if necessary and send a msgMarkCreateToken to the new component.

To make the mark point at nothing, pass it a pointer to an all-zero structure; do NOT pass it a null pointer!

**msgMarkGetComponent**

Returns the UUID of the app the contains the token and the UUID and UID of the component that contain the token.

Takes P_MARK_COMPONENT, returns STATUS.

#define msgMarkGetComponent MakeMsg(clsMark, 5)

typedef struct MARK_COMPONENT {
    UUID appUUID;
    UUID compUUID;
    UID compUID;
} MARK_COMPONENT, * P_MARK_COMPONENT;

Comments

If the app is not open, then pArgs->compUID will be objNull. If the target is in the app object, then pArgs->compUUID will be zeros.

**msgMarkCompareMarks**

Determines if two marks refer to the same component, and, if so, what order their targets are in.

Takes MARK, returns STATUS.

#define msgMarkCompareMarks MakeMsg(clsMark, 6)

Return Value

stsMarkTokensEqual the targets of the marks are the same
stsMarkTokenAfter the target of the receiver is after the argument
stsMarkTokenBefore the target of the receiver is before the argument
**msgMarkCopyMark**

Creates a new mark, identical to this mark.

Takes P_MARK, returns STATUS.

```
#define msgMarkCopyMark MakeMsg(clsMark, 7)
```

**Comments**

Because marks can't easily reverse direction across components, it's sometimes desirable to save the original position. Since the duplicate mark is independent of the original, it doesn't move when the original does.

**msgMarkGotoMark**

Causes a mark to be selected and displayed to the user.

Takes P_MARK_GOTO, returns STATUS.

```
#define msgMarkGotoMark MakeMsg(clsMark, 8)
```

**Arguments**

```
typedef struct MARK_GOTO {
    BOOLEAN noSelect    : 1,  // inhibits the selection of the target
    noDisplay         : 1,  // inhibits the display of the target
    turnTo            : 1,  // if closed, will do a turn to
    bringTo           : 1,  // if closed, will do a bring to
    reserved          : 12;
} MARK_GOTO, *P_MARK_GOTO;
```

**Comments**

By default, the target is selected and scrolled on screen, provided the document is on screen. Optionally, the document can be activated and either turned to or floated on screen.

**Messages Sent to Components**

Important: message handlers for the first three messages (msgMarkCreateToken, msgMarkDeleteToken, and msgMarkCompareTokens) must have the following as its first statement. Replace "clsYourClassHere" with the uid of your class.

```
MarkHandlerForClass(clsYourClassHere);
```

**msgMarkCreateToken**

Instructs a component to create a token for its data items, and start the token pointing at before all data items.

Takes P_MARK_TOKEN, returns STATUS.

```
#define msgMarkCreateToken MakeMsg(clsMark, 40)
```

**Message Arguments**

```
typedef struct MARK_TOKEN {
    CLASS   classLevel;  // which class level is the data at
    U32     index;       // index to the data item
    U32     index2;      // secondary index if needed
} MARK_TOKEN, *P_MARK_TOKEN;
```

**Comments**

You can only forget about the token associated with a mark when a corresponding msgMarkDeleteToken is received, or the target data is deleted. In the later case you must be careful never to generate that token again as there still might be outstanding tokens for it.
msgMarkDeleteToken
Tells a component that the given token will no longer be in use.
Takes P_MARK_TOKEN, returns STATUS.

#define msgMarkDeleteToken MakeMsg(clsMark, 41)

typedef struct MARK_TOKEN {
    CLASS classLevel; // which class level is the data at
    U32 index; // index to the data item
    U32 index2; // secondary index if needed
} MARK_TOKEN, * P_MARK_TOKEN;

See msgMarkCreateToken.

msgMarkCompareTokens
Asks a component to compare the ordering of two tokens.
Takes P_MARK_COMPARE_TOKENS, returns STATUS.

#define msgMarkCompareTokens MakeMsg(clsMark, 42)

typedef struct MARK_COMPARE_TOKENS {
    MARK_TOKEN firstToken;
    MARK_TOKEN secondToken;
} MARK_COMPARE_TOKENS, * P_MARK_COMPARE_TOKENS;

stsMarkTokensEqual the two tokens point to the same place
stsMarkTokenAfter the first token comes after the second
stsMarkTokenBefore the first token comes before the second

msgMarkGetDataAncestor
Asks for the next higher superclass that contains traversable data.
Takes P_CLASS, returns STATUS.

#define msgMarkGetDataAncestor MakeMsg(clsMark, 46)

Asks a component what the next ancestor the argument inherits data from. The component’s response should is based on what the argument is. Assuming the class of the component is clsMyThing:

objNull respond with clsMyThing
clsMyThing respond with the next class clsMyThing gets data from, typically clsEmbeddedWin or objNull (if none).
otherwise call the ancestor

Example

    if (*pArgs == objNull) *pArgs = clsMyThing;
    else if (*pArgs == clsMyThing) *pArgs = clsEmbeddedWin;
    else return stsOK;
    ObjCallAncestorCtxRet(ctx);

If your code doesn’t inherit data then you’ll do the following:

    if (*pArgs == objNull) *pArgs = clsMyThing;
    else *pArgs = objNull;
    return stsOK;
msgMarkGetParent
Asks a component to set the argument to its parent (embedding) component.
Takes P_MARK_COMPONENT, returns STATUS.

```c
#define msgMarkGetParent MakeMsg(clsMark, 43)
```

**Arguments**
```c
typedef struct MARK_COMPONENT {
    UUID appUUID;
    UUID compUUID;
    UID compUID;
} MARK_COMPONENT, * P_MARK_COMPONENT;
```

**Comments**
Either the UID or the UUIDs should be filled in, mark will take care of the rest. If the component is descended from clsEmbeddedWin or clsApp, it already inherits the correct response and implementation is necessary.

msgMarkGetUUIDs
Asks a component to set the argument to its own app and component UUIDs if it can.
Takes P_MARK_COMPONENT, returns STATUS.

```c
#define msgMarkGetUUIDs MakeMsg(clsMark, 45)
```

**Arguments**
```c
typedef struct MARK_COMPONENT {
    UUID appUUID;
    UUID compUUID;
    UID compUID;
} MARK_COMPONENT, * P_MARK_COMPONENT;
```

**Comments**
If it can't it should return stsMarkNoUUIDs. If your component is a descendant of clsApp or clsEmbeddedWin then you inherit the correct implementation.

msgMarkValidateComponent
Asks a component to verify that it is okay to traverse it.
Takes P_MARK_COMPONENT, returns STATUS.

```c
#define msgMarkValidateComponent MakeMsg(clsMark, 44)
```

**Arguments**
```c
typedef struct MARK_COMPONENT {
    UUID appUUID;
    UUID compUUID;
    UID compUID;
} MARK_COMPONENT, * P_MARK_COMPONENT;
```

**Comments**
This message is sent to objects before a mark refers to them. This gives an object a chance to point the mark at a different object as the component. Typically, a driver might create a mark with the selection holder as the component. However, the selection holder might not be the desired component for a mark (the selection could be a data object, but the mark component should be the app object). Mark sends this message to the proposed component. The proposed component then can either not implement the message (or return stsOK) or set the argument to another component object and return stsMarkRedirectMark. In the first case the proposed component becomes the used component, in the second the returned component becomes the new proposed component.
Messages Sent to Components via msgMarkDeliver

Note: As these are defined in clsMark, these messages may be sent to the mark directly without using msgMarkDeliver, msgMarkDeliverPos, or msgMarkDeliverNext (with mode = markMsgNormal) as appropriate. As usual, the mark and token fields will be filled in by the mark, and then the message passed on.

In addition special processing will be done for some of the messages which would NOT be done if the message was sent via standard delivery messages. This processing is noted for those messages under the heading: 'If sent directly to mark'

Important: all message handlers for these messages must have the following as its first statement.

MarkHandlerForClass(clsYourClassHere);

### msgMarkPositionAtEdge
Asks a component to reposition the token to one end or the other of the data.

Takes P_MARK_POSITION_EDGE, returns STATUS.

```c
#define msgMarkPositionAtEdge MakeMsg(clsMark, 80)
```

**Arguments**

```c
typedef struct MARK_POSITION_EDGE {
    MARK_MSG_HEADER header;
    MARK_LOCATION location;  // either markLocBeginning or markLocEnd
} MARK_POSITION_EDGE, *P_MARK_POSITION_EDGE;
```

### msgMarkPositionAtToken
Asks a component to reposition the token to the same position as another token for the same component.

Takes P_MARK_POSITION_TOKEN, returns STATUS.

```c
#define msgMarkPositionAtToken MakeMsg(clsMark, 81)
```

**Arguments**

```c
typedef struct MARK_POSITION_TOKEN {
    MARK_MSG_HEADER header;
    MARK otherMark;  // In; the other mark
    MARK_TOKEN otherToken;  // In: the token to copy
} MARK_POSITION_TOKEN, *P_MARK_POSITION_TOKEN;
```

**Comments**

If sent directly to mark: you only need to fill in the otherMark field, the mark will take care of the rest & will check to see that both marks point at the same component. Since you'd have no idea what the other Token is, this is the only sensible way to send this message (via msgMarkDeliver won't work).

### msgMarkPositionAtChild
Asks a component to reposition the token to the given child component which is given as a UUID/UID pair.

Takes P_MARK_POSITION_CHILD, returns STATUS.

```c
#define msgMarkPositionAtChild MakeMsg(clsMark, 82)
```

**Arguments**

```c
typedef struct MARK_POSITION_CHILD {
    MARK_MSG_HEADER header;
    MARK_COMPONENT child;  // In: the child to position to;
} MARK_POSITION_CHILD, *P_MARK_POSITION_CHILD;
```

**Comments**

The UID may be null if it is unknown, but the UUID will always be valid.
**msgMarkPositionAtGesture**

Asks a component to reposition the token at the given gesture.

Takes `P_MARK_POSITION_GESTURE`, returns `STATUS`.

```c
#define msgMarkPositionAtGesture MakeMsg(clsMark, 83)
```

**Arguments**

typedef struct MARK_POSITION_GESTURE {
    MARK_MSG_HEADER header;
    GWIN GESTURE gesture;
    MARK_LOCATION location;
} MARK_POSITION_GESTURE, *P_MARK_POSITION_GESTURE;

**Comments**

The location parameter indicates how to position relative to the gesture. Note that there are a variety of location codes that might be or'd together.

**msgMarkPositionAtSelection**

Asks a component to reposition the token to the selection, which it presumably owns.

Takes `P_MARK_POSITION_SELECTION`, returns `STATUS`.

```c
#define msgMarkPositionAtSelection MakeMsg(clsMark, 85)
```

**Arguments**

typedef struct MARK_POSITION_SELECTION {
    MARK_MSG_HEADER header;
    MARK_LOCATION location;
} MARK_POSITION_SELECTION, *P_MARK_POSITION_SELECTION;

**Comments**

If the component doesn't own the selection, then return `stsFailed`.

**msgMarkNextChild**

Requests the component to move the token to the next child.

Takes `P_MARK_MESSAGE`, returns `STATUS`.

```c
#define msgMarkNextChild MakeMsg(clsMark, 86)
```

**Message**

typedef struct MARK_MESSAGE {
    MARK_MSG_HEADER header;
} MARK_MESSAGE, *P_MARK_MESSAGE;

**Comments**

If a child is found and the token moved to it, return `stsMarkEnterChild`, not `stsOK`. If return, the mark is likely (but may not) send `msgMarkGetChild` to find out who the child actually is.

**msgMarkGetChild**

Requests the component to fill in the component at the current token.

Takes `P_MARK_GET_CHILD`, returns `STATUS`.

```c
#define msgMarkGetChild MakeMsg(clsMark, 90)
```

**Arguments**

typedef struct MARK_GET_CHILD {
    MARK_MSG_HEADER header;
    MARK_COMPONENT child; //Out: fill in uid or uuids
    BOOLEAN childIsDoc; //Out: is the child a document?
    BOOLEAN childIsOpen; //Out: is the child open?
} MARK_GET_CHILD, *P_MARK_GET_CHILD;

**Comments**

This is sent because, presumable, the response to some other move message was `stsMarkEnterChild`. If the token doesn't point at a child, return `stsFailed`. 
pArgs->childIsDoc should set true if the child is an embedded document. If the child is just an embedded component that is to be considered part of the receiving component, then set this field false. This field is used by clsMark to determine if it should apply the markEnterMode bits that control entering embedded documents (they don't control entering embedded components, this is always done.)

If pArgs->childIsDoc is set true, then childIsOpen must be set to reflect the "open" status of the embedded doc.

If your component is managing its own embedees, typically your component will only deal with the embedded instances of clsAppWin. These are components that are part of your component: you should set pArgs->childIsDoc to false (pArgs->childIsOpen doesn't matter in this case). When the appWin is entered, it will handle the proper reporting of the embedded document. (clsAppWin sets pArgs->childIsDoc to true and pArgs->childIsOpen appropriately.)

**msgMarkSelectTarget**

Requests the component to select the target data item.

Takes P_MARK_MESSAGE, returns STATUS.

```c
#define msgMarkSelectTarget MakeMsg(clsMark, 89)
```

**msgMarkShowTarget**

Request the component to return the window that contains the graphical view of the target.

Takes P_MARK_SHOW_TARGET, returns STATUS.

```c
#define msgMarkShowTarget MakeMsg(clsMark, 88)
```

The rectangle returned is the area within the window that encloses the target.

Some components may not have a viewable representations of the target, in which case they can return stsMarkNoWin, or simply not implement this message. Other components may have a graphical view only part of the time. In this case, it should ensure that the target has a graphical representation, otherwise return stsMarkNoWin if the target isn't right now.

Note that this message requests that the target be scrolled into view. That should be done by sending msgEmbeddedWinShowChild to the win showing the target (usually the win that is returned in pArgs->win).

**Messages Sent Internally**

**msgMarkEnterChild**

Sent when a component requests the mark to enter a child (usually via returning stsMarkEnterChild to a message send with msgMarkDeliverNext).

Takes P_MARK_MESSAGE, returns STATUS.

```c
#define msgMarkEnterChild MakeMsg(clsMark, 120)
```
typedef struct MARK_MESSAGE {
    MARK_MSG_HEADER header;
} MARK_MESSAGE, *P_MARK_MESSAGE;

This message sends msgMarkGetChild to the component to get the child at the token and then enters the child if appropriate.

msgMarkEnterLevel
Sent when a component requests the mark to bump up a level in its class chain, or when a position or next message fails and the mark tries the next class level.

Takes P_MARK_MESSAGE, returns STATUS.

#define msgMarkEnterLevel MakeMsg(clsMark, 121)

typedef struct MARK_MESSAGE {
    MARK_MSG_HEADER header;
} MARK_MESSAGE, *P_MARK_MESSAGE;

This message sends msgMarkGetDataAncestor to the component and resets the token.

msgMarkEnterParent
Sent when a component runs out of data altogether and the mark needs to move on (and up).

Takes P_MARK_MESSAGE, returns STATUS.

#define msgMarkEnterParent MakeMsg(clsMark, 122)

typedef struct MARK_MESSAGE {
    MARK_MSG_HEADER header;
} MARK_MESSAGE, *P_MARK_MESSAGE;

This message may send msgMarkGetParent to the component to find out who the parent is.

msgMarkgetToken
Sent from one mark to another to get the other's token.

Takes P_MARK_TOKEN, returns STATUS.

#define msgMarkgetToken MakeMsg(clsMark, 123)

typedef struct MARK_TOKEN {
    CLASS classLevel;   // which class level is the data at
    U32 index;          // index to the data item
    U32 index2;         // secondary index if needed
} MARK_TOKEN, *P_MARK_TOKEN;

This is not intended to be used by clients of mark.
This file contains the API for clsPrintFrame. clsPrFrame inherits from clsCustomLayout. Provides the page outline during printing.

```c
#ifndef PRFRAME_INCLUDED
#define PRFRAME_INCLUDED
#ifndef GO_INCLUDED
#include <go.h>
#endif
#ifndef UID_INCLUDED
#include <uid.h>
#endif
#ifndef PRINT_INCLUDED
#include <print.h>
#endif

Common #defines and typedefs

Window Tags for Child Windows

#define tagPrFrameLeftHeader          MakeTag(clsPrFrame, 255)
#define tagPrFrameCenterHeader        MakeTag(clsPrFrame, 254)
#define tagPrFrameRightHeader         MakeTag(clsPrFrame, 253)
#define tagPrFrameLeftFooter          MakeTag(clsPrFrame, 252)
#define tagPrFrameCenterFooter        MakeTag(clsPrFrame, 251)
#define tagPrFrameRightFooter         MakeTag(clsPrFrame, 250)
#define tagPrFrameMarginWin            MakeTag(clsPrFrame, 249)
#ifndef NO_NEW
#ifndef prFrameNewFields
#include <clayout.h>
#endif
#endif

Messages

msgNewDefaults
Initializes the PRFRAME_NEW_ONLY structure to default values.

Takes P_PRFRAME_NEW, returns STATUS. Category: class message.

typedef PRINT_SETUP PRFRAME_NEW_ONLY, *P_PRFRAME_NEW_ONLY;
#define prFrameNewFields                     
    customLayoutNewFields               
    PRFRAME_NEW_ONLY prFrame;
Arguments
typedef struct PRFRAME_NEW {
    prFrameNewFields
} PRFRAME_NEW, *P_PRFRAME_NEW;
```
msgNewDefaults

Initializes the default new arguments.

Takes P_PRFRAME_NEW, returns STATUS. Category: class message.

datatype struct PRFRAME_NEW {
    prFrameNewFields
} PRFRAME_NEW, *P_PRFRAME_NEW;

no header or footer text
headerMargin.top = 500 Mils
headerMargin.left = 750 Mils
headerMargin.right = 750 Mils
headerMargin.bottom = 0 Mils
footerMargin.top = 0 Mils
footerMargin.left = 750 Mils
footerMargin.right = 750 Mils
footerMargin.bottom = 500 Mils
mainMargin.top = 750 Mils
mainMargin.left = 750 Mils
mainMargin.right = 750 Mils
mainMargin.bottom = 750 Mils

msgNew

Creates a new print frame object.

Takes P_PRFRAME_NEW, returns STATUS. Category: class message.

#endif
#endif
datatype struct PRFRAME_NEW {
    prFrameNewFields
} PRFRAME_NEW, *P_PRFRAME_NEW;

msgPrFrameSend

Sends the tagged window the message.

Takes P_PRFRAME_SEND, returns STATUS.

datatype struct PRFRAME_SEND {
    U32 tag; // window tag
    MESSAGE msg; // message to send
    P_ARGS pArgs; // arguments to pass
    SIZEOF lenSend; // argument length
} PRFRAME_SEND, *P_PRFRAME_SEND;
#define msgPrFrameSend MakeMsg(clsPrFrame, 1)

msgPrFrameSetup

Sets the print frame values/fields to the setup information.

Takes P_PRINT_SETUP, returns STATUS.

#define msgPrFrameSetup MakeMsg(clsPrFrame, 2)
**msgPrFrameExpand**

Expand any abbreviated labels for the current page/date/doc name.

Takes P_PRFRAME_EXPAND, returns STATUS.

```c
typedef struct PRFRAME_EXPAND {
    char page[nameBufLength];    // printing page number (pg.)
    char date[nameBufLength];    // date string (dt.)
    char name[nameBufLength];    // doc name (nm.)
    char reserved[nameBufLength];
} PRFRAME_EXPAND, *P_PRFRAME_EXPAND;
#define msgPrFrameExpand MakeMsg(clsPrFrame, 3)
```
This file contains the API for clsPrint.

clsPrint inherits from clsApp.

Provides a wrapper to guide PenPoint documents through the printing process.

To print a document, the Application Framework creates a wrapper document (an instance of clsPrint) that embeds the document to be printed in itself. To print the document, the wrapper first opens the document to the printer (rather than to the screen). The wrapper then uses and instance of clsPrLayout to send printing-related messages to the document and any of its embedded documents.

Developers: You should not subclass clsPrint. However, to support printing, your application needs to handle many of the messages defined by clsPrint.

**Pagination**

There are two basic styles of pagination: flow and nonflow. The printing wrapper sends msgPrintGetProtocols to a document to ask it what style of pagination it supports.

For more information on pagination, please refer to the chapter on Printing in the PenPoint Architectural Reference.

**Option Cards for Printing**

The Application Framework provides a Print Setup option sheet, which allows the user to change margins and the running headers and footers that are printed with a document.

If your application has other printing options that you want the user to change, you should add your option cards to the Print Setup sheet. To do so, your application should handle msgAppAddCards and should add your cards when the tag passed in is tagAppPrintSetupOptSheet.

```c
ifndef PRINT_INCLUDED
#define PRINT_INCLUDED
#endif

ifndef UUID_INCLUDED
#include <uuid.h>
#endif

ifndef CLSMGR_INCLUDED
#include <clsmgr.h>
#endif

ifndef GEO_INCLUDED
#include <geo.h>
#endif

ifndef SYSFONT_INCLUDED
#include <sysfont.h>
#endif

ifndef WIN_INCLUDED
#include <win.h>
#endif

ifndef EMBEDWIN_INCLUDED
#include <embedwin.h>
#endif
```
# Common #defines and typedefs

## Status Codes

```c
#define stsPrintErrorCheckPrinter  MakeStatus(clsPrint, 1)
```

## Print Option Sheet Tags

```c
#define tagPrJobDialog           MakeTag(clsPrint, 255)
#define tagPrOption              MakeTag(clsPrint, 254)
#define tagPrPrinterLabel        MakeTag(clsPrint, 253)
#define tagPrPaperSizeLabel      MakeTag(clsPrint, 251)
#define tagPrStartingPageLabel   MakeTag(clsPrint, 250)
#define tagPrPrinter             MakeTag(clsPrint, 249)
#define tagPrStatus              MakeTag(clsPrint, 248)
#define tagPrEnabledOn           MakeTag(clsPrint, 247)
#define tagPrEnabledOff          MakeTag(clsPrint, 246)
#define tagPrPages               MakeTag(clsPrint, 245)
#define tagPrPagesAll            MakeTag(clsPrint, 244)
#define tagPrPagesRange          MakeTag(clsPrint, 243)
#define tagPrPagesFrom           MakeTag(clsPrint, 242)
#define tagPrPagesTo             MakeTag(clsPrint, 241)
#define tagPrPaperSize           MakeTag(clsPrint, 240)
#define tagPrPaperWidth          MakeTag(clsPrint, 239)
#define tagPrPaperHeight         MakeTag(clsPrint, 238)
#define tagPrStartingPage        MakeTag(clsPrint, 237)
#define tagPrStartingPageLabel   MakeTag(clsPrint, 236)
```  

## Print Margins

This structure contains the margin offsets (in Mils) measured from the top, bottom, left, and right edges of the paper.

```c
typedef struct PRINT_MARGINS {
    S32 top;           // offset for top margin
    S32 bottom;        // offset for bottom margin
    S32 left;          // offset for left margin
    S32 right;         // offset for right margin
} PRINT_MARGINS, *P_PRINT_MARGINS;
```

## Header and Footer Strings

This structure contains the strings for either a header or a footer.

```c
typedef struct PRINT_HFDATA {
    U8 reserved;       // reserved - must be 0
    char leftData[nameBufLength]; // string on left side
    char centerData[nameBufLength]; // string in center
    char rightData[nameBufLength];  // string on right side
} PRINT_HFDATA, *P_PRINT_HFDATA;
```

## Print Setup

This structure contains setup information for printing.

```c
typedef struct PRINT_SETUP {
    OBJECT frame;           // reserved
    PRINT_MARGINS mainMargins; // print margins for the document
    PRINT_MARGINS headerMargins; // print margins for the header
    PRINT_MARGINS footerMargins; // print margins for the footer
    PRINT_HFDATA headerInfo;  // strings to display in the header
    PRINT_HFDATA footerInfo;  // strings to display in the footer
    SYSDC_FONT_SPEC fontSpec;  // header/footer font data
    U16 fontSize;             // header/footer font size, in points
} PRINT_SETUP, *P_PRINT_SETUP;
```
**Embeddee Print Info**

Users can decide:

- to not print an embedded document;
- to print the visible portion of an embedded document in the place in the parent document where it is embedded;
- to print the entire embedded document at the end of the parent.

This structure contains information for printing embedded documents. Note: `expandInPlace` and `printBorders` are not currently implemented.

```c
typedef struct EMBEDDEE_PRINT_INFO {
    U16 expandInPlace : 1; // TRUE to print entire contents in place
    U16 expandAtEnd : 1; // TRUE to print entire contents at end
    U16 invisible : 1; // TRUE to not print
    U16 printBorders : 1; // TRUE to show borders around the window
    U16 reserved : 12; // reserved
    U16 reserved2 : 16; // reserved
} EMBEDDEE_PRINT_INFO, *P_EMBEDDEE_PRINT_INFO;
```

**Spool mode values**

Note: Spooling is not implemented.

```c
#define prModeCopy 0 // to copy the doc for spooling
#define prModeLock 1 // to lock the doc for spooling
```

**Print Metrics**

This structure defines the public instance data that `clsPrint` maintains for a document. You get a copy of this structure when you send `msgPrintGetMetrics` to a document.

```c
typedef struct PRINT_METRICS {
    U32 reserved1; // reserved
    U16 pageRangeStart; // start page # (not used if pageAll is TRUE)
    U16 pageRangeEnd; // end page # (not used if pageAll is TRUE)
    U16 startingPage; // starting page # (to be printed on pages)
    U16 copies; // not used
    U16 collating: 2; // not used
    U16 orientation: 2; // either prOrientPortraitNormal or pdOrientLandscapeNormal (see win.h)
    U16 pageAll: 1; // TRUE to print all pages
    U16 spoolMode: 2; // see spool mode values
    U16 firstPageHeader: 1; // TRUE to enable first page headers
    U16 reserved2: 8; // reserved
    U8 pageSizeType; // Popular paper type (see clsPrn.h)
    SIZE32 pageSize; // Size of paper in Mils
    PRINT_SETUP firstPageSetup; // not used
    PRINT_SETUP pageSetup; // page setup information
    char *printer[nameBufLength]; // name of printer to use
    EMBEDDEE_PRINT_INFO embedding; // how to print embedded documents
    U32 reservedData[6]; // reserved
} PRINT_METRICS, *P_PRINT_METRICS;
```

**Print Embeddee Action**

This structure is used by `msgPrintEmbeddeeAction` and `msgPrintExamineEmbeddee` to pass information about the child being processed.
typedef struct PRINT_EMBEDDEE_ACTION {
    WIN embeddedWin; // embedded win to act on
    U16 action; // proposed embeddee action flag
    EMBEDDEE_PRINT_INFO embedPrintInfo; // embeddee print properties
    U32 reserved[3]; // reserved
} PRINT_EMBEDDEE_ACTION, *P_PRINT_EMBEDDEE_ACTION;

void Embeddee Action Flags

#define prEmbedActionAsIs 0 // visible part printed in place (default)
#define prEmbedActionExpandInPlace 1 // not supported
#define prEmbedActionExtract 2 // invisible or moved to end; either
                                 // way, child removed from parent

Print Page

This structure is used by msgPrintStartPage and msgPrintLayoutPage to pass information about what page needs to be printed next.

typedef struct PRINT_PAGE {
    U16 pageNumber; // In: #pages printed when this one is done
    U16 displayPageNumber; // In: page number to display on page
    U16 logicalPageNumber; // In: #times msgPrintStartPage has been sent
    OBJECT jobUID; // In: print layout driver object
    OBJECT appLayoutUID; // Out: obj to receive msgPrintEmbeddeeAction
    U32 reserved[3]; // reserved
} PRINT_PAGE, *P_PRINT_PAGE;

Messages

msgPrintStartPage

Advance the document to its next logical page.

Takes P_PRINT_PAGE, returns STATUS.

#define msgPrintStartPage MakeMsg(clsPrint, 1)

typedef struct PRINT_PAGE {
    U16 pageNumber; // In: #pages printed when this one is done
    U16 displayPageNumber; // In: page number to display on page
    U16 logicalPageNumber; // In: #times msgPrintStartPage has been sent
    OBJECT jobUID; // In: print layout driver object
    OBJECT appLayoutUID; // Out: obj to receive msgPrintEmbeddeeAction
    U32 reserved[3]; // reserved
} PRINT_PAGE, *P_PRINT_PAGE;

This message is sent to a document as a signal to initialize its internal pagination data to a new page. When the document has no more pages to print it should return stsEndOfData in response to this message. Note: the document does not return stsEndOfData when it paginates its last page; it waits until the next time this message is sent (when it has no data left to paginate). If the document does have more pages to print, the following happens:

- the document receives msgPrintGetProtocols
- the mainWin of document receives msgWinLayout at least once
- the document receives msgPrintLayoutPage
If `appLayoutUID` is `objNull`, the print layout driver will send any messages regarding embeddee actions (msgPrintEmbeddeeAction) to the document; otherwise it will send them to the `appLayoutUID` object set by the document in this structure.

Developers: Your application should handle this message to support pagination.

### msgPrintLayoutPage

Document lays out its logical page.

Takes P_PRINT_PAGE, returns STATUS.

```c
#define msgPrintLayoutPage MakeMsg(clsPrint, 12)
```

**Message Arguments**

- `U16 pageNumber;` // In: \# pages printed when this one is done
- `U16 displayPageNumber;` // In: page number to display on page
- `U16 logicalPageNumber;` // In: \# times msgPrintStartPage has been sent
- `OBJECT jobUID;` // In: print layout driver object
- `OBJECT appLayoutUID;` // Out: obj to receive msgPrintEmbeddeeAction
- `U32 reserved[3];` // reserved

**Comments**

The wrapper sends this message to the document after it sends msgPrintStartPage and msgPrintGetProtocols. This message can be thought of as a substitute for msgWinLayout. However, unlike msgWinLayout, it is sent only once per page.

Developers: Your application should handle this message to support pagination.

### msgPrintGetMetrics

Gets the application's print metrics.

Takes P_PRINT_METRICS, returns STATUS.

```c
#define msgPrintGetMetrics MakeMsg(clsPrint, 2)
```

**Message Arguments**

- `U32 reserved1;` // reserved
- `U16 pageRangeStart;` // start page \# (not used if pageAll is TRUE)
- `U16 pageRangeEnd;` // end page \# (not used if pageAll is TRUE)
- `U16 startingPage;` // starting page \# (to be printed on pages)
- `U16 copies;` // not used
- `U16 collating: 2;` // not used
- `U16 orientation: 2;` // either pOrientPortraitNormal or pOrientLandscapeNormal (see win.h)
- `U16 pageAll: 1;` // TRUE to print all pages
- `U16 spoolMode: 2;` // see spool mode values
- `U16 firstPageHeader:1;` // TRUE to enable first page headers
- `U16 reserved2: 8;` // reserved
- `U8 pageSizeType;` // Popular paper type (see clsPrn.h)
- `SIZE32 pageSize;` // Size of paper in Mils
- `PRINT_SETUP firstPageSetup;` // not used
- `PRINT_SETUP pageSetup;` // page setup information
- `char printer[nameBufLength];` // name of printer to use
- `EMBEDDEE_PRINT_INFO embedding;` // how to print embedded documents
- `U32 reservedData[6];` // reserved

**Comments**

You can send this message to OSThisApp() to get the current application's print metrics. During printing you can send this message to the jobUID (given in the pArgs of msgPrintStartPage) to get EFFECTIVE print metrics. EFFECTIVE print metrics are those from the original top-level document.
in this print job. Deferred embedded documents print with effective margins, headers and footers, and orientation; the values in their own print metrics are ignored.

Developers: Your application does not need to handle this message.

**msgPrintSetMetrics**

Sets the application's print metrics.

Takes P_PRINT_METRICS, returns STATUS.

```c
#define msgPrintSetMetrics MakeMsg(clsPrint, 3)
```

**Message**

typedef struct PRINT_METRICS {
    U32 reserved;
    U16 pageRangeStart; // start page # (not used if pageAll is TRUE)
    U16 pageRangeEnd;  // end page # (not used if pageAll is TRUE)
    U16 startingPage;  // starting page # (to be printed on pages)
    U16 copies;       // not used
    U16 collating: 2;  // not used
    U16 orientation: 2; // either prOrientPortraitNormal or
                        // pdOrientLandscapeNormal (see win.h)
    U16 pageAll: 1;    // TRUE to print all pages
    U16 spoolMode: 2;  // see spool mode values
    U16 firstPageHeader: 1; // TRUE to enable first page headers
    U16 reserved2: 8; // reserved
    U8 pageSizeType;  // Popular paper type (see clsPrn.h)
    SIZE32 paperSize; // Size of paper in Mils
    PRINT_SETUP firstPageSetup; // not used
    PRINT_SETUP pageSetup;    // page setup information
    char printer[nameBufLength]; // name of printer to use
    EMBEDDED_PRINT_INFO embedding; // how to print embedded documents
    U32 reservedData[6]; // reserved
} PRINT_METRICS, *P_PRINT_METRICS;
```

**Comments**

You can send this message to OSThisApp() to set the current application's print metrics.

Developers: Your application does not need to handle this message.

**msgPrintApp**

Prints a document.

Takes P_PRINT_DATA, returns STATUS.

```c
#define msgPrintApp MakeMsg(clsPrint, 4)
```

**Arguments**

typedef struct PRINT_DATA {
    OBJECT appUID;    // In: UID if this is the active app
    UUID appUUID;     // In: application UUID
    U32 reserved[2];  // reserved
} PRINT_DATA, *P_PRINT_DATA;
```

**Comments**

This message prints the document. If you want to invoke printing, you send this message to thePrintManager, using ObjectSend or ObjectPost.

Developers: Your application does not need to handle this message.

**msgPrintPaperArea**

Passes back the width and height of the printing area on the paper.

Takes P_PRINT_AREA, returns STATUS.

```c
#define msgPrintPaperArea MakeMsg(clsPrint, 7)
```
typedef struct PRINT_AREA {
    P_PRINT_METRICS pMetrics; // In: pNull or metrics for computation
    SIZE32 size; // Out: size of print area, in Mils.
} PRINT_AREA, *P_PRINT_AREA;

thePrintManager returns the size of the printing area on a sheet of paper, adjusted to take into account margin values and interpreted relative to the orientation. Thus, thePrintManager swaps the computed width and height values if the page orientation is landscape vs portrait.

The size of the printing area is in Mils. It does not account for printer hardware limitations, i.e., the "unprintable area" on a page.

You can send this message to thePrintManager at any time to get the current document's printing area. You can either pass in the metrics from which to compute the area or set pMetrics = pNull. If pMetrics is pNull, thePrintManager will obtain the print metrics from theProcessResList.

Developers: Your application does not need to handle this message.

msgPrintGetProtocols

Gets the pagination and embeddee printing protocols for the document.

Takes P_PRINT_PROTOCOLS, returns STATUS.

#define msgPrintGetProtocols MakeMsg(clsPrint, 9)

typedef struct PRINT_PROTOCOLS {
    U16 paginationMethod; // Out: paginationMethod value
    U16 embeddeeSearch; // Out: embeddeeSearch value
} PRINT_PROTOCOLS, *P_PRINT_PROTOCOLS;

The wrapper sends this message to the document after each msgPrintStartPage.

Developers: Your application needs to handle this message and pass back the pagination method (see "paginationMethod Values" below) and the embeddee searching method (see "embeddeeSearch Values").

paginationMethod Values

#define prPaginationTile 1 // tile pagination style
#define prPaginationFlow 2 // flow pagination style
#define prPaginationScale 3 // scale pagination style

embeddeeSearch Values

#define prEmbeddeeSearchByPrintJob 1 // print layout driver finds children
#define prEmbeddeeSearchByApp 2 // app finds children while paginating

msgPrintEmbeddeeAction

Asks the document for permission to perform an action on an embeddee.

Takes P_PRINT_EMBEDDEE_ACTION, returns STATUS.

#define msgPrintEmbeddeeAction MakeMsg(clsPrint, 10)

typedef struct PRINT_EMBEDDEE_ACTION {
    WIN embeddedWin; // embedded win to act on
    U16 action; // proposed embeddee action flag
    EMBEDDEE_PRINT_INFO embedPrintInfo; // embeddee print properties
    U32 reserved[3]; // reserved
} PRINT_EMBEDDEE_ACTION, *P_PRINT_EMBEDDEE_ACTION;
The wrapper sends this message to the (top-level) document being printed; it requests permission to perform an action on an embeddee.

Developers: You should handle this message and return \texttt{stsOK} for yes, \texttt{stsRequestDenied} for no.

In parameters:

- \texttt{embeddedWin} embedded win to act on
- \texttt{action} proposed embeddee action
- \texttt{embedPrintInfo} embeddee print properties

---

\textbf{msgPrintExamineEmbeddee}

Sent to the print layout driver to interpret an embedded window's print properties.

Takes \texttt{P_PRINT_EMBEDDEE_ACTION}, returns \texttt{STATUS}.

\begin{verbatim}
#define msgPrintExamineEmbeddee MakeMsg(clsPrint, 11)

typedef struct PRINT_EMBEDDEE_ACTION {
    WIN embeddedWin;     // embeddee action
    U16 action;          // proposed embeddee action flag
    EMBEDDEE_PRINT_INFO embedPrintInfo;  // embeddee print properties
    U32 reserved[3];     // reserved
} PRINT_EMBEDDEE_ACTION, *P_PRINT_EMBEDDEE_ACTION;
\end{verbatim}

Documents that are being printed (or their layout objects) can send this message to the wrapper. It tells the print layout driver to interpret the embedded win's print properties and propose an action via \texttt{msgPrintEmbeddeeAction}. \texttt{msgPrintEmbeddeeAction} is sent subsequently even if no action is necessary.

In parameters:

- \texttt{embeddedWin} embedded win to examine

Out parameters:

- \texttt{action} proposed embeddee action
- \texttt{embedPrintInfo} embeddee print properties

Developers: You do not need to handle this message.

---

\textbf{msgPrintSetPrintableArea}

Sent to the \texttt{printJob} to adjust margins for the "unprintable area".

Takes \texttt{PRINTABLE_AREA}, returns \texttt{STATUS}.

\begin{verbatim}
#define msgPrintSetPrintableArea MakeMsg(clsPrint, 13)
#define prAdjustActualForUnprintable flagO

typedef struct PRINTABLE_AREA {
    U16 flags;
    PRINT_MARGINS printMetricsMargins;  // user-set margins
    PRINT_MARGINS unprintableMargins;   // hardware limitations
    PRINT_MARGINS actualMargins;        // actual margins used by print
} PRINTABLE_AREA, *P_PRINTABLE_AREA;
\end{verbatim}

A (top-level) document can send this to the \texttt{printJob} during printing as a request to adjust margins to account for printer hardware limitations (i.e., an unprintable area on the page). It affects only the current page. You typically first send \texttt{msgPrintGetPrintableArea} to get the margins that the \texttt{printJob} is
currently using. Then you can set the flags argument to `ptAdjustActualForUnprintable`, and send the structure on to this message.

Automatic tiling by the `printJob` always adjusts the user-set (print metrics) margins to account for the unprintable area on the page.

Typically graphics (non-flow) applications will desire this type of adjustment, while word processing (flow) apps won't since it may cause data reformatting. Sometimes, as with text, it is more user-friendly not to adjust (and let the data get clipped) so that the source of the problem is obvious to the user. Auto adjustment may induce unwanted visual changes and obscure their source.

Developers: You do not need to handle this message.

---

**msgPrintGetPrintableArea**

Sent to the print job during printing to request margin information.

Takes `PRINTABLE_AREA`, returns `STATUS`.

```c
#define msgPrintGetPrintableArea MakeMsg(clsPrint, 14)

typedef struct PRINTABLE_AREA {
    U16 flags;
    PRINT_MARGINS printMetricsMargins; // user-set margins
    PRINT_MARGINS unprintableMargins; // hardware limitations
    PRINT_MARGINS actualMargins; // actual margins used by print
} PRINTABLE_AREA, *P_PRINTABLE_AREA;

Flags are ignored.

Developers: You do not need to handle this message.
This file contains the API definition for clsPrLayout.

clsPrLayout inherits from clsObject.

A prLayout object makes a document paginate.

A print layout object guides the top-level document through the pagination process and assists it in implementing its embeddees' print properties.

```c
#ifndef PRLAYOUT_INCLUDED
#define PRLAYOUT_INCLUDED
#endif
#endif
#ifndef UID_INCLUDED
#include <uid.h>
#endif
#ifndef PRINT_INCLUDED
#include <print.h>
#endif

Common #defines and typedefs

```c
typedef struct PRLAYOUT_METRICS {
  OBJECT topLevelApp;   // outermost document printed
  OBJECT currentApp;    // current document being printed
  OBJECT prFrame;       // as top-level
  OBJECT winDev;        // instance of clsPrFrame
  OBJECT printJob;      // printer is bound to this window device
  U32 reserved;
} PRLAYOUT_METRICS, *P_PRLAYOUT_METRICS;
```

Messages

msgNew

Create a new object.

Takes P_PRLAYOUT_NEW, returns STATUS.

```c
#define prLayoutNewFields
objectNewFields
PRLAYOUT_METRICS prLayout;
```
**msgPrLayoutGetMetrics**

Get PrLayout metrics.

Takes P_PRLAYOUT_METRICS, returns STATUS.

```c
#define msgPrLayoutGetMetrics MakeMsg(clsPrLayout, 1)
```

```c
typedef struct PRLAYOUT_METRICS {
    OBJECT topLevelApp; // outermost document printed
    OBJECT currentApp; // current document being printed
    OBJECT prFrame; // as top-level
    OBJECT winDev; // instance of clsPrFrame
    OBJECT printJob; // printer is bound to this window device
    U32 reserved; // "owner" of this object
} PRLAYOUT_METRICS, *P_PRLAYOUT_METRICS;
```

**msgPrLayoutSetMetrics**

Set PrLayout metrics.

Takes P_PRLAYOUT_METRICS, returns STATUS.

```c
#define msgPrLayoutSetMetrics MakeMsg(clsPrLayout, 2)
```

```c
typedef struct PRLAYOUT_METRICS {
    OBJECT topLevelApp; // outermost document printed
    OBJECT currentApp; // current document being printed
    OBJECT prFrame; // as top-level
    OBJECT winDev; // instance of clsPrFrame
    OBJECT printJob; // printer is bound to this window device
    U32 reserved; // "owner" of this object
} PRLAYOUT_METRICS, *P_PRLAYOUT_METRICS;
```

**msgPrLayoutNextPage**

Get next page.

Takes PRLAYOUT_PAGE, returns STATUS.

```c
#define msgPrLayoutNextPage MakeMsg(clsPrLayout, 3)
```

```c
typedef struct PRLAYOUT_PAGE {
    U16 pageNumber; // In: paper sheets
    U16 displayPageNumber; // In: number displayed on page
    U16 logicalPageNumber; // Out: num times msgPrintStartPage sent
    OBJECT currentApp; // Out: top level app supplying current page
    BOOLEAN appChanged; // Out: true if first page from currentApp
} PRLAYOUT_PAGE, *P_PRLAYOUT_PAGE;
```

**Comments**

Uses print protocol messages defined in print.h to get the next page from the document being printed.
This file contains the API for clsPrMargin.

clsPrMargin inherits from clsWin.

Provides clipping of children.

```
#ifndef PRMARGIN_INCLUDED
#define PRMARGIN_INCLUDED
#ifndef GO_INCLUDED
#include <go.h>
#endif
#ifndef UID_INCLUDED
#include <uid.h>
#endif
#ifndef CLSMGR_INCLUDED
#include <clsmgr.h>
#endif
#ifndef CLAYOUT_INCLUDED
#include <clayout.h>
#endif
```

**Common #defines and typedefs**

### msgNew

Create a new object.

Takes P_PRMARGIN_NEW, returns STATUS.

**Arguments**

```
typedef struct PRMARGIN_NEW_ONLY {
    OBJECT client; // object to adjust layout
} PRMARGIN_NEW_ONLY, *P_PRMARGIN_NEW_ONLY;
```

**Comments**

The prmargin object handles msgCstmLayoutGetChildSpec and then sends it on to the client for adjustment of default layout behavior.

### msgPrMarginSetMetrics

Set the prMargin metrics.

Takes P_PRMARGIN_METRICS, returns STATUS.

**Arguments**

```
typedef struct PRMARGIN_METRICS {
    OBJECT client;
} PRMARGIN_METRICS, *P_PRMARGIN_METRICS;
```

**Comments**

```
#define msgPrMarginSetMetrics MakeMsg(clsPrMargin, 1)
```
This file contains the API definition for clsRootContainerApp.

clsRootContainerApp inherits from clsApp.

Abstract class for root containers.

This class defines the API for all root container applications. Root containers are expected to respond to this API as part of their implementation.

PenPoint includes one implementation of a root container: the notebook. The messages defined in this class allow programmatic control of a root container application.

To get the uid of the root container of interest use msgAppGetRoot (see app.h) or msgAppMgrGetRoot (see appmgr.h).

```c
#ifndef RCAPP_INCLUDED
#define RCAPP_INCLUDED
#include <clsmgr.h>
#include <uuid.h>
```

## Common #defines and typedefs

```c
typedef OBJECT RCAPP, *P_RCAPP;
```

## Messages

### Sequential Access Messages

The next four messages provide sequential access to documents within the target root container.

**msgRCAppNextDoc**

Increments a root container’s internal pointer to the next document.

- Takes nothing, returns STATUS.
- `#define msgRCAppNextDoc MakeMsg(clsRootContainerApp, 1)`

**Comments**

This message is sent to a root container to cause it to move to the next page. This message does not actually cause the page turn to occur. After one or more `msgRCAppNextDoc`, you must send `msgRCAppExecuteGotoDoc` to actually force the page turn to happen.

**msgRCAppPrevDoc**

Decrements a root container’s internal pointer to the previous document.

- Takes nothing, returns STATUS.
- `#define msgRCAppPrevDoc MakeMsg(clsRootContainerApp, 2)`
This message is sent to a root container to cause it to move to the previous page. This message does not actually cause the page turn to occur. After one or more msgRCAppPrevDoc, you must send msgRCAppExecuteGotoDoc to actually force the page turn to happen.

**msgRCAppExecuteGotoDoc**

Turns a root container to the page pointed to by its internal pointer.

Takes nothing, returns STATUS.

```c
#define msgRCAppExecuteGotoDoc MakeMsg(clsRootContainerApp, 3)
```

**Comments**

Send this message after a series of msgRCAppNextDoc or msgRCAppPrevDoc calls to force the page turn to happen.

**msgRCAppCancelGotoDoc**

Resets a root container’s internal pointer to the current document.

Takes P_UUID, returns STATUS.

```c
#define msgRCAppCancelGotoDoc MakeMsg(clsRootContainerApp, 4)
```

**Comments**

Send this message after a series of msgRCAppNextDoc or msgRCAppPrevDoc calls to cancel the calls reset the root container’s internal pointer to the current page.

**Random Access Messages**

The next two messages provide random access to documents within the target root container.

**msgRCAppGotoContents**

Turns a root container to its contents page.

Takes nothing, returns STATUS.

```c
#define msgRCAppGotoContents MakeMsg(clsRootContainerApp, 5)
```

**Comments**

Send this message to a root container to force it to turn to its table of contents.

**msgRCAppGotoDoc**

Turns a root container to a document, or floats the document over the current page.

Takes P_RCAPP_GOTO_DOC, returns STATUS.

```c
#define msgRCAppGotoDoc MakeMsg(clsRootContainerApp, 6)
```

**Arguments**

typedef struct RCAPP_GOTO_DOC {
    BOOLEAN gotoDoc;           // True=turn to, False=float.
    UUID    docUUID;           // UUID of target document.
    UUID    reserved1;         // Reserved.
    U32     reserved2[2];     // Reserved.
    char    reserved3[nameBufLength]; // Reserved.
    U32     reserved4[4];     // Reserved.
} RCAPP_GOTO_DOC, *P_RCAPP_GOTO_DOC;

**Comments**

Send this message to a root container to turn to or float a document. The specified document must be within the root container.
This file contains the API definition for clsView.

clsView inherits from clsCustomLayout.

clsView is an abstract class that defines an association between a data object and a view onto that data.

Since clsView is an abstract class it should never be directly instantiated.

```c
#ifndef VIEW_INCLUDED
#define VIEW_INCLUDED
#endif
#ifndef CLAYOUT_INCLUDED
#include <clayout.h>
#endif
```

# Common #defines and typedefs

typedef OBJECT VIEW, *P_VIEW;

# Messages

**msgNew**

Creates a new view.

Takes P_VIEW_NEW, returns STATUS. Category: class message.

**Arguments**

typedef struct VIEW_NEW_ONLY {
    OBJECT dataObject; // Data object to view.
    BOOLEAN createDataObject; // Auto-create data object?
} VIEW_NEW_ONLY, *P_VIEW_NEW_ONLY;

#define viewNewFields
customLayoutNewFields \ VIEW_NEWONLY view;

typedef struct VIEW_NEW {
    viewNewFields
} VIEW_NEW, *P_VIEW_NEW;

**Comments**

If pArgs->view.dataObject is non-null, the new view object becomes an observer of the data object.

clsView does not act on the value of pArgs->view.createDataObject. It is up to descendants of clsView to act on this field, typically by creating a new data object if the field is true. This behavior may not be appropriate of all descendants, however.

Descendants: You should never handle msgNew directly. Instead, handle msgInit by initializing your instance data. The ancestor must be called before your msgInit handler.

**msgNewDefaults**

Initializes the VIEW_NEW structure to default values.

Takes P_VIEW_NEW, returns STATUS. Category: class message.
typedef struct VIEW_NEW {
    viewNewFields
} VIEW_NEW, *P_VIEW_NEW;

In response to this message, clsView does the following:

    pArgs->embeddedWin.style.embeddor = true;
pArgs->embeddedWin.style.embeddee = true;
pArgs->view.dataObject = objNull;
pArgs->view.createDataObject = false;

Descendants: You should handle msgNewDefaults by initializing your _NEW structure to default values. The ancestor must be called before your handler.

---

**msgFree**

Defined in clsmgr.h.

Takes OBJ_KEY, returns STATUS.

Comments

In addition to standard msgFree behavior, the view removes itself as an observer of its data object. It does NOT send msgFree to the data object.

Descendants: You should handle msgFree by destroying all objects and resources you have created. It may be appropriate for you to destroy the data object if your view is the only observer of it. The ancestor must be called after your handler.

---

**msgSave**

Defined in clsmgr.h.

Takes P_OBJ_SAVE, returns STATUS.

Comments

In response to this message, the view sends msgResPutObject to pArgs->file with the data object as the value of pArgs. In effect, this means that saving the view also saves the data object. (If the data object is null, this writes the "null object" id into the resource file.)

Descendants: You should handle msgSave by saving your instance data. The ancestor must be called before your handler.

---

**msgRestore**

Defined in clsmgr.h.

Takes P_OBJ_RESTORE, returns STATUS.

Comments

In response to this message, the view sends msgResGetObject to pArgs->file. In effect, this means that restoring the view also restores the data object. (If the data object was null when the view was saved, the data object is null after msgRestore is handled.)

If the restored data object is non-null, the view becomes an observer of the data object.

Descendants: You should handle msgSave by restoring your instance data. The ancestor must be called before your handler.
**msgFreePending**
Defined in clsmgr.h.
Takes OBJECT, returns STATUS.
Comments
If the object being freed is the view’s data object, the view sets its data object to objNull.
Descendants: If you maintain instance data on the data object, you may need to handle this message by updating your instance data to reflect the impending destruction of the data object. The ancestor should be called before your handler. It is recommended, however, that your view not keep any information on the data object, thus maintaining a strict view/data separation. In such cases, you will not need to handle msgFreePending.

**msgViewSetDataObject**
Specifies a view’s data object.
Takes OBJECT, returns STATUS.
#define msgViewSetDataObject MakeMsg(clsView, 1)
Comments
If the current data object is non-null, the view removes itself as an observer of the current data object. It then sets the current data object to pArgs and, if the new data object is non-null, becomes an observer of it.
Descendants: If you maintain instance data on the data object, you may need to handle this message by updating your instance data to reflect the changed data object. The ancestor may be called before or after your handler. It is recommended, however, that your view not keep any information on the data object, thus maintaining a strict view/data separation. In such cases, you will not need to handle msgViewSetDataObject.

**msgViewGetDataObject**
Passes back a view’s current data object
Takes P_OBJECT, returns STATUS.
#define msgViewGetDataObject MakeMsg(clsView, 2)
Comments
Descendants: You do not normally handle this message.
This file contains the API for clsBitmap.

clsBitmap inherits from clsObject.

Support class for clsIcon (see icon.h). Serves as data object for the Bitmap Editor. Based on cached images (see sysgraf.h).

clsBitmap takes a sampled image description, and optionally a mask, and a hotspot. It will fill this description. It also provides messages to modify the bitmap appearance. The Bitmap Editor treats bitmaps as data objects. It creates a bitmap, files it, and will export it as resource file. This resource file can be processed further by SDK utility programs (see resappnd).

A bitmap will prepare an argument structure for use by msgDcCacheImage so that the sampled image data in the bitmap can be converted to a cached image for quick rendering. See msgBitmapCacheImageDefaults.

```c
#ifndef BITMAP_INCLUDED
#define BITMAP_INCLUDED
#ifndef SYSGRAF_INCLUDED
#include <sysgraf.h>
#endif
#endif
```

**Typedefs, #defines, and Status Values**

```c
#define bitmapResId MakeTag(clsBitmap, 1)
#define bmEncodeNone 0 // no data
#define bmEncodeRunLength 1 // run length encoded
#define bmEncode1BPS 2 // 1 bit per sample
#define bmEncode2BPS 3 // 2 bits per sample
#define bmEncode4BPS 4 // 4 bits per sample
#define bmEncode8BPS 5 // 8 bits per sample
#define bmEncode16BPS 6 // unused (reserved)
#define bmEncode24BPS 7 // unused (reserved)
#define bmMono 0 // default
#define bmColorMap 1 // Not Working (reserved)
#define bmDirectColor 2 // Not Working (reserved)

typedef struct BITMAP_STYLE
{
    U16 pixEncoding : 4,
    maskEncoding : 4,
    colorEncoding : 3,
    version : 5;
} BITMAP_STYLE, *P_BITS_MAP_STYLE;
```
**Messages**

**msgNew**

Creates a bitmap.

Takes P_BITMAP_NEW, returns STATUS. Category: class message.

```c
typedef struct BITMAP_NEW_ONLY
{
    BITMAP_STYLE style;       // overall style
    SIZE16  size;              // # of source samples
    P_U8    pPixels;           // actual samples
    P_U8    pMask;             // mask (must be bmEncode1BPS) or pNull
    XY16    hotSpot;           // lower-left corner relative hot spot
    U32     spare1;
    U32     spare2;
} BITMAP_NEW_ONLY, *P_BITMAP_NEW ONLY,

#define bitMapNewFields \    
    objectNewFields \    
    BITMAP_NEW_ONLY bitmap;

typedef struct BITMAP_NEW
{
    bitMapNewFields
} BITMAP_NEW, *P_BITMAP_NEW;
```

**msgNewDefaults**

Initializes the BITMAP_NEW structure to default values.

Takes P_BITMAP_NEW, returns STATUS. Category: class message.

```c
typedef struct BITMAP_NEW
{
    bitMapNewFields
} BITMAP_NEW, *P_BITMAP_NEW;

bitmap.style.pixEncoding = bmEncode8BPS;
bitmap.style.maskEncoding = bmEncode1BPS;
bitmap.style.colorEncoding = brnMono;
bitmap.style.version = 0;
bitmap.size.w  = 0;
bitmap.size.h  = 0;
bitmap.pPixels = pNull;
bitmap.pMask   = pNull;
bitmap.hotSpot.x = 0;
bitmap.hotSpot.y = 0;
```

**msgBitmapGetMetrics**

Gets bitmap metrics.

Takes P_BITMAP_GET_METRICS, returns STATUS.

```c
#define msgBitmapGetMetrics  
MakeMsg(clsBitmap, 0)
```

**msgBitmapSetMetrics**

Sets bitmap metrics.

Takes P_BITMAP_METRICS, returns STATUS.

```c
#define msgBitmapSetMetrics  
MakeMsg(clsBitmap, 1)
```
**msgBitmapSetSize**
Sets bitmap size, resizing heap block if necessary.
Takes P_SIZE16, returns STATUS.

```c
#define msgBitmapSetSize MakeMsg(clsBitmap, 2)
```

**msgBitmapInvert**
Inverts the colors of the bitmap.
Takes nothing, returns STATUS.

```c
#define msgBitmapInvert MakeMsg(clsBitmap, 3)
```

**msgBitmapLighten**
Lightens the colors of the bitmap by 1/4.
Takes nothing, returns STATUS.

```c
#define msgBitmapLighten MakeMsg(clsBitmap, 4)
```

**msgBitmapFill**
Fills bitmap pixels with RGB value leaving mask alone.
Takes RGB value, returns STATUS.

```c
#define msgBitmapFill MakeMsg(clsBitmap, 6)
```

**msgBitmapCacheImageDefaults**
Prepares argument structure for msgDcCacheImage.
Takes P_SYSDC_CACHE_IMAGE, returns STATUS.

```c
#define msgBitmapCacheImageDefaults MakeMsg(clsBitmap, 43)
```

Comments:
After sending this message to the bitmap, pArgs is ready to be sent to a DC via using msgDcCacheImage (see sysgraf.h).

**Messages sent to observers**

**msgBitmapPixChange**
Sent to observing objects if a pixel is dirty.
Takes P_BITMAP_PIX_CHANGE, returns STATUS. Category: observer notification.

```c
#define msgBitmapPixChange MsgNoError(MakeMsg(clsBitmap, 5))
```

Arguments:
```c
typedef struct BITMAP_PIX_CHANGE {
  XY16     pix;
  OBJECT   sourceObject;
  P_BITMAP_METRICS pBitmap;
} BITMAP_PIX_CHANGE, *P_BITMAP_PIX_CHANGE;
```
msgBitmapChange
Sent to observing objects if bitmap has changed.
Takes nothing, returns STATUS. Category: observer notification.

#define msgBitmapChange        MsgNoError(MakeMsg(clsBitmap, 10))

msgBitmapMaskChange
Sent to observing objects if bitmap's mask has changed.
Takes nothing, returns STATUS. Category: observer notification.

#define msgBitmapMaskChange    MsgNoError(MakeMsg(clsBitmap, 11))
CCITT Fax Group 3, one-dimensional data encoding and decoding routines. The functions described in this file are contained within CCITT.LIB.

ifndef CCITT_INCLUDED
#define CCITT_INCLUDED

typedef struct ENCODE31
{
    U16 pixCnt;     // In: How many pixels in the scanline.
    BOOLEAN photoNegative; // In: Input bitmap's palette:
                           // true: 0 = white, 1 = black.
                           // false: 1 = white, 0 = black.
    P_U8 pScanLine; // In: Scanline data to encode.
                      // Note: A scanline must be
                      // a multiple of words.
    BOOLEAN writeEol; // In: EOL code is to be written at
                       // the beginning of the scanline.
    BOOLEAN writeRtc; // In: 6 EOLs are to be written at
                       // the end of the scanline.
    P_U8 pOutBuf;   // In: Starting byte at which to put data.
                      // The buffer size must accomodate a
                      // worst case encoding for one scanline.
                      // 2*pixCnt, +2 w/EOL, +9 w/RTC.
    U16 inBitPos;   // In: Bit # in pOutBuf to start encoding
                    // Bit 0 = MSB, Bit 7 = LSB
    P_U8 pOutLast;  // Out: Last byte where data was put
                    // Out: Number of bytes used for encoding,
                    // including the last partial byte.
    U16 outBitPos;  // out: Bit # in pOutLast where last bit
                    // was put + 1.
} ENCODE31, *P_ENCODE31;

#define ccittDecodeToPacked 0 // Decode to Packed bitmap.
#define ccittDecodeToRunLen 1 // Decode to sample image operator Run-Length.
#define ccittDecodeToGroup3_1D 2 // Decode to Group 3 1-Dimension fax encoding.

typedef struct DECODE31
{
    S16 format;      // in: ccittDecodeToPacked, RunLen, or Group3_1D.
    S16 pixCnt;      // in: How many pixels comprise a scanline.
    BOOLEAN readEolRtc; // in: EOL or RTC string is to be read
                         // at the end of each scanline.
    BOOLEAN photoNegative; // in: Output palette:
                           // true: 0 = white, 1 = black
                           // false: 1 = white, 0 = black.
    BOOLEAN newLine;  // in/out: Must be set to true at the
                       // start of each scanline and left
                       // alone for remainder of scanline.
    P_U8 pInBuf;     // in: Input buffer:
                    // Starting byte of data to decode.
    U16 inBitPos;    // in: Bit # in pInBuf to start decoding.
                    // Bit 0 = MSB, Bit 7 = LSB
    S16 inBufSz;     // in: # of data bytes within input buffer.
    P_U8 pOutBuf;   // in: Output buffer:
                    // This field should be initialized once
                    // at the beginning of each scanline and
                    // left alone for the rest of the line.
                    // The size of the output buffer must
CcitteEncode31

Encode one scanline of a packed bitmap into fax group 3 T.4 1-D format.

Returns nothing.

Function Prototype

```c
void EXPORTED CcitteEncode31 ( 
    P_ENCODE31 pEncode );
```

CcitteDecode31

Decode one scanline worth of fax group 3 T.4 1-D image data.

Returns BOOLEAN.

Function Prototype

```c
BOOLEAN EXPORTED CcitteDecode31 ( 
    P_DECODE31 pDecode );
```

Output can be either the packed bitmap format, sampled image operatorlength encoded format, or Group 3 1 dimensional image format without. Function returns true if successful, false if the input datanot valid fax data. The interface to this function is such that calls may be needed to decode a complete scanline. As such, states are kept in the interface structure. Fields labeled private are not to be molested by the caller.

Example of decoding a TIFF CCITT/3 image (where there is no EOL or RTC and the number of scanlines is known a priori, using a decoderof our run length format:

```c
decode.format = ccitteDecodeToRunLen; 
decode.pixCnt = 1024; 
decode.readEolRtc = false; 

for (all scanlines) 
{
    decode.inBitPos = 0; 
    decode.pOutBuf = whatever; 
    decode.pInBuf = whatever; 
    decode.inBufSz = whatever; 
    decode.newLine = true; 

    while(true) 
    { 
        if (!CcitteDecode31(&decode)) 
```
break;     // ------ the input data is screwed up.
if (decode.done)
    break;     // ------ done decoding current scanline.

// Supply new bits for next call. Note that there may be
// partial bits left undecoded within the last decoded byte.
// The next call to decode must start with any undecoded bits.
// If you buffer the source bits, then copy all undecoded bits
// into the new buffer. The pInLast and lastBitPos fields tell
// you the amount left undecoded.

decode.pInBuf = pInLast;       // Or your new buffer.
decode.inBufSz = whatever;     // # of bytes w/in buffer.
decode.inBitPos = decode.lastBitPos;     // Assuming that you copy
                                           // *decode.pInLast to new
                                           // buffer.
}

// Done decoding a scanline.

Example of decoding a raw fax input where there is EOLs and RTCand the number of scanlines is not
known a priori, using a format of packed bit output:

decode.format = ccittDecodeToPacked;
decode.inBitPos = 0;
decode.pInBuf = whatever;
decode.inBufSz = whatever;
decode.readEolRtc = true;
decode.rtcRead = false;

while (!decode.rtcRead)
{
    decode.newLine = true;
decode.pOutBuf = whatever;
decode.pixCnt = whatever;     // # of pixels of packed data
                               // *pOutBuf can accommodate.

    while (true)
    {
        if (!CcittDecode31(&decode))
            break;     // ------ the input data is screwed up.

        if (decode.done)
            break;     // ------ done decoding current scanline.

        // Supply new bits for next call. Note that there may be
        // partial bits left undecoded within the last decoded byte.
        // The next call to decode must start with any undecoded bits.
        // If you buffer the source bits, then copy all undecoded bits
        // into the new buffer. The pInLast and lastBitPos fields tell
        // you the amount left undecoded.

        decode.pInBuf = pInLast;  // Or your new buffer.
        decode.inBufSz = whatever;     // # of bytes w/in buffer.
        decode.inBitPos = decode.lastBitPos;     // Assuming that you copy
                                                  // *decode.pInLast to new
                                                  // buffer.
    }

// Done decoding a scanline.
This file contains the API definition for PenPoint's geometry package. The package provides points, rectangles, matrices, etc., and is used by the graphics and windowing software.

Typical application software will only need the types defined in this file and not need to use the functions.

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

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

### Typedefs, #defines, and Status Values

```c
typedef S32 COORD32;
typedef S16 COORD16;
typedef S16 ANGLE;

typedef struct
{
    FIXED x,
    y;
} SCALE, * P_SCALE;
typedef struct
{
    COORD32 x,
    y;
} XY32, * P_XY32;
typedef struct
{
    COORD32 w,
    h;
} SIZE32, * P_SIZE32;
typedef struct
{
    XY32 origin;
    SIZE32 size;
} RECT32, * P_RECT32;

typedef struct
{
    COORD16 x,
    y;
} XY16, * P_XY16;
typedef struct
{
    COORD16 w,
    h;
} SIZE16, * P_SIZE16;
```

// Foley/VanDam counter clockwise angles
typedef struct
{
    XY16 origin;
    SIZE16 size;
} RECT16, *P_RECT16;

Type MAT represents a 3x3 matrix; however m13, m23 and m33 are constant and so they are not stored.

    m11  m12  m13
    m21  m22  m23
    m31  m32  m33

    sX  a  0
    a  sY  0
    tX  tY  1

typedef struct
{
    FIXED m11,
    m12,
    m21,
    m22;
    S32 m31,
    m32;
} MAT, *P_MAT;

Enum16 (GEO_MAT_MULT) {geoPreMultiply, geoPostMultiply};

**Handy macros**

#define Coord32To16(c) ((c>0)?(COORD16)Min(c,maxS16):(COORD16)Max(c,minS16))
#define Coord16from32(c) Coord32To16 (c)
#define RectInit(r, _x, _y, _w, _h) {
    (r)->origin.x = (_x);
    (r)->origin.y = (_y);
    (r)->size.w = (_w);
    (r)->size.h = (_h);
}
#define RectRight(r) ((r)->origin.x + (r)->size.w)
#define RectTop(r) ((r)->origin.y + (r)->size.h)

**Functions**

**Rect16To32**

Take a RECT16 and produce a RECT32.

Returns nothing.

**Function Prototype**

void EXPORTED Rect16To32 (  
P_RECT32 p32, // Out  
P_RECT16 p16  // In  
);

**Rect32To16**

Take a RECT32 and produce a RECT16 with rounding.

Returns nothing.

**Function Prototype**

void EXPORTED Rect32To16 (  
P_RECT16 p16, // Out  
P_RECT32 p32  // In  
);

**Comments**

Each 32-bit number is rounded to 16-bits using Coord32To16.
Rect32Intersect
Take two RECT32's and produce their intersection.

Returns BOOLEAN.

Function Prototype

```c
BOOLEAN EXPORTED Rect32Intersect {
    P_RECT32 pA,   // In
    P_RECT32 pB,   // In
    P_RECT32 pRet  // Out: the intersection
};
```

Comments
Returns whether the two rectangles intersect. When TRUE, the rectangle returned will always have positive width and height, even though either of the parameter rectangles may have negative width or height.

Rect32sIntersect
Test if two RECT32's intersect.

Returns BOOLEAN.

Function Prototype

```c
BOOLEAN EXPORTED Rect32sIntersect {
    P_RECT32 pA,   // In
    P_RECT32 pB    // In
};
```

Comments
Either of the parameter rectangles may have negative width or height.

Rect32EnclosesRect32
Test if a RECT32 encloses another RECT32.

Returns BOOLEAN.

Function Prototype

```c
BOOLEAN EXPORTED Rect32EnclosesRect32 {
    P_RECT32 pA,   // In
    P_RECT32 pB    // In
};
```

Comments
Returns true if rect A completely encloses rect B. Either of the parameter rectangles may have negative width or height.

Rect16Intersect
Take two RECT16's and produce their intersection.

Returns BOOLEAN.

Function Prototype

```c
BOOLEAN EXPORTED Rect16Intersect {
    P_RECT16 pA,   // In
    P_RECT16 pB,   // In
    P_RECT16 pRet  // Out: the intersection
};
```

Comments
Returns whether the two rectangles intersect. When TRUE, the rectangle returned will always have positive width and height, even though either of the parameter rectangles may have negative width or height.
**XY32inRect32**
Test if an XY32 point is inside a RECT32.
Returns BOOLEAN.

**Function Prototype**
```c
BOOLEAN EXPORTED XY32inRect32 (
    P_RECT32 pRect,  // In
    P_XY32  pXY   // In
);
```

**Rect32Empty**
Test if a RECT32 has a width or height that is zero.
Returns BOOLEAN.

**Function Prototype**
```c
BOOLEAN EXPORTED Rect32Empty ( 
    P_RECT32 pRect  // In
);
```

**Comments**
Also, if pRect is pNull then this function returns true.

**Rect16Empty**
Test if a RECT16 has a width or height that is zero.
Returns BOOLEAN.

**Function Prototype**
```c
BOOLEAN EXPORTED Rect16Empty ( 
    P_RECT16 pRect  // In
);
```

**Comments**
Also, if pRect is pNull then this function returns true.

**MatCreate**
Create a MAT given a translate, rotate, and scale.
Returns nothing.

**Function Prototype**
```c
void EXPORTED MatCreate ( 
    P_MAT  pMat,  // Out
    COORD32 tX,  // In
    COORD32 tY,  // In
    ANGLE   angle,  // In
    FIXED   sX,  // In
    FIXED   sY   // In
);
```

**Comments**
pMat is set to identity. Then the three transformation are post-multiplied in the order (1) translate, (2) rotate, and (3) scale.

**MatIdentity**
Set a MAT to the identity matrix.
Returns nothing.

**Function Prototype**
```c
void EXPORTED MatIdentity ( 
    P_MAT  // Out
);
```
MatRotate
Rotate a MAT.
Returns nothing.

Function Prototype
void EXPORTED MatRotate (  
   GEO_MAT_MULT order,  // In: {geoPreMultiply,geoPostMultiply}  
   P_MAT pMat,  // In-Out:  
   ANGLE angle  // In: 0-359 degrees
);

MatTranslate
Translate a MAT.
Returns nothing.

Function Prototype
void EXPORTED MatTranslate (  
   GEO_MAT_MULT order,  // In: {geoPreMultiply,geoPostMultiply}  
   P_MAT pMat,  // In-Out:  
   P_XY32 xy  // In:
);

MatScale
Scale a MAT.
Returns nothing.

Function Prototype
void EXPORTED MatScale (  
   GEO_MAT_MULT order,  // In: {geoPreMultiply,geoPostMultiply}  
   P_MAT pMat,  // In-Out:  
   P_SCALE scale  // In:
);

MatInvert
Invert a MAT.
Returns nothing.

Function Prototype
void EXPORTED MatInvert (  
   P_MAT pDest,  // Out:  
   P_MAT pSource  // In:
);

Comments
pSource is inverted and placed in pDest. pSource and pDest can be the same matrix.

MatMultiply
Multiply two MAT’s.
Returns nothing.

Function Prototype
void EXPORTED MatMultiply (  
   GEO_MAT_MULT order,  // In: {geoPreMultiply,geoPostMultiply}  
   P_MAT answer,  // Out  
   P_MAT left,  // In  
   P_MAT right  // In
);

Comments
If order is geoPreMultiply, then answer = right * left. If order is geoPostMultiply, then answer = left * right;
MatXYTransform16
Transform a XY32 producing a XY16 result.
Returns nothing.

Function Prototype
void EXPORTED MatXYTransform16 (  
  P_MAT pMat,  // In  
  P_XY32 pSource,  // In  
  P_XY16 pDest  // Out  
);

Comments
Each 32-bit number is rounded to 16-bits using Coord32To16.

MatXYTransform32
Transform a XY32 producing a XY32 result.
Returns nothing.

Function Prototype
void EXPORTED MatXYTransform32 (  
  P_MAT pMat,  // In  
  P_XY32 pSource,  // In  
  P_XY32 pDest  // Out  
);

MatWHTransform16
Transform a SIZE32 producing a SIZE16 result.
Returns nothing.

Function Prototype
void EXPORTED MatWHTransform16 (  
  P_MAT pMat,  // In  
  P_SIZE32 pSource,  // In  
  P_SIZE16 pDest  // Out  
);

Comments
This transformation is similar to MatXYTransform16 except the translation components of the matrix are ignored and the values returned are always positive.
Each 32-bit number is rounded to 16-bits using Coord32To16.

MatWHTransform32
Transform a SIZE32 producing a SIZE32 result.
Returns nothing.

Function Prototype
void EXPORTED MatWHTransform32 (  
  P_MAT pMat,  // In  
  P_SIZE32 pSource,  // In  
  P_SIZE32 pDest  // Out  
);

Comments
This transformation is similar to MatXYTransform32 except the translation components of the matrix are ignored and the values returned are always positive.
MatTransformRECT32
Transform a RECT32.
Returns nothing.

Function Prototype  void EXPORTED MatTransformRECT32 (  
                        P_MAT    pMat,  // In
                        P_RECT32 pSource  // In-Out
                    );

Debugging Functions

MatDump
Prints the fields of a matrix.
Returns nothing.

Function Prototype  void EXPORTED MatDump(P_MAT pm);

Comments  This function may not work unless the debugging version of win.dll is being used.

DumpRect
Prints the fields of a rectangle.
Returns nothing.

Function Prototype  void EXPORTED DumpRect(P_RECT32 pRect);

Comments  This function may not work unless the debugging version of win.dll is being used.

Special Functions

WARNING: The functions in this section (MatXTransform16, MatYTransform16,  
MatWTransform16, and MatHTransform16) work only in a limited set of cases: NO translation, NO  
rotation, and they perform NO rounding and thus can overflow the 16 bit result.
These functions should not normally be used by application software.

Function Prototype  COORD16 EXPORTED MatXTransform16(P_MAT pi, COORD16 x);  
COORD16 EXPORTED MatYTransform16(P_MAT pi, COORD16 y);  
COORD16 EXPORTED MatWTransform16(P_MAT pi, COORD16 w);  
COORD16 EXPORTED MatHTransform16(P_MAT pi, COORD16 h);
This file contains the API definition for clsPicSeg (Picture Segments).

clsPicSeg inherits from clsSysDrwCtx.

clsPicSeg provides a database and storage for drawing primitives.

A Picture Segment creates a display list from the stream of messages defined by drawing context. The graphic elements in a PicSeg are called grafics. The display list can repaint to the same window or store the grafics and later repaint it to another window. It also provides a move/copy transfer type for grafics.

The Picture Segment stores the following shapes as defined by clsSysDrwCtx: rectangle, ellipse, Bezier, polyline, polygon, sector rays, arc rays, chord rays, text. In addition, it defines a spline, and object types as an enhancement to the drawing context. It doesn't store images or raster operations such as CopyRect and XOR. Raster operations like XOR, AND, dynamic and fast modes defined by the drawing context apply to the whole display list. Similarly, transformations scale, translate, rotate and units apply to the PicSeg before drawing the list. The PicSeg stores the grafics in Logical Unit Coordinates as defined by the drawing context.

PicSeg's provide display query messages allowing changes to grafic shapes it stores. The grafics in a picture segment are ordered; it keeps track of the current grafic. You can retrieve, alter, reorder, and delete individual grafics.

Common uses of PicSeg's:

PicSeg's generally used as the Data Object of a View (clsView). A drawing View (like clsGrafPaper) translates the input strokes into grafics and draws them to the PicSeg, treating the PicSeg just like a Drawing Context. When the View gets msgWinRepaint it sends msgPicSegPaint to the PicSeg.

The PicSeg's file data as an Object so they can be used as resources. A Drawing View could file many PicSegs with different resource ids to the same file. Latter a display View could look up the different PicSegs in the resource file and display them.

PicSeg's are used to Move/Copy grafic data between Views. The transfer (xfer) mechanism uses an intermediate global PicSeg for grafics.

```c
#ifndef PICSEG_INCLUDED
#define PICSEG_INCLUDED
#ifndef SYSGRAF_INCLUDED
#include <sysgraf.h>
#endif
#endif
```

- **Common #defines and typedefs**

- **Data Collection and Drawing Modes**

  The PicSeg flags determine what to do with a draw messages. By default a message like msgDcDrawRectangle causes the PicSeg to store the rectangle in the display list and draw it on the window set by msgDcSetWindow. The following flags can prevent one or both of these thing from happening (picseg.flags).
#define picSegAdd flag0 // on if PicSeg should add graffics.
#define picSegDraw flag1 // on if PicSeg should draw graffics
#define picSegSendDestroy flag2 // on ObjectCall(msgDestroy, ...)
          // to an object graffic when it is
          // deleted from the PicSeg or if the
          // PicSeg is freed.

The first graffic in the display list is 0. The last can be set by using msgPicSegSetCurrent with
picSegTopGrafic or asking for the current number of graffics and then setting the current graffic.
#define picSegTopGrafic 0x7FFFFFFF // theoretical maximum number of graffics

## OpCodes

Each graffic in the PicSeg is given an OpCode that identifies what type of data is stored in the pData
member of PIC_SEG_GRAFIC.

typedef U16 OP_CODE;
typedef P_U16 P_OP_CODE;
#define opCodeMaskInvisible 0x1000 // graffic.pData
#define opCodePolyline 100 // PIC_SEG_POLYLINE
#define opCodeRectangle 101 // PIC_SEG_RECT
#define opCodeEllipse 102 // PIC_SEG_ELLIPSE
#define opCodePolygon 103 // PIC_SEG_POLYGON
#define opCodeSpline 104 // PIC_SEG_SPLINE
#define opCodeArcRays 105 // PIC_SEG_ARC_RAYS
#define opCodeSectorRays 106 // PIC_SEG_ARC_RAYS
#define opCodeChordRays 107 // PIC_SEG_ARC_RAYS
#define opCodeText 55 // PIC_SEG_TEXT
#define opCodeObject 150 // PIC_SEG_OBJECT

The basic graffic used with msgPicSegGetGrafic. The pData allocated in the a heap and must be freed
by creator of the PicSeg.

typedef struct {
    OP_CODE          opCode;       // the type of graffic stored in pData
    P_UNKNOWN        pData;        // pointer to the graffic data
} PIC_SEG_GRAFIC, * P_PIC_SEG_GRAFIC;

Every graffic provides the basic painting attributes.

typedef struct {
    SYSDC_PATTERN  linePat,      // the line pattern
        fillPat;       // the fill pattern
    SYSDC_RGB      foregroundRGB, // the foreground color
        backgroundRGB; // the background color
    U16            lineThickness; // the line width
} PIC_SEG_PAINT, * P_PIC_SEG_PAINT;

The polyline, polygon, and spline graffics provide line attributes.

typedef struct {
    U8             join;
    U8             cap;
    U8             miterLimit;
    U8             spare;
} PIC_SEG_PLINE_TYPE, * P_PIC_SEG_PLINE_TYPE;

Text style attributes.

typedef struct PIC_SEG_FONT_STYLE{
    U16            alignChr : 3, // see sysDcAlignChr??
        underline : 2, // 0, 1, 2
        strikeout : 2, // 0, 1
        spare : 9; // spare - default 0
} PIC_SEG_FONT_STYLE, P_PIC_SEG_FONT_STYLE;
The \texttt{grafic.pData} provided with \texttt{grafic.opCode == opCodeText}.

\begin{verbatim}
typedef struct PIC_SEG_TEXT{
  PIC_SEG_PAINT paint;
  RECT32 rectangle;
  SYS_DC_FONT_SPEC fontSpec;  // unique font
  PIC_SEG_FONT_STYLE style;
  SIZE16 size;  // size of text
  XY32 cp;      // text position
  U16 length;  // length of text
  U8 text[l];  // null terminated text
} PIC_SEG_TEXT, *P_PIC_SEG_TEXT;
\end{verbatim}

The \texttt{grafic.pData} provided with \texttt{grafic.opCode == opCodeEllipse}.

\begin{verbatim}
typedef struct {
  PIC_SEG_PAINT paint;
  RECT32 ellipse;
} PIC_SEG_ELLIPSE, *P_PIC_SEG_ELLIPSE;
\end{verbatim}

The \texttt{grafic.pData} provided with \texttt{grafic.opCode == opCodeRectangle}.

\begin{verbatim}
typedef struct {
  PIC_SEG_PAINT paint;
  PIC_SEG_RECT rectangle;
  S16 radius;  // The rectangle radius
    // 0 for square corners.
} PIC_SEG_RECT, *P_PIC_SEG_RECT;
\end{verbatim}

The \texttt{grafic.pData} provided with \texttt{grafic.opCode == opCodePolyline}. The pData is of variable size depending on the number of points (pData->count). For Example, the third point is pData->points[3]. The size of pData is: (sizeof(PIC_SEG_POLYLINE) + sizeof(XY32) * ((pData->count)-1)).

\begin{verbatim}
typedef struct {
  PIC_SEG_PAINT paint;
  PIC_SEG_PLINE_TYPE type;
  U16 count;  // number of points
  XY32 points[l];  // variable number of points
} PIC_SEG_POLYLINE, *P_PIC_SEG_POLYLINE;
\end{verbatim}

The \texttt{grafic.pData} provided with \texttt{grafic.opCode == opCodePolygon}. The pData is of variable size depending on the number of points (pData->count). For Example, the third point is pData->points[3]. The size of pData is: (sizeof(PIC_SEG_POLYGON) + sizeof(XY32) * ((pData->count)-1)).

\begin{verbatim}
typedef struct {
  PIC_SEG_PAINT paint;
  PIC_SEG_PLINE_TYPE type;
  U16 count;  // number of points
  XY32 points[l];  // variable number of points
} PIC_SEG_POLYGON, *P_PIC_SEG_POLYGON;
\end{verbatim}

The \texttt{grafic.pData} provided with \texttt{grafic.opCode == opCodeSpline}. A spline is a continuous number of four point Bezier curves. The first Bezier is defined by the first four points in pData->points. The second Bezier starts at pData->points[3]. The count minus one is a multiple of three. \texttt{msgDcDrawBezier} stores as a spline. The pData is of variable size depending on the number of points (pData->count). For Example, the third point is pData->points[3]. The size of pData is: (sizeof(PIC_SEG_SPLINE) + sizeof(XY32) * ((pData->count)-1)).

\begin{verbatim}
typedef struct {
  PIC_SEG_PAINT paint;
  PIC_SEG_PLINE_TYPE type;
  U16 count;  // number of points
  XY32 points[l];  // variable number of points
} PIC_SEG_SPLINE, *P_PIC_SEG_SPLINE;
\end{verbatim}
The grafic.pData provided with grafic.opCode == opCodeArcRays, opCodeChordRays, or opCodeSectorRays.

typedef struct {
    PIC_SEG_PAINT    paint;
    RECT32           bounds;
    XY32             rays[2];
} PIC_SEG_ARC_RAYS, *P_PIC_SEG_ARC_RAYS;

The grafic.pData provided with grafic.opCode == opCodeObject.

typedef struct {
    PIC_SEG_PAINT    paint;
    RECT32           rectangle;
    OBJECT           object;
} PIC_SEG_OBJECT, *P_PIC_SEG_OBJECT;

#define maxPolylineSize ((OxFFFF / sizeof(XY32)) - sizeof(PIC_SEG_POLYLINE))

typedef struct PIC_SEG_METRICS {
    U16 flags;
    MESSAGE units;            // information only
    S32 numberGrafics;        // information only
    S32 currentGrafic;        // information only
    SYSDEC_PATTERN fillPat;   // attributes of the next
    SYSDEC_PATTERN linePat;   // draw grafic
    SYSDEC_RGB foregroundRGB;
    SYSDEC_RGB backgroundRGB;
    SYSDEC_LINE line;
    SYSDEC_PATTERN clearFillPat;  // clear
    SYSDEC_PATTERN clearLinePat;
    SYSDEC_RGB clearForegroundRGB;
    SYSDEC_RGB clearBackgroundRGB;
    SYSDEC_FONT_SPEC fontSpec;  // font stuff
    SIZE16 fontSize;
    PIC_SEG_FONT_STYLE fontStyle;
    S32 reserved[5];           // reserved
    S32 spare[8];              // reserved
} PIC_SEG_NEW_ONLY, PIC_SEG_METRICS,
    *P_PIC_SEG_NEW_ONLY, *P_PIC_SEG_METRICS;

Messages

msgDump

Dumps a PicSeg. Debug version only!

Takes S32, returns STATUS. Category: class message.

Comments

pArgs == 0 everything and dc. pArgs == 1 PicSeg and metrics and does not Dump ancestor. pArgs == 2 PicSeg metrics only and does not Dump ancestor. pArgs == 3 PicSeg database only and does not Dump ancestor.

msgNew

Creates a new PicSeg.

Takes P_PIC_SEG_NEW, returns STATUS. Category: class message.

#define picSegNewFields \
    sysdcNewFields \
    PIC_SEG_NEW_ONLY    picSeg;
typedef struct PIC_SEG_NEW {
    picSegNewFields
} PIC_SEG_NEW, *P_PIC_SEG_NEW;

**msgNewDefaults**

Initializes a PIC_SEG_NEW structure to default values.

Takes P_PIC_SEG_NEW, returns STATUS. Category: class message.

Message Arguments
typedef struct PIC_SEG_NEW {
    picSegNewFields
} PIC_SEG_NEW, *P_PIC_SEG_NEW;

Comments

Defaults:

- picSeg.flags = picSegDraw | picSegAdd | picSegSendDestroy
- picSeg.units = msgDcUnitsPoints
- picSeg.currentGrafic = -1
- picSeg.fillPat = sysDcPatBackground
- picSeg.linePat = sysDcPatForeground
- picSeg.backgroundRGB.all = SysDcGrayRGB(255)
- picSeg.foregroundRGB.all = SysDcGrayRGB(0)
- picSeg.line.cap = 0
- picSeg.line.join = 0
- picSeg.line.miterLimit = 10
- picSeg.line.radius = 0
- picSeg.line.thickness = 1
- picSeg.clearFillPat = sysDcPatNil
- picSeg.clearLinePat = sysDcPatNil
- picSeg.clearForegroundRGB = SysDcGrayRGB(255)
- picSeg.clearBackgroundRGB = SysDcGrayRGB(0)
- picSeg.fontSpec.id = Nil
- picSeg.fontSpec.attr.group = sysDcGroupDefault
- picSeg.fontSpec.attr.weight = sysDcWeightNormal
- picSeg.fontSpec.attr.aspect = sysDcAspectNormal
- picSeg.fontSpec.attr.italic = false
- picSeg.fontSpec.attr.monospaced = false
- picSeg.fontSpec.attr.encoding = sysDcEncodeHWX850
- picSeg.fontSize.w = 1
- picSeg.fontSize.h = 1
- picSeg.fontStyle.alignChr = 0
- picSeg.fontStyle.underline = sysDcAlignChrBaseline
- picSeg.fontStyle.strikeout = 0
- picSeg.fontStyle.spare = 0

**msgRestore**

Restores the PicSeg metrics and grafics and sets the DC state.

Takes P_OBJ_RESTORE, returns STATUS. Category: class message.

Comments

The Restore doesn’t connect the PicSeg to a window. Before using the PicSeg it must be set to a window with msgDcSetWindow.
msgSave
Saves the PicSeg metrics and graphics and the DC units and LUC matrix.
Takes P_OBJ_SAVE, returns STATUS. Category: class message.
Comments
The Save doesn’t save the window connected to the PicSeg.

Drawing Messages
Messages of clsSysDrwCtx used by clsPicSeg: All of the following messages draw the shape and add it as a graphic to the PicSeg display list, provided the add and draw flags are turned on.
msgDcDrawEllipse, msgDcDrawRectangle, msgDcDrawPolyline, msgDcDrawPolygon,
msgDcDrawSectorRays, msgDcDrawArcRays, msgDcDrawChordRays, msgDcDrawBezier,
msgDcDrawText
PicSeg text defaults: spaceChar, spaceExtra, otherExtra
All of the following messages change the DC and also the PicSeg state. PicSeg converts the x,y font scale to 16 bits dc units.
msgDcSetForegroundRGB, msgDcSetBackgroundRGB, msgDcSetLinePat, msgDcSetFillPat,
msgDcSetLine, msgDcSetLineThickness, msgDcOpenFont msgDcScaleFont, msgDcIdentityFont,
msgDcUnits...

msgPicSegPaint
Paints the graphics in the PicSeg.
Takes pNull, returns STATUS.
#define msgPicSegPaint MakeMsg(clsPicSeg, 7)
Comments
Object Call either msgWinBeginPaint or msgWinBeginRepaint before using this message.

msgPicSegDrawSpline
Adds and draws the graphic to the end of the display list.
Takes P_PIC_SEG_SPLINE, returns STATUS.
#define msgPicSegDrawSpline MakeMsg(clsPicSeg, 104)

Message
typedef struct { 
    PIC_SEG_PAINT paint;
    PIC_SEG_PLINE_TYPE type;
    U16 count; // number of points
    XY32 points[1]; // variable number of points
} PIC_SEG_SPLINE, * P_PIC_SEG_SPLINE;

msgPicSegDrawObject
Adds and draws an object to the PicSeg display list.
Takes P_PIC_SEG_OBJECT, returns STATUS.
#define msgPicSegDrawObject MakeMsg(clsPicSeg, 121)

Message
typedef struct { 
    PIC_SEG_PAINT paint;
    RECT32 rectangle;
    OBJECT object;
} PIC_SEG_OBJECT, * P_PIC_SEG_OBJECT;
msgPicSegPaintObject
Sent by the PicSeg to objects in its database so they can draw themselves.
Takes P_PIC_SEG_PAINT_OBJECT, returns STATUS.

#define msgPicSegPaintObject MakeMsg(clsPicSeg, 46)

typedef struct {
    PIC_SEG_PAINT paint;
    RECT32 rectangle;
    OBJECT object;
    OBJECT picSeg;
    S32 reserved[6];
} PIC_SEG_PAINT_OBJECT, * P_PIC_SEG_PAINT_OBJECT;

msgPicSegDrawGrafic
Draws a grafic from the PicSeg.
Takes P_PIC_SEG_GRAFIC, returns STATUS.

#define msgPicSegDrawGrafic MakeMsg(clsPicSeg, 10)

typedef struct {
    OP_CODE opCode; // the type of grafic stored in pData
    P_UNKNOWN pData; // pointer to the grafic data
} PIC_SEG_GRAFIC, * P_PIC_SEG_GRAFIC;

The grafic opCode must be set to one of the opCode's defined by PicSeg's. Can be used for HitTest on a specific grafic. Never adds the grafic to the PicSeg. Responds to flags picSegDraw.

msgPicSegDrawGraficIndex
Sets the current grafic to index and draws it.
Takes S32 index, returns STATUS.

#define msgPicSegDrawGraficIndex MakeMsg(clsPicSeg, 11)

Can be used for HitTest on a specific grafic.

msgPicSegDrawGraficList
Draws all the grafics indexed by the list.
Takes P_PIC_SEG_LIST, returns STATUS.

#define msgPicSegDrawGraficList MakeMsg(clsPicSeg, 8)

typedef struct {
    S32 count; // number of grafic in list to draw
    P_S32 pIndex; // pointer to the list of grafics
} PIC_SEG_LIST, * P_PIC_SEG_LIST;

msgPicSegAddGrafic
Adds a grafic to the PicSeg and Draws the grafic.
Takes P_PIC_SEG_GRAFIC, returns STATUS.

#define msgPicSegAddGrafic MakeMsg(clsPicSeg, 9)

typedef struct {
    OP_CODE opCode; // the type of grafic stored in pData
    P_UNKNOWN pData; // pointer to the grafic data
} PIC_SEG_GRAFIC, * P_PIC_SEG_GRAFIC;
The graphic opCode must be set to one of the opCode's defined by PicSeg's. Responds to flags picSegAdd and picSegDraw.

**msgPicSegGetMetrics**

Passes back the metrics of the PicSeg.

Takes P_PIC_SEG_METRICS, returns STATUS.

```c
#define msgPicSegGetMetrics MakeMsg(clsPicSeg, 3)
```

**msgPicSegSetMetrics**

Sets the metrics of the PicSeg.

Takes P_PIC_SEG_METRICS, returns STATUS.

```c
#define msgPicSegSetMetrics MakeMsg(clsPicSeg, 4)
```

Comments

You cannot set picseg.numberGrafics.

**msgPicSegSetFlags**

Sets the PicSeg flags.

Takes S32, returns STATUS.

```c
#define msgPicSegSetFlags MakeMsg(clsPicSeg, 5)
```

**msgPicSegGetFlags**

Gets the PicSeg flags.

Takes P_S32, returns STATUS.

```c
#define msgPicSegGetFlags MakeMsg(clsPicSeg, 6)
```

---

**Hit Test**

**msgPicSegHitTest**

Performs a hit test on the PicSeg, passing back a single grafic index.

Takes P_PIC_SEG_HIT_LIST, returns STATUS.

```c
#define msgPicSegHitTest (clsPicSeg, 23)
```

**Arguments**

```c
typedef struct {
  RECT32 rect; // rectangle for the hit test
  S32 index; // in start grafic - out end grafic
} PIC_SEG_HIT_LIST,
```

**Comments**

index - in: First grafic to start hit test hit stops at grafic 0. Use picSegTopGrafic for starting at the top most grafic. out: The grafic hit if status is stsDcHitOn or stsDcHitIn. Otherwise 0.

STATUS return:

stsDcHitOn if the line intersects the hit rectangle

stsDcHitIn if the rectangle is inside a closed figure

stsDcHitOut if there was no hit

msgWinBeginPaint must be sent to the window first. msgWinEndPaint must be sent to the window after.
Editing the PicSeg Display List

**msgPicSegErase**
Deletes all graphics.
Takes nothing, returns STATUS.

#define msgPicSegErase MakeMsg(clsPicSeg, 24)

**msgPicSegDelete**
Deletes a graphic, takes a graphic Index. Sends msgDestroy to objects in the PicSeg.
Takes S32, returns STATUS.

#define msgPicSegDelete MakeMsg(clsPicSeg, 26)

**msgPicSegRemove**
Deletes a graphic, takes a graphic Index. Does not send msgDestroy to objects in the PicSeg.
Takes S32, returns STATUS.

#define msgPicSegRemove MakeMsg(clsPicSeg, 45)

**msgPicSegDelta**
Changes the current graphic.
Takes P_PIC_SEG_GRAFIC, returns STATUS.

#define msgPicSegDelta MakeMsg(clsPicSeg, 27)

typedef struct {
    OP_CODE    OpCode;
    P_UNKNOWN  pData;
} PIC_SEG_GRAFIC,
* P_PIC_SEG_GRAFIC;

**msgPicSegGetGrafic**
Gets the current graphic.
Takes P_PIC_SEG_GRAFIC, returns STATUS.

#define msgPicSegGetGrafic MakeMsg(clsPicSeg, 28)

typedef struct {
    OP_CODE    OpCode;
    P_UNKNOWN  pData;
} PIC_SEG_GRAFIC,
* P_PIC_SEG_GRAFIC;

Comments
Data must be freed by caller.

**msgPicSegSetCurrent**
Sets the current graphic index.
Takes S32, returns STATUS.

#define msgPicSegSetCurrent MakeMsg(clsPicSeg, 30)

Comments
Specifying picSegTopGrafic sets the current graphic to the last graphic in the list.
msgPicSegGetCurrent
Gets the index of the current graphic.
Takes P_S32, returns STATUS.
#define msgPicSegGetCurrent MakeMsg(clsPicSeg, 31)

msgPicSegGetCount
Gets the number of graphics in the PicSeg.
Takes P_S32, returns STATUS.
#define msgPicSegGetCount MakeMsg(clsPicSeg, 32)

msgPicSegMakeInvisible
Makes the given graphic invisible.
Takes S32, returns STATUS.
#define msgPicSegMakeInvisible MakeMsg(clsPicSeg, 33)
Comments
Changes the graphic's opCode by oring in OpCodeMaskInvisible.

msgPicSegMakeVisible
Makes the given graphic visible.
Takes S32, returns STATUS.
#define msgPicSegMakeVisible MakeMsg(clsPicSeg, 34)
Comments
Changes the graphic's opCode by masking out OpCodeMaskInvisible.

msgPicSegChangeOrder
Changes the order of the graphics in the display, Moving the current graphic to the given index.
Takes S32, returns STATUS.
#define msgPicSegChangeOrder MakeMsg(clsPicSeg, 35)
If the given index is less than the current index, then the graphics in between shift forward.
If the given index is greater than the current index, then the graphics in between shift backward.

msgPicSegSizeof
Returns the size of the (PIC_SEG_GRAFIC).pData in bytes.
Takes P_PIC_SEG_GRAFIC, returns S32.
#define msgPicSegSizeof MakeMsg(clsPicSeg, 39)

typedef struct {
  OP_CODE  OpCode;  // the type of graphic stored in pData
  P_UNKNOWN pData;  // pointer to the graphic data
} PIC_SEG_GRAFIC, * P_PIC_SEG_GRAFIC;
Messages Used For Move Copy

You can move and copy graphics in picture segments using the selection manager XFER mechanism type `xferPicSegObject`. The PicSeg is a data object and only helps define the method. The PicSeg itself does not have the selection. Usually the View, using the PicSeg as its data object, responds to move and copy messages. The selected View puts `xferPicSegObject` on the list when it receives `msgXferList`. With a match the receiving View creates a global heap PicSeg and sets up the XFER_OBJECT:

```c
XFER_OBJECT xferObject;
OBJECT picSeg;
MAT matrix;

memset(&xferObject, 0, sizeof(XFER_OBJECT));
xferObject.id = xferPicSegObject;
xferObject.receiver = self;

StsJmp(ObjectSendUpdate(msgXferGet, sel, &xferObject, 
 (U32)sizeof(XFER_OBJECT)), sts, error);

xferPicSeg = xferObject.uid;
ObjectCall(msgDcSetWindow, xferPicSeg, (P_ARGS)self);
ObjectCall(msgPicSegScaleUnits, xferPicSeg, (P_ARGS)psMetrics.units);
matrix.m31 = pTip->x - bounds.origin.x;
matrix.m32 = pTip->y - bounds.origin.y;
MatIdentity(matrix);
ObjectCall(msgPicSegTransform, xferPicSeg, &matrix);
ObjectCall(msgPicSegCopy, picSeg, (P_ARGS)xferPicSeg);
ObjectCall(msgDestroy, xferPicSeg, pNull);
```

The receiving View then ObjectSends `msgXferGet` to the selection. The selected View takes `msgXferGet` sets the xfer PicSeg's metrics to its own and puts the selected graphics into the global PicSeg. The receiving View must rebind the xfer PicSeg to a window using `msgDcSetWindow`. Then transform the xfer PicSeg with `msgPicSegScaleUnits` and `msgPicSegTransform`. The xferPicSeg is copied into the receiving View's PicSeg with `msgPicSegCopy`. The global PicSeg is then freed by the receiving View.

```c
#define tagPicSeg MakeTag(clsPicSeg, 0)
```

### msgPicSegScaleUnits

Scales all coordinates in the PicSeg from the old units to the new units, then sets the units of the PicSeg to the new units.

Takes MESSAGE, returns STATUS.

```c
#define msgPicSegScaleUnits MakeMsg(clsPicSeg, 36)
```

Comments

Valid arguments: `msgDcUnitsMetric`, `msgDcUnitsMil`, `msgDcUnitsPoints`, `msgDcUnitsTwips`, `msgDcUnitsPen`, `msgDcUnitsPen`, `msgDcUnitsDevice`, `msgDcUnitsLayout`.

Invalid arguments: `msgDcUnitsWorld`.

### msgPicSegTransform

Transforms all coordinates in the PicSeg database with the provided matrix.

Takes MAT, returns STATUS.

```c
#define msgPicSegTransform MakeMsg(clsPicSeg, 37)
```

Comments

Doesn't change line thickness, text size and rect radius. Thus this message is best used for Rotation and Translation only.
msgPicSegCopy
Copies the contents of the specified PicSeg to self.
Takes OBJECT, returns STATUS.

#define msgPicSegCopy
MakeMsg(clsPicSeg, 38)

Comments
Takes no account for units, scale, rotate and translate differences.
SYSFONT.H

This file provides font related definitions used by sysgraf.h.

## Overview

This file defines the values you give Sysgraf if you want to set the font parameters. See sysgraf.h, starting with msgSysDcFontId.

```c
#ifndef SYSFONT_INCLUDED
#define SYSFONT_INCLUDED

Font Attributes

#define sysDcGroupDefault 0 // also "system" font
#define sysDcGroupUserInput 1
#define sysDcGroupVenetian 2
#define sysDcGroupOldStyle 3
#define sysDcGroupTransitional 4
#define sysDcGroupModernRoman 5
#define sysDcGroupEgyptian 6
#define sysDcGroupScript 9
#define sysDcGroupModernScript 10
#define sysDcGroupTypewriter 11
#define sysDcSoftwareDefined 15 // subclass must draw glyphs
#define sysDcWeightLight 0
#define sysDcWeightNormal 1
#define sysDcWeightBold 2
#define sysDcWeightExtraBold 3
#define sysDcAspectCondensed 0
#define sysDcAspectNormal 1
#define sysDcAspectExtended 2
#define sysDcEncodeLinear 0
#define sysDcEncodeAdobeStandard 1
#define sysDcEncodeAdobeSymbol 2 // not implemented
#define sysDcEncodeIBM850 3 // MiniText and MiniNote expect this
#define sysDcEncodeGoSystem 4
#define sysDcEncodeHWX850 5
#define sysDcEncodeUnicode 6
#define sysDcAlignChrTop 0
#define sysDcAlignChrCenter 1
#define sysDcAlignChrBaseline 2
#define sysDcAlignChrDescender 3
```
Font Specification

To open a font a `SYSDC_FONT_SPEC` is used. This is a 32 bit number which may be interesting to file as a compact representation of a particular font specification (family, styles, etc., size is another matter).

It consists of two major fields, an "id", which is a 16-bit number that identifies a family, like Times Roman, or Futura.

This number can be derived from a four-byte string like "TR55" using the function `SysDcFontId` (defined in `sysgraf.h`). However, it is better to query the system as to the list of currently available fonts. Support for this exists in `tktable.h` (see `TkTableFillArrayWithFonts`) and `fontbox.h` (see `clsFontListBox`).

The second field contains attributes like boldness, italic, etc. Also, it contains a field called group. The group is a redundant encoding of information in the id. If the id, which identifies a specific font or font family, is not available, the group is used to locate a font with similar characteristics.

Another interesting field is encoding. This field serves to identify the "character set" of the bytes passed to `msgDcDrawText`.

Thus, if you file this 32-bit number along with a string of text the following will hold true:

1. The "interpretation" of the characters in the string is noted.
2. The "font family" is noted.
3. If the "font family" is not available the next time the string is displayed (perhaps on a different machine), then an acceptable substitute can be found.

```c
typedef struct
{
    U16    group : 4,     // use sysDcGroup...
    weight : 2,     // use sysDcWeight...
    aspect : 2,      // use sysDcAspect...
    italic : 1,      // use TRUE for italic
    monospaced : 1,  // use TRUE for monospaced
    encoding : 6;    // use sysDcEncode...
} SYSDC_FONT_ATTR, *P_SYSDC_FONT_ATTR;

typedef struct
{
    U16    id;     // for now 0 binds to "default" font
    SYSDC_FONT_ATTR attr;
} SYSDC_FONT_SPEC, *P_SYSDC_FONT_SPEC;

typedef struct
{
    SYSDC_FONT_SPEC    spec;     // actual
    CHAR name[80];
    COORD16 spaceWidth,
             underThickness,
             underPos,     // usually a small negative number
             xPos,
             ascenderPos,
             descenderPos; // usually a small negative number
    SIZE16 em;
    COORD16 maxY,
             minY;
} SYSDC_FONT_METRICS, *P_SYSDC_FONT_METRICS;

typedef struct
{
    COORD16 widths[256];     // per spec.encoding
} SYSDC_FONT_WIDTHS, *P_SYSDC_FONT_WIDTHS;
```
typedef struct {
    U16 alignChr; // use sysDcAlignChr...
    U16 underline; // use 0, 1, or 2
    U16 strikeout; // use 0 or 1
    P_CHAR pText;
    U16 lenText; // in (and out for measure)
    XY32 cp; // in and out, where to place string
    COORD32 stop; // used by msgDcMeasureText
    U16 spaceChar; // code for space, usually 32
    COORD16 spaceExtra; // added to width of space
    otherExtra; // added to width of every char
} SYSDC_TEXT_OUTPUT, *P_SYSDC_TEXT_OUTPUT;

typedef struct {
    XY16 min,
        max;
    COORD16 width;
} SYSDC_EXTENTS16, *P_SYSDC_EXTENTS16;

typedef struct {
    P_SYSDC_EXTENTS16 pExtents;
    P_CHAR pText;
    U16 len;
} SYSDC_CHAR_METRICS, *P_SYSDC_CHAR_METRICS;
SYSGRAF.H

This file provides the API for clsSysDrwCtx.

clsSysDrwCtx inherits from clsDrwCtx, an abstract class.

Defines the fundamental drawing services. An instance of clsSysDrwCtx, often called a "DC", is an object that is used to draw onto windows. After a DC is created, it is bound to a window (see msgDcSetWindow). After this step, drawing messages sent to the DC will result in drawing onto the bound window. While a DC may remain bound to a window forever, such drawing messages are only effective inside an "update episode" bracketed by msgWinBeginRepaint and msgWinEndRepaint.

There are a number of other DC messages that do not have to be sent inside an "update episode"; for instance msgDcLWCtoLUC_XY32. However, many of these messages implicitly require device or window metrics to produce the correct results. Thus, as a rule, a DC should be bound to a window before it is used.

Terminology:

DU4 -- Device Units, 4th Quadrant. A 4th quadrant coordinate system; device space, device units. This is used internally, but not seen by application software.

LWC -- Logical Window Coordinates. A 1st quadrant coordinate system. The lower-left-hand corner of the window is 0,0. The units are device pixels.

LUC -- Logical Unit Coordinates. A 1st quadrant coordinate system provided by the DC. The default units can be a real-world measure like points or mils; and they can be translated, rotated and scaled.

A number of font-related data structures are defined in sysfont.h.
Overview

Sysgraf (aka clsSysDrawCtxt aka ImagePoint) is the lowest level drawing interface PenPoint provides above the bit level. The division of labor here is that Windows worry about parceling out screen real-estate while Sysgraf worries about drawing on the screen. If you want to draw things in a window, you create a drawing context (an instance of clsSysDrawCtxt), bind it to the window you want to draw in (by sending msgDcSetWindow to the drawing context), and send messages to the drawing context.

If you plan to use a drawing context to render text, you should understand the use of msgDcMeasureText, which lets you determine how large a piece of text will be before you actually draw it. It is also important to know that although sysgraf allows you to set many different parameters, including font, rotation, line thickness, etc. you may only change these between drawing calls. That is, if you want to render plain text, a word in italics, and more plain text, you need to send three separate msgDcDrawText messages, changing to italics after the first one and back to normal after the second.

If you plan to use sysGraf at all, it will be well worth your while to browse all the messages below.

// Message numbers available: 7, 8, 9, 34, 35, 36, 37, 38; next up: 110

msgNew

Creates a system drawing context.

Takes P_SYSDC_NEW, returns STATUS. Category: class message.

typedef struct SYSDC_NEW ONLY {
    U32 reserved;
} SYSDC_NEW ONLY, *P_SYSDC_NEW ONLY;
#define sysdcNewFields
    objectNewFields
    SYSDC_NEW ONLY sysDc;

typedef struct {
    sysdcNewFields
} SYSDC_NEW, *P_SYSDC_NEW;

msgNewDefaults

Initializes the SYSDC_NEW structure to default values.

Takes P_SYSDC_NEW, returns STATUS. Category: class message.

typedef struct {
    sysdcNewFields
} SYSDC_NEW, *P_SYSDC_NEW;

sysDc.reserved = 0;

Binding to a Window

msgDcSetWindow

Binds a window to the receiver and returns the previously bound window.

Takes WIN, returns WIN.

#define msgDcSetWindow msgDrwCtxSetWindow

All output through the DC will now appear on this window. A DC must be bound to a window before most messages will work.

**msgDcGetWindow**

Gets the window to which the drawing context is bound.

Takes pNull, returns WIN.

#define msgDcGetWindow msgDrwCtxGetWindow

**Graphic State Control**

**msgDcInitialize**

Sets graphics state to initial values.

Takes pNull, returns stsOK.

#define msgDcInitialize MakeMsg(clsSysDrwCtx, 50)

The initial values are:

- units in (LUC) = msgDcUnitsPoints
- units out = msgDcUnitsDevice
- matrix = identity, 1st quadrant
- premultiply = FALSE
- clipping = none, except to window
- raster op = sysDcRopCopy
- drawing mode = sysDcDrawNormal | sysDcHoldDetail
- plane mask = see msgDcPlaneNormal
- line.cap = sysDcCapButt
- line.join = sysDcJoinMiter
- line.thickness = 1 unit (point)
- line.miterLimit = 10
- line.radius = 0
- foreground color = sysDcRGBBlack
- background color = sysDcRGBWhite
- fill pattern = sysDcPatBackground
- fill mode = even/odd (see sysDcWindingFill)
- line pattern = sysDcPatForeground
- logical font = default font, size is 1 unit (point)

**msgDcPush**

Gets the graphics state and stores it.

Takes P_SYSDC_STATE, returns stsOK.

#define msgDcPush MakeMsg(clsSysDrwCtx, 31)

While the names msgDcPush/msgDcPop imply a stack-like use for these messages (as is their intended application); this is not a requirement. There is no stack internal to the DC. State is copied in and out of the argument buffer.

One application is to pre-stage frequently needed combinations of state (fonts, colors, etc.) in an array of these buffers; and then pop them into a single DC as needed. This is more memory efficient than having several DC’s, and nearly as fast.
SYSDC_STATE is an opaque data type. There is no value in examining the bytes therein. It can be stored temporarily; but, it should not be filed, as it may change from release to release of the software.

**msgDcPop**
Sets the graphics state from one saved by msgDcPush.
Takes P_SYSDC_STATE, returns stsOK.

```c
#define msgDcPop MakeMsg(clsSysDrwCtx, 32)

typedef struct
{
    U8 state[448];
} SYSDC_STATE, *P_SYSDC_STATE;
```

**msgDcPushFont**
Gets the font state and stores it.
Takes P_SYSDC_FONT_STATE, returns stsOK.

```c
#define msgDcPushFont MakeMsg(clsSysDrwCtx, 51)

typedef struct
{
    U8 state[256];
} SYSDC_FONT_STATE, *P_SYSDC_FONT_STATE;
```

The same comments made under msgDcPush apply to msgDcPushFont.

**msgDcPopFont**
Sets the font state from one saved by msgDcPushFont.
Takes P_SYSDC_FONT_STATE, returns stsOK.

```c
#define msgDcPopFont MakeMsg(clsSysDrwCtx, 52)

typedef struct
{
    U8 state[256];
} SYSDC_FONT_STATE, *P_SYSDC_FONT_STATE;
```

**msgDcSetMode**
Sets the drawing mode and returns the old SYSDC_MODE.
Takes SYSDC_MODE, returns SYSDC_MODE.

```c
#define msgDcSetMode MakeMsg(clsSysDrwCtx, 2)

Enum16 (SYSDC_MODE)
{
    sysDcDrawNormal  = 0,
    sysDcDrawFast    = flag0,  // draw faster with gross loss of fidelity
    sysDcDrawDynamic = flag1,  // sets up XOR style drawing
    sysDcHoldDetail  = flag2,  // keeps lines from vanishing
    sysDcWindingFill = flag3,
    sysDcHitTest     = flag4,  // must set with msgDcHitTest
    sysDcAccumulate  = flag5,  // must set with msgDcAccumulateBounds
    sysDcHoldLine    = flag7,  // must set with msgDcHoldLine
    sysDcPreMultiply = flag6    // can set with msgDcSetPreMultiply
};
```
**msgDcGetMode**

Gets the drawing mode.

Takes pNull, returns SYSDC_MODE.

```
#define msgDcGetMode MakeMsg(clsSysDrwCtx, 65)
```

**msgDcSetPreMultiply**

Sets the pre-multiply state and returns the old state.

Takes BOOLEAN, returns BOOLEAN.

```
#define msgDcSetPreMultiply MakeMsg(clsSysDrwCtx, 96)
```

Comments

This affects the matrix arithmetic implicit in msgDcScale, msgDcRotate and msgDcTranslate. The default mode is post-multiply. The default for PostScript is pre-multiply; so when borrowing algorithms from PostScript sources this could be useful.

**msgDcSetRop**

Sets the raster op and returns the old rop.

Takes SYSDC_ROP, returns SYSDC_ROP.

```
#define msgDcSetRop Enum16(SYSDC_ROP)
{
    sysDcRopCOPY,
    sysDcRopAND,
    sysDcRopOR,
    sysDcRopXOR,
    sysDcRopNCOPY,
    sysDcRopNAND,
    sysDcRopNOR,
    sysDcRopNXOR
};
```

Arguments

Enum16(SYSDC_ROP)

Comments

Note that there are not many good reasons to be using this message; the results are rather device dependent. If you need to draw with an XOR raster op, use msgDcSetMode to set the sysDcDrawDynamic flag instead.

**msgDcPlaneNormal**

Sets the plane mask to the normal plane(s), returning the old mask.

Takes nothing, returns SYSDC_PLANE_MASK.

```
#define msgDcPlaneNormal MakeMsg(clsSysDrwCtx, 41)
typedef U16 SYSDC_PLANE_MASK;
```

**msgDcPlanePen**

Sets the plane mask to the plane(s) for pen ink, returning the old mask.

Takes nothing, returns SYSDC_PLANE_MASK.

```
#define msgDcPlanePen MakeMsg(clsSysDrwCtx, 42)
```

Comments

In most situations it is better to use clsTrack to draw on the pen plane(s). See track.h.
msgDcPlaneMask
Sets an arbitrary plane mask, returning the old mask.
Takes SYSDC_PLANE_MASK, returns SYSDC_PLANE_MASK.

#define msgDcPlaneMask MakeMsg(clsSysDrwCtx, 43)

Comments
This interface is NOT RECOMMENDED for application software. It is inherently non-portable.

msgDcGetLine
Gets all line attributes if pArgs is P_SYSDC_LINE. Returns line thickness.
Takes P_SYSDC_LINE, returns COORD16.

#define msgDcGetLine MakeMsg(clsSysDrwCtx, 62)

Comments
If P_SYSDC_LINE is pNull then only line thickness is returned.

msgDcSetLine
Sets all line attributes. Returns old line thickness.
Takes P_SYSDC_LINE, returns COORD16.

#define msgDcSetLine MakeMsg(clsSysDrwCtx, 4)

Arguments
Enum16 (SYSDC_CAP)
{ sysDcCapSquare = 0,
  sysDcCapButt = 1,
  sysDcCapRound = 2,
};
Enum16 (SYSDC_JOIN)
{ sysDcJoinMiter = 0,
  sysDcJoinBevel = 1,
  sysDcJoinRound = 2,
};
typedef struct
{
  SYSDC_CAP cap;
  SYSDC_JOIN join;
  COORD16 thickness;
  U16 miterLimit;  // Choose + number, 10 recommended.
  S16 radius;     // For rounded corner rectangles
                  // use + number or sysDcRadiusAuto.
                  // For square corner rectangles use 0.
                  // This number is in LUC.
} SYSDC_LINE, * P_SYSDC_LINE;

#define sysDcRadiusAuto (-1)

Comments
Both line thickness and the radius value for creating rounded corner rectangles are in LUC.

msgDcSetLineThickness
Sets line thickness to new value; returns old line thickness.
Takes COORD16, returns COORD16.

#define msgDcSetLineThickness MakeMsg(clsSysDrwCtx, 79)

Comments
This is the best message for quickly changing line thickness and restoring it back.
**msgDcHoldLine**

Turns hold line thickness mode on/off; returns old hold mode.

Takes BOOLEAN, returns BOOLEAN.

```c
#define msgDcHoldLine MakeMsg(clsSysDrwCtx, 63)
```

**msgDcHoldLine**(TRUE) causes the current line thickness to be made immune from the effects of scaling (msgDcScale, msgDcUnitsXXX). **msgDcHoldLine**(FALSE) will cancel hold mode.

**msgDcSetLine/Thickness** messages will cause the line thickness to change, but having changed, it will still be immune from the effects of scaling until hold mode is canceled.

The DC must be bound to a window when this message is sent.

---

**Device Independent Color**

```c
#define sysDcRGBTransparent ((U32)0)
#define sysDcRGBBlack (SysDcGrayRGB(0))
#define sysDcRGBGray66 (SysDcGrayRGB(85))
#define sysDcRGBGray33 (SysDcGrayRGB(170))
#define sysDcRGBWhite (SysDcGrayRGB(255))
```

typedef union
{
    U32 all;
    struct
    {
        U8 red,
        green,
        blue,
        transparency;
    } rgb;
} SYSDC_RGB, * P_SYSDC_RGB;

```c
#define SysDcGrayRGB(v) MakeU32(MakeU16(v,v),MakeU16(v,255))
```

These messages set and get the foreground and background colors by RGB specification. The "set" messages take an RGB specification (cast to a U32) and return stsOK.

The "get" messages store the current value into a U32 (or SYSDC_RGB) pointed to by pArgs.

The structure SYSDC_RGB is a union of the four r-g-b-t fields and a U32. This allows RGB values to be compared easily as U32 values. The transparency byte should always be 255 for opaque color. It can be 0 when setting the background color to transparent (in which case the red, green, blue values are not examined). Intermediate transparency values are not supported.

The macro SysDcGrayRGB takes a value between 0..255 and returns a U32 with the r-g-b bytes set to the value, and the transparency byte set to 255. The value 0 can be used for a pure transparent RGB.

The set messages find the closest matching color to the RGB specification; they do not create new colors. To create new colors see msgDcMixRGB (which is not implemented yet).

Unlike the palette oriented messages (msgDcSetForegroundColor, msgDcSetBackgroundColor) colors set using these RGB messages are portable across a variety of devices and are automatically retranslated when the DC is connected to a different device.
**msgDcSetForegroundRGB**
Sets foreground color using an RGB specification.
Takes U32, returns stsOK.

```c
#define msgDcSetForegroundRGB MakeMsg(clsSysDrwCtx, 75)
```

When using this interface, see the constants sysDcRGB... for the standard colors.

**msgDcSetBackgroundRGB**
Sets background color using an RGB specification.
Takes U32, returns stsOK.

```c
#define msgDcSetBackgroundRGB MakeMsg(clsSysDrwCtx, 76)
```

When using this interface, see the constants sysDcRGB... for the standard colors.

**msgDcInvertColors**
Swaps foreground and background colors.
Takes pNull, returns stsOK.

```c
#define msgDcInvertColors MakeMsg(clsSysDrwCtx, 64)
```

**msgDcGetForegroundRGB**
Returns foreground RGB value.
Takes P_U32 or P_SYSDC_RGB, returns stsOK.

```c
#define msgDcGetForegroundRGB MakeMsg(clsSysDrwCtx, 77)
```

**msgDcGetBackgroundRGB**
Returns background RGB value.
Takes P_U32 or P_SYSDC_RGB, returns stsOK.

```c
#define msgDcGetBackgroundRGB MakeMsg(clsSysDrwCtx, 78)
```

---

**Device Dependent Color**

```c
typedef U16 SYSDC_COLOR;
#define sysDcInkTransparent ((SYSDC_COLOR)0x8000)
#define sysDcInkBlack ( (SYSDC_COLOR)0x0000)
#define sysDcInkGray66 ( (SYSDC_COLOR)0x0001)
#define sysDcInkGray33 ( (SYSDC_COLOR)0x0002)
#define sysDcInkWhite ( (SYSDC_COLOR)0x0003)
```

**msgDcMatchRGB**
Returns palette entry that best matches an RGB.
Takes U32, returns SYSDC_COLOR.

```c
#define msgDcMatchRGB MakeMsg(clsSysDrwCtx, 10)
```

This interface is NOT RECOMMENDED for application software. Set colors directly using the msgDcSetForegroundRGB and msgDcSetBackgroundRGB messages.
msgDcSetForegroundColor
Sets foreground color using a hardware palette index, returning old color.
Takes SYSDC_COLOR, returns SYSDC_COLOR.
#define msgDcSetForegroundColor MakeMsg(clsSysDrwCtx, 5)
Comments
This interface is NOT RECOMMENDED for application software. Use msgDcSetForegroundRGB instead of this message.
When using this interface, see the constants sysDcInk... for predefined palette index values.

msgDcSetBackgroundColor
Sets background color using a hardware palette index, returning old color.
Takes SYSDC_COLOR, returns SYSDC_COLOR.
#define msgDcSetBackgroundColor MakeMsg(clsSysDrwCtx, 6)
Comments
This interface is NOT RECOMMENDED for application software. Use msgDcSetBackgroundRGB instead of this message.
When using this interface, see the constants sysDcInk... for predefined palette index values.

msgDcMixRGB
Programs a palette slot to a specific RGB.
Takes P_SYSDC_MIX_RGB, returns STATUS.
#define msgDcMixRGB MakeMsg(clsSysDrwCtx, 80)
Arguments
typedef struct
{
    SYSDC_COLOR slot;
    SYSDC_RGB spec;
} SYSDC_MIX_RGB, *P_SYSDC_MIX_RGB;
Comments
*** NOT IMPLEMENTED YET ***
This interface is NOT RECOMMENDED for application software. The type SYSDC_MIX_RGB is defined now to support msgWinDevMixRGB.

msgDcSetLinePat
Sets the line pattern; returns old value.
Takes SYSDC_PATTERN, returns SYSDC_PATTERN.
#define msgDcSetLinePat MakeMsg(clsSysDrwCtx, 11)
typedef U16 SYSDC_PATTERN;
#define sysDcPat75 ((SYSDC_PATTERN)1) // 75% fgnd 25% bgnd
#define sysDcPat50 ((SYSDC_PATTERN)2) // 50% fgnd 50% bgnd
#define sysDcPat25 ((SYSDC_PATTERN)3) // 25% fgnd 75% bgnd
#define sysDcPat12 ((SYSDC_PATTERN)4) // 12% fgnd 88% bgnd
#define sysDcPat6 ((SYSDC_PATTERN)5) // 6% fgnd 94% bgnd
#define sysDcPat3 ((SYSDC_PATTERN)6) // 3% fgnd 97% bgnd
#define sysDcPat2 ((SYSDC_PATTERN)7) // 2% fgnd 98% bgnd
#define sysDcPatLD50 ((SYSDC_PATTERN)8) // darkest left diagonal
#define sysDcPatLD37 ((SYSDC_PATTERN)9) // left diagonal
#define sysDcPatLD25 ((SYSDC_PATTERN)10) // left diagonal
#define sysDcPatLD12 ((SYSDC_PATTERN)11) // lightest left diagonal
#define sysDcPatRD50 ((SYSDC_PATTERN)12) // darkest right diagonal
The line pattern is used to draw lines around the edge of geometric figures when the line thickness is > 0.

When using this interface, see the constants `sysDcPat...` for predefined patterns.

### msgDcSetFillPat
Sets the fill pattern; returns old value.

Takes `SYSDC_PATTERN`, returns `SYSDC_PATTERN`.

```c
#define msgDcSetFillPat MakeMsg(clsSysDrwCtx, 12)
```

**Comments**
The fill pattern is used to draw the interior of closed geometric figures.

When using this interface, see the constants `sysDcPat...` for predefined patterns. `sysDcPatRandom` is unique for each window.

### msgDcGetLinePat
Gets the line pattern.

Takes `pNull`, returns `SYSDC_PATTERN`.

```c
#define msgDcGetLinePat MakeMsg(clsSysDrwCtx, 13)
```

**Comments**

*** NOT IMPLEMENTED YET ***

### msgDcGetFillPat
Gets the fill pattern.

Takes `pNull`, returns `SYSDC_PATTERN`.

```c
#define msgDcGetFillPat MakeMsg(clsSysDrwCtx, 14)
```

**Comments**

*** NOT IMPLEMENTED YET ***

### msgDcMixPattern
Mixes a custom pattern.

Takes `P_SYSDC_MIX_PAT`, returns `STATUS`.

```c
#define msgDcMixPattern MakeMsg(clsSysDrwCtx, 15)
```

**Arguments**

```c
typedef struct {
    SYSDC_PATTERN slot;
    U8 pattern[8];
} SYSDC_MIX_PAT, * P_SYSDC_MIX_PAT;
```

**Comments**

*** NOT IMPLEMENTED YET ***
**msgDcAlignPattern**
Sets the pattern alignment in LUC.
Takes P_XY32, returns STATUS.

```c
#define msgDcAlignPattern MakeMsg(clsSysDrwCtx, 16)
```

Comments
Can be used to keep pattern tiling aligned to a particular point in LUC when pixels are moved (msgWinCopyRect or wsGrow* flags). This is most commonly used to preserve pattern alignment during "scrolling" when parts of an image are copied pixels, and parts are newly painted pixels.

The default alignment is 0,0 in LUC. If the image is scrolled by msgDcTranslate then this message may not be necessary, as the alignment point will move in device space too.

If P_XY32 is pNull, default alignment is set to 0,0.

---

**LUC Space Transformations**

**msgDcUnitsMetric**
Sets input units to 0.01 mm.
Takes pNull, returns stsOK.

```c
#define msgDcUnitsMetric MakeMsg(clsSysDrwCtx, 17)
```

**msgDcUnitsMil**
Sets input units to 0.001 inch.
Takes pNull, returns stsOK.

```c
#define msgDcUnitsMil MakeMsg(clsSysDrwCtx, 18)
```

**msgDcUnitsPoints**
Sets input units to points (1/72 of an inch).
Takes pNull, returns stsOK.

```c
#define msgDcUnitsPoints MakeMsg(clsSysDrwCtx, 19)
```

**msgDcUnitsTwips**
Sets input units to 1/20 of a point.
Takes pNull, returns stsOK.

```c
#define msgDcUnitsTwips MakeMsg(clsSysDrwCtx, 20)
```

**msgDcUnitsPen**
Sets input units to pen sample units.
Takes pNull, returns stsOK.

```c
#define msgDcUnitsPen MakeMsg(clsSysDrwCtx, 71)
```
msgDcUnitsLayout
Sets input units to UI toolkit layout units.
Takes pNull, returns stsOK.

#define msgDcUnitsLayout MakeMsg(clsSysDrwCtx, 85)

Comments
Note that the scale this implicitly computes is a function of the current system font size. However, if the system font size changes after this message is sent, the scale is not "reliably" reevaluated (because of caching it may or may not be reevaluated). Thus, you may need to observe the SystemPreferences. For a small performance cost you can just send this message prior to each operation that is affected by unit scaling.

msgDcUnitsRules
Sets input units to the 'rules' associated with the system font.
Takes pNull, returns stsOK.

#define msgDcUnitsRules MakeMsg(clsSysDrwCtx, 3)

Comments
A 'rule' is 1/20 of the thickness of a line that aesthetically matches the weight of the system font, as specified by the font designer. Typically this will be the thickness of a single underline, and so a rule would be 1/20 of an underline.

Note that the scale this implicitly computes is a function of the current system font size. However, if the system font size changes after this message is sent, the scale is not "reliably" reevaluated (because of caching it may or may not be reevaluated). Thus, you may need to observe the SystemPreferences. For a small performance cost you can just send this message prior to each operation that is affected by unit scaling.

msgDcUnitsDevice
Sets input units to device pixels.
Takes pNull, returns stsOK.

#define msgDcUnitsDevice MakeMsg(clsSysDrwCtx, 21)

msgDcUnitsWorld
Sets input units to an arbitrary number of device pixels.
Takes pNull, returns stsOK.

#define msgDcUnitsWorld MakeMsg(clsSysDrwCtx, 25)

See Also
msgDcScaleWorld

msgDcUnitsOut
Sets output units produced by transformation of input units.
Takes MESSAGE, returns stsOK.

#define msgDcUnitsOut MakeMsg(clsSysDrwCtx, 70)

Comments
Takes one of:
msgDcUnitsMetric
In general, this message should not be used. Reverse transformations, from device units to other units can be made by using the msgDcLUCtoLWC... messages.

This interface can be used to change from one logical unit system to another. Since most such transformations are known in advance this is generally useless; however, transformation to and from pen units to a known unit system is the real purpose of this interface. For instance, pen units to mils can be used to store pen units in a device independent form. Pen units can thus remain device dependent.

This interface cannot change a graphic device unit into a device independent unit. To do this, units IN must be the chosen target unit (e.g. points), units OUT must be device, and the reverse transformation, msgDcLUTCtoLWC must be used.

---

**msgDcIdentity**

Sets LUC matrix to identity.

Takes pNull, returns stsOK.

```c
#define msgDcIdentity MakeMsg(clsSysDrwCtx, 22)
```

**msgDcRotate**

Rotates LUC matrix.

Takes ANGLE, returns stsOK.

```c
#define msgDcRotate MakeMsg(clsSysDrwCtx, 23)
```

**msgDcScale**

Scales LUC matrix.

Takes P_SCALE, returns stsOK.

```c
#define msgDcScale MakeMsg(clsSysDrwCtx, 26)
```

If P_SCALE is pNull then operation is same as msgDcIdentity.

**msgDcScaleWorld**

Creates a world scale of window width/height.

Takes P_SIZE32, returns stsOK.

```c
#define msgDcScaleWorld MakeMsg(clsSysDrwCtx, 61)
```

The window width/height is divided into SIZE32 width/height units. If the window is not physically square on the graphic device then the scale will not be uniform in x and y.

This message scales the LUC matrix. Typically, this matrix must be reset to identity (msgDcIdentity), and this message must be resent, whenever the window changes size (see msgWinSized for help).

The DC must be bound to a window when it receives this message.

**msgDcTranslate**

Translates LUC matrix.

Takes P_XY32, returns stsOK.

```c
#define msgDcTranslate MakeMsg(clsSysDrwCtx, 24)
```
Coordinate Conversion

These messages convert coordinates from LUC to LWC, or LWC to LUC. The DC must be bound to a window before it receives these messages.

**msgDcWCToLUC_XY32**
Transforms a point from window (device) space to logical space.
Takes P_XY32, returns stsOK.
#define msgDcWCToLUC_XY32 MakeMsg(clsSysDrwCtx, 27)
Comments
The DC transforms by:
LWC --> fractional LUC --> round to nearest integer LUC

**msgDcLUCtoLWC_XY32**
Transforms a point from logical space to window (device) space.
Takes P_XY32, returns stsOK.
#define msgDcLUCtoLWC_XY32 MakeMsg(clsSysDrwCtx, 39)
Comments
The DC transforms by:
LUC --> fractional LWC --> round to nearest integer LWC

**msgDcWCToLUC_SIZE32**
Transforms a size from window (device) space to logical space.
Takes P_SIZE32, returns stsOK.
#define msgDcWCToLUC_SIZE32 MakeMsg(clsSysDrwCtx, 44)
Comments
The DC transforms by:
LWC --> fractional LUC --> round to nearest integer LUC --> Abs()

**msgDcLUCtoLWC_SIZE32**
Transforms a size from logical space to window (device) space.
Takes P_SIZE32, returns stsOK.
#define msgDcLUCtoLWC_SIZE32 MakeMsg(clsSysDrwCtx, 45)
Comments
The DC transforms by:
LUC --> fractional LWC --> round to nearest integer LWC --> Abs()

**msgDcWCToLUC_RECT32**
Transforms a rectangle from window (device) space to logical space.
Takes P_RECT32, returns stsOK.
#define msgDcWCToLUC_RECT32 MakeMsg(clsSysDrwCtx, 46)
Comments
The DC transforms by:
1) converting the rectangle's origin and opposite corner (x+w, y+h) into fractional LUC,
2) rounding each point to the nearest integer coordinate, and
3) using those coordinates to determine a rectangle (whose width and height may be positive or negative).

msgDcLUCtoLWC_RECT32
Transforms a rectangle from logical space to window (device) space.
Takes _P_RECT32, returns stsOK.
#define msgDcLUCtoLWC_RECT32 MakeMsg(clsSysDrwCtx, 47)

Comments
The DC transforms by:
1) converting the rectangle’s origin and opposite corner (x+w, y+h) into fractional LWC,
2) rounding each point to the nearest integer coordinate, and
3) using those coordinates to determine a rectangle (whose width and height may be positive or negative).

msgDcGetMatrix
Returns the LWC matrix.
Takes _P_MAT, returns stsOK.
#define msgDcGetMatrix MakeMsg(clsSysDrwCtx, 40)

Comments
The DC must be bound to a window when this message is sent. This matrix transforms LUC to LWC (first quadrant, but device dependent units) coordinates. These coordinates are suitable for positioning windows.

msgDcGetMatrixLUC
Returns the LUC matrix.
Takes _P_MAT, returns stsOK.
#define msgDcGetMatrixLUC MakeMsg(clsSysDrwCtx, 87)

Comments
This matrix combines transformations to LUC space. It is identity unless msgDcScale, msgDcRotate, msgDcTranslate, msgDcScaleWorld, or msgDcSetMatrixLUC have been previously sent.
The default for these combinations is post-multiplication. See message msgDcSetPreMultiply for more on this subject.

msgDcSetMatrixLUC
Replaces the LUC matrix.
Takes _P_MAT, returns stsOK.
#define msgDcSetMatrixLUC MakeMsg(clsSysDrwCtx, 88)
**Clipping**

**msgDcClipRect**
Sets or clears clip rectangle.
Takes P_RECT32 or pNull, returns stsOK.

```c
#define msgDcClipRect MakeMsg(clsSysDrwCtx, 28)
```

Comments:
If P_RECT32 is pNull then operation is same as msgDcClipClear.

**msgDcClipClear**
Returns clipping to entire window.
Takes pNull, returns stsOK.

```c
#define msgDcClipClear MakeMsg(clsSysDrwCtx, 29)
```

**msgDcClipNull**
Suspends all clipping (except to raw device).
Takes pNull, returns stsOK.

```c
#define msgDcClipNull MakeMsg(clsSysDrwCtx, 30)
```

Comments:
The pen handler uses this to dribble ink anywhere on-screen (in the pen plane(s) only).
This interface is NOT RECOMMENDED for application software. It will protection fault if the caller does not have hardware privilege.

**Hit Detection**

**msgDcHitTest**
Turns hit testing on/off.
Takes P_RECT32 or pNull, returns stsOK.

```c
#define msgDcHitTest MakeMsg(clsSysDrwCtx, 66)
```

Comments:
To turn hit testing on supply a rectangle to test against. To turn hit testing off send pNull.

In general, drawing messages (msgDcDraw...) will return one of the status values stsDcHit... if hit testing is on:
- **stsDcHitOn** if the line intersects the hit rectangle
- **if the rectangle is inside a closed figure**
- **if there was no hit**

The following drawing messages implement hit testing:
- **msgDcDrawPolyline**

**Bounds Accumulation**
A region is available to accumulate the bounding rectangles of drawing operations.
**msgDcAccumulateBounds(pNull)** clears this region to empty and turns on accumulation. At this point, as in hit testing, drawing operations will not be output; rather, their bounding rectangles will be added to the accumulation. At any time the accumulation can be retrieved by using **msgDcGetBounds**. It can
be retrieved with another call to `{ msgDcAccumulateBounds(P_Rect32) }` which will both retrieve it, and
turn off accumulation so normal drawing can resume.

Normally, bounds are accumulated for the purpose of repainting part of a window.
`msgDcDirtyAccumulation` can be used to add the accumulation directly to the dirty region of the
current window. This is more efficient than getting the bounds rectangle and then sending
`msgWinDirtyRect`.

Bounds accumulation occurs in DU4 space; while the bounds rectangle is returned in LUC, it always
represents a rectangle in DU4. Thus, drawing which is clipped because of windowing, or because it falls
off the edges of the device, is not accumulated.

The bounds accumulation region itself is not part of the logical state, although the flag that determines
whether drawing operations accumulate or draw is part of the logical state. Thus, while calls to push and
pop the state may turn accumulation on or off, there are not separate copies of the accumulation region
itself in the state.

Bounds accumulation and hit testing cannot be performed at the same time. If, through program error,
both modes are enabled, bounds accumulation will take priority.

Neither bounds accumulation or hit testing should be used during repainting initiated by the window
manager sending `msgWinRepaint`. Rather, they should be used within a `msgWinBeginPaint`
`msgWinEndPaint` bracket.

---

**msgDcAccumulateBounds**

Starts or stops bounds accumulation; retrieve bounds.

Takes P_RECT or pNull, returns stsOK.

```c
#define msgDcAccumulateBounds MakeMsg(clsSysDrwCtx, 81)
```

Comments

If pArgs is pNull, clears current accumulation and turns accumulation on. If pArgs is P_RECT32, returns
accumulated bounds, and turns bounds accumulation off.

The DC computes the LUC rectangle so that it:

1) mathematically includes all of the accumulated pixels, and
2) has non-negative width and height.

---

**msgDcDirtyAccumulation**

Marks accumulation dirty; turns accumulation off; retrieves bounds.

Takes P_RECT32 or pNull, returns stsOK.

```c
#define msgDcDirtyAccumulation MakeMsg(clsSysDrwCtx, 82)
```

Comments

Adds current bounds accumulation directly to the dirty region of the current window; then clears
current bounds accumulation and turns accumulation off. If pArgs is P_RECT32, returns accumulated
bounds as in `msgDcAccumulateBounds`.

---

**msgDcGetBounds**

Retrieves current accumulation bounds rectangle.

Takes P_RECT32, returns stsOK.

```c
#define msgDcGetBounds MakeMsg(clsSysDrwCtx, 83)
```
typedef struct
{
  U16  count;  // number of points in points array
  P_XY32  points;  // pointer to array of at least 2 points
} SYSDC_POLYGON, *P_SYSDC_POLYGON,
SYSDC_POLYLINE, *P_SYSDC_POLYLINE;

typedef struct
{
  RECT32  bounds;
  XY32    rays[2];
} SYSDC_ARC_RAYS, *P_SYSDC_ARC_RAYS;

Does not clear accumulation or turn accumulation off. The DC computes the LUC rectangle as in msgDcAccumulateBounds.

Open Figures

msgDcDrawPolyline
Draws a line; needs at least 2 points. Returns either hit test or stsOK.
Takes P_SYSDC_POLYLINE, returns STATUS.
#define msgDcDrawPolyline  MakeMsg(clsSysDrwCtx, 100)

Return Value
stsDcHitOn

msgDcDrawBezier
Draws a Bezier curve; needs exactly 4 points.
Takes P_XY32 (array of 4), returns STATUS.
#define msgDcDrawBezier  MakeMsg(clsSysDrwCtx, 104)

Comments
Returns either hit test or stsOK.

Return Value
stsDcHitOn

msgDcDrawArcRays
Draws an arc using the two rays method. Returns either hit test or stsOK.
Takes P_SYSDC_ARC_RAYS, returns STATUS.
#define msgDcDrawArcRays  MakeMsg(clsSysDrwCtx, 105)

Message
typedef struct
{
  RECT32  bounds;
  XY32    rays[2];
} SYSDC_ARC_RAYS, *P_SYSDC_ARC_RAYS;

Return Value
stsDcHitOn
Closed Figures

msgDcSetPixel
Sets a pixel with a value.
Takes P_SYSDC_PIXEL, returns STATUS.
#define msgDcSetPixel MakeMsg(clsSysDrwCtx, 108)
Comments
If rgb is true, the color is interpreted as an RGB value; if not, color.all will be interpreted as a SYSDC_COLOR (a hardware palette index). If rgb is used then the transparency byte must be 255 (opaque) or the drawing will not take place.

msgDcGetPixel
 Gets a pixel value.
Takes P_SYSDC_PIXEL, returns STATUS.
#define msgDcGetPixel MakeMsg(clsSysDrwCtx, 109)
Arguments
typedef struct
 { BOOLEAN rgb;
   SYSDC_RGB color;
   XY32 xy;
 } SYSDC_PIXEL, *P_SYSDC_PIXEL;
Comments
If rgb is TRUE the color is returned as an RGB value; if not color.all will be a small number which should be interpreted as a SYSDC_COLOR (a hardware palette index). If rgb is used the transparency byte will always be returned as 255 (opaque).

msgDcDrawRectangle
Draws a rectangle. Returns either hit test or stsOK.
Takes P_RECT32, returns STATUS.
#define msgDcDrawRectangle MakeMsg(clsSysDrwCtx, 101)
Return Value
stsDcHitOn

msgDcDrawEllipse
Draws an ellipse. Returns either hit test or stsOK.
Takes P_RECT32, returns STATUS.
#define msgDcDrawEllipse MakeMsg(clsSysDrwCtx, 102)
Return Value
stsDcHitOn

msgDcDrawPolygon
Draws a polygon. Returns either hit test or stsOK.
Takes P_SYSDC_POLYGON, returns STATUS.
#define msgDcDrawPolygon MakeMsg(clsSysDrwCtx, 103)
Arguments
typedef struct
 { U16 count; // number of points in points array
   P_XY32 points; // pointer to array of at least 2 points
 } SYSDC_POLYGON, *P_SYSDC_POLYGON,
Return Value
stsDcHitOn
**msgDcDrawSectorRays**

Draws a sector (pie wedge) using the two rays method.

Takes P_SYSDC_ARC_RAYS, returns STATUS.

```c
#define msgDcDrawSectorRays MakeMsg(clsSysDrwCtx, 106)
```

**Message Arguments**

```c
typedef struct {
    RECT32 bounds;
    XY32 rays[2];
} SYSDC_ARC_RAYS,
* P_SYSDC_ARC_RAYS;
```

**Comments**

Returns either hit test or stsOK.

**Return Value**

stsDcHitOn

---

**msgDcDrawChordRays**

Draws a chord using the two rays method. Returns either hit test or stsOK.

Takes P_SYSDC_ARC_RAYS, returns STATUS.

```c
#define msgDcDrawChordRays MakeMsg(clsSysDrwCtx, 107)
```

**Message Arguments**

```c
typedef struct {
    RECT32 bounds;
    XY32 rays[2];
} SYSDC_ARC_RAYS,
* P_SYSDC_ARC_RAYS;
```

**Return Value**

stsDcHitOn

---

**msgDcFillWindow**

Frames window with a line and fills the window.

Takes pNull, returns stsOK.

```c
#define msgDcFillWindow MakeMsg(clsSysDrwCtx, 33)
```

**Comments**

Draws a rectangle exactly the size of the window. All line, fill and color attributes apply.

When drawing a rectangle, the first pixel of line thickness is painted "inside" the rectangle, the second "outside", and it alternates from there. Therefore, lines > 1 pixel thick will have 1/2 their thickness fall outside the window when using this message. If that drawing is clipped (as it normally is) the line will appear 1/2 as thick as one would expect.

---

**Sampled Image Processing**

**msgDcDrawImage**

Draws an image from sampled image data. The image will be scaled, rotated, translated, according to the current state.

Takes P_SYSDC_IMAGE_INFO, returns STATUS.

```c
#define msgDcDrawImage MakeMsg(clsSysDrwCtx, 48)
#define msgDcGetSrcRow MakeMsg(clsSysDrwCtx, 49)
```
Sampled Image Processing

Arguments

```c
Enum16(SYSDC_IMAGE_FLAGS)
{
    sysDcImageNoFilter = 0, // fast but poor fidelity
    sysDcImageLoFilter = flag0, // use this for most image data
    sysDcImageHiFilter = flag1, // use this for color image data
    sysDcImageRunLength = flag2, // run length encoded
    sysDcImage1BPS = flag3, // 1 bit per sample
    sysDcImage2BPS = flag2|flag3, // 2 bits per sample
    sysDcImage4BPS = flag4, // 4 bits per sample
    sysDcImage8BPS = flag0, // 8 bits per sample
    sysDcImageCallBack = flag8, // callBack is a P_SYSDC_GETROW function
    sysDcImageCallObject = flag9, // callBack is a OBJECT
    sysDcImageFillWindow = flag10, // dstRect not provided
    sysDcImagePolarityFalse = flag11 // paint '0' w/ background color
                                       // else paint '1' w/ foreground color
};
```

typedef struct SYSDC_IMAGE_INFO * P_SYSDC_IMAGE_INFO;
typedef BOOLEAN FunctionPtr(P_SYSDC_GETROW) (P_SYSDC_IMAGE_INFO pCtx);
typedef struct SYSDC_IMAGE_INFO
{
    RECT32 dstRect; // destination size and position
    SIZE16 srcSize; // # of source samples
    SYSDC_IMAGE_FLAGS flags;
    union
    {
        P_SYSDC_GETROW function;
        OBJECT object;
    }
    callBack;
    P_UNKNOWN pBuffer;
    P_UNKNOWN pClientData;
    P_UNKNOWN reserved[3];
} SYSDC_IMAGE_INFO;
```

Comments

This message is similar to the PostScript image operator. Sample data, in the form of numbers ranging from 0..max are interpreted as grey values. 0 is black and max is white. The value of max is determined by the size of the input numbers, which can be 1, 2, 4 or 8 bits.

Because the sample data may be large, in a file, or incrementally decompressed, this message can work with a callback strategy. The callback can be either a function (flag sysDcImageCallBack), or an object (flag sysDcImageCallObject) to which msgDcGetSrcRow is sent. In both cases the argument is the same pointer to a SYSDC_IMAGE_INFO that is the argument to msgDcDrawImage itself. To support client context during the callback, the field pClientData is provided, for the callback to use as necessary.

The source sample data width and height is described by srcSize. The rectangle at the destination, which will be filled by the image, is dstRect; or optionally, the flag sysDcImageFillWindow can be used to fill the entire window.

During the callback, pBuffer will point to a buffer that needs to be filled with srcSize.w samples. If pBuffer is pNull, it means the operator is skipping a row (because of clipping perhaps). Thus, no samples need to be provided, but the context must be "advanced" to skip the row. If anything goes wrong, the callback can return FALSE (for a function), or a bad status code (for an object) to terminate the drawing.

If callback is not used at all, then pBuffer should be set by the caller to point to all of the sample data at the outset.

The result of the drawing is that dstRect is filled with an image. Since dstRect is in LUC space, its size and location is the same as if it were drawn with msgDcDrawRectangle.
Before using this interface to display "tiff" images, investigate clsTiff (tiff.h) for a much higher level service.

Return Value

stsDcHitOn

**msgDcDrawImageMask**

Draws a mask from sampled image data. Similar to **msgDcDrawImage**.

Takes P_SYSDC_IMAGE_INFO, returns STATUS.

```c
#define msgDcDrawImageMask MakeMsg(clsSysDrwCtx, 97)
```

```c
typedef struct SYSDC_IMAGE_INFO {
    RECT32 dstRect; // destination size and position
    SIZE16 srcSize; // # of source samples
    SYSDC_IMAGE_FLAGS flags;
    union {
        P_SYSDC_GETROW function;
        OBJECT object;
    }
    callBack;
    P_UNKNOWN pBuffer;
    P_UNKNOWN pClientData;
    P_UNKNOWN reserved[3];
} SYSDC_IMAGE_INFO;
```

Comments

This message is similar to the PostScript imagemask operator and **msgDcDrawImage**. The input parameters are the same as for **msgDcDrawImage** with the addition of one flag, **sysDcImagePolarityFalse**, which would normally not be set (TRUE). However, the results of this message are visually different than **msgDcDrawImage**.

**msgDcDrawImage** reduces the input data to grey values and paints an opaque parallelogram. The values of the current foreground and background colors have no effect on the behavior of **msgDcDrawImage**.

**msgDcDrawImageMask** reduces the input data to the values '0' and '1'. The default behavior is for the '1' values to be painted with the current foreground color; the '0' values are not painted at all.

This behavior can be reversed by setting the **sysDcImagePolarityFalse** flag. In this case the '0' values are painted with the current background color and the '1' values are not painted.

Return Value

stsDcHitOn

**msgDcCacheImage**

Passes back a cached image in **pCache**, given a sampled image and an optional mask.

Takes P_SYSDC_CACHE_IMAGE, returns STATUS.

```c
#define msgDcCacheImage MakeMsg(clsSysDrwCtx, 91)
```

```c
typedef struct {
    SYSDC_IMAGE_INFO image[2]; // in = [0] is image, [1] is mask
    BOOLEAN hasMask; // in = if this is true
    XY16 hotSpot; // in
    P_UNKNOWN pCache; // out = cache (segment)
} SYSDC_CACHE_IMAGE, *P_SYSDC_CACHE_IMAGE;
```

Comments

A "cached image" is a segment of memory (**pCache**) that contains the device-dependent (pixelmap) representation of a sampled image (see **msgDcDrawImage**), and optionally a mask.

This operator is intended to be used for cursors and icons. It currently does not work on printer devices.
Once cached, the image can be drawn (with hotspot adjustment) using `msgDcCopyImage`.

Because of its device dependent representation, a cached image becomes obsolete when the device rotation changes (landscape vs. portrait). Thus, you may need to observe the `SystemPreferences` and rebuild the cache when appropriate. When you are finished with the cached image you should free it with `OSHeapBlockFree`.

### msgDcCopyImage

Copies a cached image to the bound window.

Takes `P_SYSDC_COPY_IMAGE`, returns `STATUS`.

```c
#define msgDcCopyImage MakeMsg(clsSysDrwCtx, 92)
```

**Arguments**

```c
typedef struct {
    XY32 xy;              // in = destination location
    P_UNKNOWN pCache;     // in
    SYSDC_COPY_IMAGE, *P_SYSDC_COPY_IMAGE;
} SYSDC_COPY_IMAGE;
```

**Comments**

The image is copied, such that the hotspot aligns on `xy`.

### Fonts

Some of the data structures used in the font interface are declared in `sysfont.h`.

All font metric information is currently computed in LUC space. However, because all the relevant numbers, except for scaling, are integers, significant round-off error can occur. For instance, at 10 or 12 points, a small feature, like x-height, will be a very small number. If LUC is relatively coarse, the error may be significant. The same holds true for the quality of inter-character spacing. Each character within a string of text output is positioned in LUC space, and the positioning will be no more accurate than the granularity of LUC. In general, the use of TWIPS units, or even finer units, is recommended if high quality text at small point sizes is required. Note that this may change in the future--read on.

While this approach produces less than perfect results on screen, it does have the benefit of maintaining very close correspondence between screen and printer; such that the same code can be used for both with no significant variance. In general, each character will be positioned to within one LUC unit or one device pixel (whichever is larger), of accuracy.

In future versions, the text measurement messages may change so that they advance character by character in an internal coordinate space that doesn't match LUC. This would allow accurate intercharacter spacing regardless of the granularity of LUC.

### SysDcFontId

Takes a 4 byte string font description and returns a 16-bit font id number.

Returns `U16`.

```c
U16 EXPORTED SysDcFontId(P_CHAR strId)    // In: a string like "HE55"
```
**SysDcFontString**

Takes a 16-bit font id number and passes back a 4 char string.

Returns void.

```c
void EXPORTED SysDcFontString(U16, // In: a font id number
   P_CHAR // Out: a string like "HE55"
);
```

Comments

The string buffer should be at least 5 bytes long.

---

**msgDcOpenFont**

Opens a font.

Takes P_SYSDC_FONT_SPEC or pNull, returns stsOK.

```c
#define msgDcOpenFont MakeMsg(clsSysDrwCtx, 53)
```

Comments

Specifying pNull will open default font.

---

**msgDcScaleFont**

Scales font matrix.

Takes P_SCALE or pNull, returns stsOK.

```c
#define msgDcScaleFont MakeMsg(clsSysDrwCtx, 54)
```

Comments

If argument is pNull then behavior is same as msgDcIdentityFont. The default size of a newly opened font is 1 unit (LUC). Use this message to scale to the desired size.

Note that this scaling is cumulative (multiplicative). A scale of 10,10 followed by a scale of 12,12 will result in a scale of 120,120. When "switching to absolute sizes" a msgDcIdentityFont will usually be needed.

Note also that font scale is affected by the overall scale established by the msgDcUnits... messages, and msgDcScale.

---

**msgDcIdentityFont**

Sets font matrix scale to default of 1 unit (LUC).

Takes pNull, returns stsOK.

```c
#define msgDcIdentityFont MakeMsg(clsSysDrwCtx, 72)
```

---

**msgDcDrawText**

Draws text in the current font.

Takes P_SYSDC_TEXT_OUTPUT, returns stsOK.

```c
#define msgDcDrawText MakeMsg(clsSysDrwCtx, 55)
```

---

**msgDcMeasureText**

Computes size of text and advances pArgs->cp accordingly.

Takes P_SYSDC_TEXT_OUTPUT, returns stsOK.

```c
#define msgDcMeasureText MakeMsg(clsSysDrwCtx, 57)
```
Measuring stops when stop is exceeded. Stop will normally be a "right margin". This is used to measure out lines of text from a large buffer. Upon return lenText will be the number of characters that "fit". This information can then be used to break and justify lines.

To simply measure an entire string, set cp.x = 0 and stop = maxS32; upon return cp.x will be the length of the string in LUC.

During measuring, other parameters, like spaceExtra and otherExtra are significant.

**msgDcDrawTextRun**
Like msgDcDrawText, except run spacing applies.
Takes P_SYSDC_TEXT_OUTPUT, returns stsOK.

```c
#define msgDcDrawTextRun MakeMsg(clsSysDrwCtx, 73)
```

Comments
Run spacing is important when spaceExtra and otherExtra contain non-zero values; especially when underlining. For instance, if otherExtra is 10, then 10 units will be added after the last character when using run spacing. It would not be added when using the normal DrawText message. This affects the cp.x value returned, and is visually significant when underlining or strikethrough are performed (the 10 units would have the lines or not). It will also affect the result when centering or right-justifying text.

**msgDcMeasureTextRun**
Like msgDcMeasureText, except run spacing applies.
Takes P_SYSDC_TEXT_OUTPUT, returns stsOK.

```c
#define msgDcMeasureTextRun MakeMsg(clsSysDrwCtx, 74)
```

Comments
See comments about "run spacing" under msgDcDrawTextRun.

**msgDcDrawTextDebug**
Like msgDcDrawText, except text is drawn with debugging lines around each char.
Takes P_SYSDC_TEXT_OUTPUT, returns stsOK.

```c
#define msgDcDrawTextDebug MakeMsg(clsSysDrwCtx, 56)
```

Comments
This function may not work unless the debugging version of win.dll is being used.

**msgDcPreloadText**
Preloads pText into cache.
Takes P_SYSDC_TEXT_OUTPUT, returns stsOK.

```c
#define msgDcPreloadText MakeMsg(clsSysDrwCtx, 58)
```

Comments
If pArgs is pNull or pArgs->pText is pNull a default set is preloaded.
This message causes the characters to be rasterized into the font cache so that during a subsequent msgDcDrawText there are no hesitations during a cache miss. This is not normally necessary, but might be useful in a "slide show" application.

**msgDcGetCharMetrics**
Gets char metrics information for a string.
Takes P_SYSDC_CHAR_METRICS, returns stsOK.

```c
#define msgDcGetCharMetrics MakeMsg(clsSysDrwCtx, 84)
```
These character metrics are more precise in some ways than those returned by msgDcGetFontMetrics. For instance, the width of a character is a purely logical value. The character image may extend past its width to the right, and may extend to the left past its "left edge". Similarly, some characters will extend above the "ascender" line or below the "descender" lines (which are just imaginary lines that guide the letterforms in general).

For each character in the string, the information returned is the minimum and maximum x and y coordinates found in that glyph, as if the glyph were drawn at 0,0. There are no "string semantics" to the "string" (x is not accumulating left to right); rather, this is similar to a 'width table' except values for a specific string only are returned.

See the caveat below for msgDcGetFontWidths.

**msgDcGetFontMetrics**

Gets the font metrics for the current font.

Takes P_SYSDC_FONT_METRICS, returns stsOK.

```c
#define msgDcGetFontMetrics MakeMsg(clsSysDrwCtx, 59)
```

**msgDcGetFontWidths**

Gets the font width table of the current font.

Takes P_SYSDC_FONT_WIDTHS, returns stsOK.

```c
#define msgDcGetFontWidths MakeMsg(clsSysDrwCtx, 60)
```

**See the caveat below for msgDcGetFontWidths.**

This width table is an array of 255 COORD16 values. Try to use the msgDcMeasureText interface instead; as width tables become less practical as character sets get larger and larger (e.g., Kanji).

Another important reason to use msgDcMeasureText instead is that the measureText/drawText interfaces may change in the future to advance character by character in an internal coordinate space that doesn't match LUC. This would allow accurate intercharacter spacing regardless of the granularity of LUC. In short, we do not guarantee that merely adding up widths obtained by msgDcGetFontWidths would match the results of using msgDcMeasureText. msgDcMeasureText always represents the correct behavior, while msgDcGetFontWidths should be thought of as an approximation.

**Special Messages**

**msgDcDrawPageTurn**

Draws a page turn effect over the bound window.

Takes P_SYSDC_PAGE_TURN, returns stsOK.

```c
#define msgDcDrawPageTurn MakeMsg(clsSysDrwCtx, 86)
```

**Arguments**

```c
typedef struct
{  
P_RECT32 pBounds;  // may be pNull for entire window
  U16 fxNo,       // must be 0
  iterations;
  BOOLEAN fxFwd,
  landscape;
} SYSDC_PAGE_TURN, * P_SYSDC_PAGE_TURN;
```
**msgDsCopyPixels**
Copies pixels from *srcWindow* to the bound window.
Takes P_SYSDC_PIXELS, returns stsOK.

```c
#define msgDsCopyPixels MakeMsg(clsSysDrwCtx, 89)
```

**Arguments**
typedef struct {
  OBJECT      srcWindow;    // in=on a clsImgDev
  P_RECT32    pBounds;      // in=may be pNull for entire window
  XY32        xy;           // in=destination location
  BOOLEAN     dstDirty;     // in=
} SYSDC_PIXELS, *P_SYSDC_PIXELS;

**Comments**
The rectangle *pBounds* on *srcWindow* is copied to the destination (bound) window at location *xy*. If *dstDirty* is TRUE, "dirty" pixels from the *srcWindow* cause the corresponding pixels on the destination window to be marked dirty (however, the dirty pixels ARE copied anyway).

The *srcWindow* must be on an "image device". See clsImgDev.

**Return Value**
stsTruncatedData source rectangle not entirely on the window device; some dest pixels not affected.

---

**msgDsDrawPixels**
Draws foreground and background colors in the bound window's pixels using *srcWindow*'s pixel values as a stencil.
Takes P_SYSDC_PIXELS, returns stsOK.

```c
#define msgDsDrawPixels MakeMsg(clsSysDrwCtx, 90)
```

**Arguments**
typedef struct {
  OBJECT      srcWindow;    // in=on a clsImgDev
  P_RECT32    pBounds;      // in=may be pNull for entire window
  XY32        xy;           // in=destination location
  BOOLEAN     dstDirty;     // in=
} SYSDC_PIXELS, *P_SYSDC_PIXELS;

**Comments**
Like msgDsCopyPixels except the source clsImgDev window must be only 1 plane and the dstDirty processing is not performed.

'1' pixels from the source are drawn with the foreground color, and '0' pixels with the background color.

**Return Value**
stsTruncatedData source rectangle not entirely on the window device; some dest pixels not affected.

---

**msgDsScreenShot**
Captures a screen image to a "tiff" file.
Takes P_SYSDC_SCREEN_SHOT, returns stsOK.

```c
#define msgDsScreenShot MakeMsg(clsSysDrwCtx, 67)
```

**Arguments**
typedef struct {
  P_RECT32    pBounds;
  P_CHAR      pFileName;
} SYSDC_SCREEN_SHOT, *P_SYSDC_SCREEN_SHOT;

**Comments**
*pBounds* can be a rectangle that is off the window and those pixels will be captured too (actually, wraparound will occur); no clipping is implied by the window, only the relative positioning of *pBounds*. 
If \texttt{pBounds} is \texttt{pNull} the whole window will be captured. If \texttt{LUC} are rotated non-modulo 90 degrees an upright rectangle bounding \texttt{pBounds} will be captured.

If you are just capturing screen shots for documentation, try using the \texttt{SShot} utility application first.

**Messages from other classes**

\texttt{msgDrwCtxSetWindow}

Binds a window to the receiver and returns the previously bound window.

Takes \texttt{WIN}, returns \texttt{WIN}.

**Comments**

All output through the DC will now appear on this window. A DC must be bound to a window before most messages will work.

\texttt{msgDrwCtxGetWindow}

Gets the window to which the drawing context is bound.

Takes \texttt{pNull}, returns \texttt{WIN}.

\texttt{msgWinDirtyRect}

Marks all or part of a window dirty.

Takes \texttt{P RECT32} or \texttt{pNull}, returns \texttt{STATUS}.

**Comments**

If \texttt{P ARGS} is not null, the DC will transform the rectangle into LWC and pass the message on to the DC's bound window. The DC computes the LWC rectangle in the same manner as \texttt{msgDcAccumulateBounds}, i.e. so that it:

1) mathematically includes the entire LUC rectangle, and

2) has non-negative width and height.

If the \texttt{P ARGS} is null, the DC will just pass the message on to the DC's bound window.

\texttt{msgWinBeginPaint}

Sets up window for painting on its visible region.

Takes \texttt{P RECT32} or \texttt{pNull}, returns \texttt{STATUS}.

**Comments**

The \texttt{P ARGS} is handled the same as in \texttt{msgWinDirtyRect}.

\texttt{msgWinBeginRepaint}

Sets up window for painting on "dirty" region.

Takes \texttt{P RECT32} or \texttt{pNull}, returns \texttt{STATUS}.

**Comments**

The DC will pass the message on to the DC's bound window and then, if \texttt{P ARGS} is not null, transform the out parameter rectangle from LWC to LUC. The DC computes the rectangle so that it:

1) mathematically includes the entire LUC rectangle, and

2) has width and height \(\geq 1\).
**msgWinBeginPaint**
Sets up window for painting on its visible region.
Takes P_RECT32 or pNull, returns STATUS.

Comments
The P_ARGS is handled the same as in msgWinBeginRepaint.

**msgWinDelta**
Moves and/or resizes a window. pArgs->bounds should be the newly desired bounds (size AND position).
Takes P_WIN_METRICS, returns STATUS.

Comments
The DC will transform pArgs->bounds into LWC and pass the message on to the DC's bound window. The DC transforms:
the in parameter bounds in the same manner as msgDcLUCtoLWC_RECT32, and
the out parameter bounds in the same manner as msgDcLWCtoLUC_RECT32.

**msgWinTransformBounds**
Transforms bounds from receiver's to another window's LWC.
Takes P_WIN_METRICS, returns STATUS.

Comments
The P_ARGS is handled the same as in msgWinDelta.

**msgWinHitDetect**
Locates the window "under" a point.
Takes P_WIN_METRICS, returns STATUS.

Comments
The DC will pass the message on to the DC's bound window. The bounds passed along to the window will be a copy of pArgs->bounds that the DC has transformed into LWC as in msgDcLUCtoLWC_RECT32.

**msgWinGetMetrics**
Gets full window metrics.
Takes P_WIN_METRICS, returns STATUS.

Comments
The DC will pass the message on to the DC's bound window, and then return a pArgs->bounds that is transformed as in msgDcLWCtoLUC_RECT32.

**msgWinCopyRect**
Copies pixels within a window.
Takes P_WIN_COPY_RECT, returns STATUS.

Comments
The DC will first transform the pArgs->srcRect from LUC to LWC as in msgWinDelta. The DC will then transform pArgs->xy:
if wsCopyRelative is set, as in msgDcLUCtoLWC_SIZE32
if wsCopyRelative is cleared, as in msgDcLUCtoLWC_XY32.
The DC will then pass the message on to the DC's bound window.

Drawing contexts respond to every other clsWin message by just forwarding the message on to its bound window. The P_ARGS are not touched by the DC.

**msgWinDevBindPixelmap**

Binds window device to a pixelmap.

Takes P_WIN_DEV_PIXELMAP, returns STATUS.

Comments
The DC will pass the message on to pArgs->device. The pArgs->size passed along to the device will be a copy of pArgs->size that the DC has transformed into LWC by:

1) setting up a local rectangle of x=0, y=0, w=pArgs->size.w, h=pArgs->size.h

2) transforming this rectangle into LWC as in msgWinDirtyRect (using the transformation matrix of the DC), and

3) setting the copied size to the resulting rectangle's size.

The DC will also change the pArgs->device passed along to be the device on which the DC's bound window was created.

**msgWinDevSizePixelmap**

Computes the amount of memory needed for a single plane.

Takes P_WIN_DEV_PIXELMAP, returns STATUS.

#ifndef // SYSGRAF_INCLUDED

Comments
The P_ARGS is handled the same as in msgWinDevBindPixelmap.

Drawing contexts respond to every other clsWinDev message by just forwarding the message on to its bound window. Note that clsWin's response to clsWinDev messages is to just call ancestor (except for messages sent to the root window on the device, which get passed on to the device).

The P_ARGS are not touched by the DC.
This file contains the API definition for clsTiff (Tagged Image File Format). clsTiff inherits from clsObject.

clsTiff provides decoding and display of TIFF file to a window.

clsTiff remembers a pathname to a TIFF file; the file must be in the same location on redisplay. TIFF objects are not windows; they take a drawing context to repaint.

clsTiff provides display of the black and white grey scale formats. It decodes compression types for packed data (type 1); Group3 (FAX) horizontal encoding (types 2 and 3); Pack Bits run-length (type 32773). Samples per pixel are limited to 1, 2, 4, or 8. TIFF images must be grey scale; it does not support colormap or direct color (RGB) images. It supports tags for photometric interpretation, fill order, orientation, dot size, Intel & Motorola byte order.

Common uses of clsTiff:

clsTiff can be the data object for a clsView object. It is used by the Fax Viewer in this way to display fax images.

```c
#ifndef TIFF_INCLUDED
#define TIFF_INCLUDED
#ifndef PICSEG_INCLUDED
#include "picseg.h"
#endif
#endif
```

### Messages

#### msgNewDefaults

Initializes a TIFF_NEW structure to default values.

Takes P_TIFF_NEW, returns STATUS. Category: class message.

**Comments**

defaults: tiff.pName = pNull; tiff.imageFlags = sysDcImageFillWindow; tiff.rectangle = zeros;

#### msgNew

Creates a new TIFF object, and optionally opens its associated file.

Takes P_TIFF_NEW, returns STATUS. Category: class message.

**Arguments**

```c
typedef struct TIFF_STYLE {
    U16 save : 1;   // false if reading and display; true for saving
    spare1 : 15;
    U16 spare2 : 16;
} TIFF_STYLE, *P_TIFF_STYLE;
typedef struct {
    P_U8 pName;
    SYSDC_IMAGE_FLAGS imageFlags;  // sysDcImageXXFilter and sysDcImageFillWindow
    RECT32 rectangle;  // display size of the tiff image in LUC
    TIFF_STYLE style;
    S32 spare[3];
} TIFF_NEW_ONLY, *P_TIFF_NEW_ONLY;
```


```c
#define tiffNewFields objectNewFields
         TIFF_NEW_ONLY tiff;
ttypedef struct TIFF_NEW {
   tiffNewFields
} TIFF_NEW, *P_TIFF_NEW;
```

If imageFlags has the sysDclImageFillWindow flag set, msgNew will pass back the size of the image in mils in the rectangle member of the TIFF_NEW struct.

### Status Codes for msgNew

- `stsTiffNumStrips` returned if the number of strips is bad.
- `stsTiffStripByteCount` returned if the number of strip byte counts does not match the image length.
- `stsTiffStripOffsets` returned if there are no strip offsets.
- `stsTiffImageTooLarge` returned if the image is too large to display (32000 pixels by 32000 pixels).
- `stsTiffByteCountZero` returned if the byte count is zero.
- `stsTiffBadName` returned if pName is bad or pNull.
- `stsFSNodeNotFound` returned if the TIFF file is not found.

And status errors form:

```c
#ifndef stsTiffNumStrips
#define stsTiffNumStrips MakeStatus(clsTiff,0)
#endif
#ifndef stsTiffStripByteCount
#define stsTiffStripByteCount MakeStatus(clsTiff,1)
#endif
#ifndef stsTiffStripOffsets
#define stsTiffStripOffsets MakeStatus(clsTiff,2)
#endif
#ifndef stsTiffImageTooLarge
#define stsTiffImageTooLarge MakeStatus(clsTiff,3)
#endif
#ifndef stsTiffByteCountZero
#define stsTiffByteCountZero MakeStatus(clsTiff,4)
#endif
#ifndef stsTiffBadFormatId
#define stsTiffBadFormatId MakeStatus(clsTiff,5)
#endif
#ifndef stsTiffBadName
#define stsTiffBadName MakeStatus(clsTiff,6)
#endif
```

### clsPicSeg messages used by clsTiff

**msgPicSegPaintObject**

Paints the Tiff to the drawing context object provided.

Takes `P_PIC_SEG_PAINT_OBJECT`, returns `STATUS`.

**Comments**

Object Call either `msgWinBeginPaint` or `msgWinBeginRepaint` before using this message. A clsPicSeg object will send this message to any Tiff object in its display list. If the rectangle in `P_PIC_SEG_PAINT_OBJECT` is all zeros then the whole window is filled with the image.

### clsTiff Messages

**msgTiffGetMetrics**

Passes back the metrics of the Tiff.

Takes `P_TIFF_METRICS`, returns `STATUS`.

```c
#define msgTiffGetMetrics MakeMsg(clsTiff, 1)
```
**msgTiffSetMetrics**

Sets the metrics of the Tiff.

Takes P_TIFF_METRICS, returns STATUS.

```c
#define msgTiffSetMetrics MakeMsg(clsTiff, 2)
```

### Orientation defines

Valid values for metrics.orientation

```c
#define tiffOrientTopLeft 1 // 1st row top; 1st column left
#define tiffOrientTopRight 2 // 1st row top; 1st column right
#define tiffOrientBottomRight 3 // 1st row bottom; 1st column right
#define tiffOrientBottomLeft 4 // 1st row bottom; 1st column left
#define tiffOrientLeftTop 5 // 1st row left; 1st column top
#define tiffOrientRightTop 6 // 1st row right; 1st column top
#define tiffOrientRightBottom 7 // 1st row right; 1st column bottom
#define tiffOrientLeftBottom 8 // 1st row left; 1st column bottom
```

### Compression types

Valid values for metrics.compression

```c
#define tiffCompPackedData 1 // only horiz. encoding
#define tiffCompGroup3 2 // only horiz. encoding w/EOL
#define tiffCompFax 3 // Mac pack bits run-length
#define tiffCompPackBits 32773 // Mac pack bits run-length
```

### Rational

The ratio of two longs (num / dem).

```c
typedef struct {
    U32 num;
    U32 dem;
} RATIONAL, * P_RATIONAL;
```

### Metrics

The data read from the file tags.

```c
typedef struct {
    P_U8 pFileName; // the path for the file
    RECT32 rectangle; // the display rect
    SYSDC_IMAGE_FLAGS imageFlags;
    U32 newSubfileType; // the tiff data read from the file
    U16 SubfileType; // the only supported value
    U32 width; // number of pixels in the x dimension
    U32 length; // number of pixels in the y dimension
    U16 bitsPerSample; // number of bits per sample 1, 2, 4 or 8
    U16 compression; // the image compression type
    U16 photometricInterpretation; // 0 - 0 black; highest value white
        // 1 - highest value black; 0 white
    U16 fillOrder; // bit order of image bytes
        // 1 - MSB first; 2 - LSB first
    P_S8 pDocumentName; // pointer to a string in a heap or pNull
    P_S8 pImageDescription; // pointer to a string in a heap or pNull
    P_S8 pMake; // pointer to a string in a heap or pNull
    P_S8 pModel; // pointer to a string in a heap or pNull
    P_S32 pStripOffsets; // pointer to an array of file locations
```
U16 orientation;       // see orient #defines for values
U16 samplesPerPixel;  // number of samples per pixel
S32 rowsPerStrip;     // number of scanlines per strip
P_S32 pStripByteCounts; // array of byte counts in each strip
RATIONAL xResolution; // x number of samples per resolution unit
RATIONAL yResolution; // y number of samples per resolution unit
U16 planarConfiguration; // 1 the only supported value
P_S8 pPageName;       // pointer to a string in a heap or pNull
RATIONAL xPosition;   // current x position (UNUSED)
RATIONAL yPosition;   // current y position (UNUSED)
U32 group3Options;    // only works if 0
U16 resolutionUnit;   // 1 for inches; 2 for millimeters
U16 pageNumber;       // page number for the image
P_S8 pSoftware;       // pointer to a string in a heap or pNull
P_S8 pDataTime;       // pointer to a string in a heap or pNull
P_S8 pArtist;         // pointer to a string in a heap or pNull
P_S8 pHostComputer;  // pointer to a string in a heap or pNull
P_U16 pColorMap;      // pointer to an array in a heap or pNull
)

} TIFF_Metrics, *P_TIFF_Metrics;

msgTiffGetSizeMils

Provides the actual size of the TIFF image in MILS (1/1000 inch).

Takes P_SIZE32, returns STATUS.

#define msgTiffGetSizeMils MakeMsg(clsTiff, 3)

msgTiffGetSizeMM

Provides the actual size of the TIFF image in millimeters.

Takes P_SIZE32, returns STATUS.

#define msgTiffGetSizeMM MakeMsg(clsTiff, 4)

msgTiffSave

Saves a TIFF file.

Takes P_TIFF_SAVE, returns STATUS.

#define msgTiffSave MakeMsg(clsTiff, 5)

// Format of Input image (style.inputDataFormat)
// The stored data type is provided in the tiff metrics.
// Currently the only conversion of image compression is
// from tiffSaveRunLength to tiffCompGroup3. The data provided for other
// compression types is written directly to the file with no conversion.
#define tiffSavePackedData 1    // NOT WORKING
#define tiffSavePackedBits 2    // NOT WORKING
#define tiffSaveRunLength 3     // can only be use for a Group 3 Fax file
#define tiffSaveGroup3 4        // NOT WORKING
// How the image data is provided (style.provideData)
#define tiffCallBack 1          // use tiffSave.callback.function() to get row
#define tiffCallObject 2        // not working
#define tiffProvided 3          // all the data is in pBuffer (NOT WORKING)

typedef struct TIFF_SAVE_STYLE {
    U16 inputDataFormat : 4,  // the compression of the input image data
    provideData : 3,
    convert : 1,            // on if the input data is to be converted
    spare1 : 8;
    U16 spare2 : 16;
} TIFF_SAVE_STYLE, *P_TIFF_SAVE_STYLE;
typedef struct TIFF_SAVE * P_TIFF_SAVE;
typedef STATUS FunctionPtr(P_TIFF_GETROW) (P_TIFF_SAVE pTiffSave);
typedef struct TIFF_SAVE {
    TIFF_SAVE_STYLE style;
    union {
        P_TIFF_GETROW function;
        object;
    } // ObjectCall with msgTiffGetRow
    callBack;
    U32 bufferCount;
        // number of bytes in pBuffer
        // if 0 its assumed there is no
        // more data and metrics.length
        // will be changed
    P_U8 pBuffer;
    object;
    // provided by the client
    P_UNKNOWN pClientData;
    // clients own data
} TIFF_SAVE;

Comments
The TIFF object must be created with the save style (tiff.style.save = true). The metrics of the TIFF
must first be set. The default metrics are:

metrics.newSubfileType = 1;
metrics.SubfileType = 1;
metrics.width = 0;
metrics.length = 0;
metrics.bitsPerSample = 1;
metrics.compression = 1;
metrics.photometricInterpretation = 0;
metrics.fillOrder = 1;
metrics.pDocumentName = pNull;
metrics.pImageDescription = pNull;
metrics.pMake = pNull;
metrics.pModel = pNull;
metrics.samplesPerPixel = 1;
metrics.orientation = tiffOrientTopLeft;
metrics.pStripOffsets = pNull;
metrics.pStripByteCounts = pNull;
metrics.rowsPerStrip = 0L;
metrics.xResolution.num = 0L;
metrics.xResolution.den = 0L;
metrics.yResolution.num = 0L;
metrics.yResolution.den = 0L;
metrics.planarConfiguration = 1;
metrics.pPageName = pNull;
metrics.group3Options = 0L;
metrics.resolutionUnit = 2;
metrics.pageNumber = 0;
metrics.pSoftware = pNull;
metrics.pDataTime = pNull;
metrics.pArtist = pNull;
metrics.pHostComputer = pNull;
metrics.pColorMap = pNull;

All pointers should be allocated on a heap with OSHeapBlockAlloc(). It will save any strings and arrays
that are not pNull. Strip offsets and strip byte counts are calculated while the image is being saved.

msgTiffSetGroup3Defaults
Sets the TIFF metrics to the Group3 compression type 2 defaults.
Takes P_TIFF_SAVE, returns STATUS.

#define msgTiffSetGroup3Defaults MakeMsg(clsTiff, 6)
typedef struct TIFF_SAVE {
    TIFF_SAVE_STYLE style;
    union {
        P_TIFF_GETROW function;
        OBJECT object;
        callBack;
    }
    U32 bufferCount;

    P_U8 pBuffer;
    P_UNKNOWN pClientData;
} TIFF_SAVE;

Takes 0 for low resolution and 1 for high resolution.

msgTiffGetRow
Sent client of the TIFF_SAVE to get the next row of the image.
Takes U32, returns STATUS.

#define msgTiffGetRow

ReverseBits
Reverses the bit ordering in each byte in an array of bytes.
Returns void.

Function Prototype

void EXPORTED ReverseBits(
    P_U8 pBuf,       // the bytes to reverse
    S32 nBytes       // the number of bytes to reverse
);
Interface to the pop-up tiling routine.

The functions described in this file are contained in MISCLIB.

```c
#ifndef TILE_INCLUDED
#define TILE_INCLUDED
#endif
#ifndef GO_INCLUDED
#include <go.h>
#endif
#ifndef GEO_INCLUDED
#include <geo.h>
#endif

typedef enum {
    tileAbove, // above the target
    tileBelow, // below the target
    tileLeft,  // to the left of the target
    tileRight, // to the right of the target
} TILE_LOCATOR;

TilePopUp

Center a rectangle under (over/to the left/right of) another rectangle but staying inside the bounds of a third rectangle.

Returns STATUS.

Function Prototype

STATUS PASCAL TilePopUp(
    TILE_LOCATOR preferred, // preferred location
    P_RECT32 pPop, // In-Out rect to be manipulated
    P_RECT32 pTarget, // anchor rect
    P_RECT32 pWorld // surrounding rect, base for pop & target
); // use pNull for the Root Window

This routine makes it easy to position pop-up windows next to existing or screen regions. pPop->origin is set to the best position to that rectangle. For example, if you want to center a popup window over a selected word but stay inside the RootWindow, you’d set preferred tileBelow, pPop->size to the size of the new window, pTarget to the containing the selection, and pWorld to pNull. If a window of size can be centered below pTarget, TilePopUp will return the position to insert it at. If it won’t fit below, but it will fit above, will give that position. If it will fit below, but not centered, will sacrifice centering to keep it all on screen.

All rects are assumed to be relative to the same origin. You stillto actually position and insert the actual window; this just you where to put it.
WIN.H

This file provides the API's for clsWin, clsWinDev. Two abstract classes, clsDrwCtx and clsPixDev are also defined, but they are not used directly by application-level clients.

clsDrwCtx inherits from clsObject.
Defines the minimal behavior for a drawing context.

clsPixDev inherits from clsObject.
Defines the minimal behavior for a pixmap graphics device.

clsWinDev inherits from clsPixDev.
Provides devices of clsPixDev that can have windows on them.

clsImgDev inherits from clsWinDev.
Provides window devices whose pixels are accessible in memory.

clsWin inherits from clsObject.
Provides windows onto clsWinDev objects.

theScreen is a well-known instance of clsWinDev. It is the main display surface for PenPoint.

theRootWindow is a well-known instance of clsWin. It is the root of the window tree on theScreen.

Terminology:

DU4 -- Device Units, 4th Quadrant. A 4th quadrant coordinate system; device space, device units. This is used internally, but not seen by application software.

LWC -- Logical Window Coordinates. A 1st quadrant coordinate system. The lower-left-hand corner of the window is 0,0. The units are device pixels. These are the coordinates in which windowing operations are specified and input is delivered.

LUC -- Logical Unit Coordinates. The nature of the coordinate system is determined by a drawing context. Such coordinates are always relative to the window. Some drawing contexts will implement window messages that takes LWC coordinates and transform them so that window operations can occur in LUC space. See sysgraf.h for details.

Debugging Flags

The clsWin debugging flag is 'W'. Defined values are:

flag0 (0x0001) window layout
flag1 (0x0002) window layout
flag2 (0x0004) flash interesting regions during damage
flag3 (0x0008) bitmap caching
flag4 (0x0010) window filing
flag5 (0x0020) font cache char ops
flag6 (0x0040) font cache char ops
flag7 (0x0080) matrix/rectangle math
flag8 (0x0100) layout timing
flag9 (0x0200) font cache macro ops
flag10 (0x0400) msgWinDumpTree outputs input flags
flag11 (0x0800) Measure/Draw text
flag12 (0x1000) window printing/clipping
flag13 (0x2000) unused
flag14 (0x4000) unused
flag15 (0x8000) window bitblt coordinates

ifndef WIN_INCLUDED
#define WIN_INCLUDED
ifndef GO_INCLUDED
#include <go.h>
#endif
ifndef CLSMGR_INCLUDED
#include <clsmgr.h>
#endif
ifndef GEO_INCLUDED
#include <geo.h>
#endif

Typedefs, #defines, and Status Values

#define stsWinConstraint MakeStatus(clsWin, 1)
#define stsWinHasParent MakeStatus(clsWin, 3)
#define stsWinParentBad MakeStatus(clsWin, 4)
#define stsWinNoEnv MakeStatus(clsWin, 5)
#define stsWinInfiniteLayout MakeStatus(clsWin, 6)
#define stsWinNoEnv MakeStatus(clsWin, 7)
#define stsWinIsChild MakeWarning(clsWin, 8)
#define stsWinIsDescendant MakeWarning(clsWin, 9)
#define stsPixDevBad MakeStatus(clsPixDev, 1)
#define stsPixDevOutOfRegions MakeStatus(clsPixDev, 2)
#define stsWinDevBad MakeStatus(clsWinDev, 1)
#define stsWinDevFull MakeStatus(clsWinDev, 2)
#define stsWinDevCachedHit MakeStatus(clsWinDev, 3)
typedef WIN * P_WIN;
typedef DRW_CTX * P_DRW_CTX;
typedef PIX_DEV * P_PIX_DEV;
typedef WIN_DEV * P_WIN_DEV;
typedef IMG_DEV * P_IMG_DEV;

Window style flags

#define wsClipChildren (U32)flag0) // Don’t draw on my children
#define wsClipSiblings (U32)flag1) // Don’t draw on my siblings
#define wsParentClip (U32)flag2) // Borrow my parent’s vis rgn
#define wsSaveUnder (U32)flag3) // Try to save pixels on insert
#define wsGrowTop (U32)flag4) // Pixels move to bottom on resize
#define wsGrowBottom (U32)flag5) // Pixels move to top on resize
#define wsGrowLeft (U32)flag6) // Pixels move to right on resize
#define wsGrowRight (U32)flag7) // Pixels move to left on resize
#define wsCaptureGeometry (U32)flag8) // I capture m,s,i,e of children
#define wsSendGeometry (U32)flag9) // Send me delta,ins,ext advice
#define wsSendOrphaned (U32)flag10) // Send msgOrphaned not msgFree
WIN.H 297

Window style flags

define wsSynchRepaint (U32)flag12 // ObjectCall to repaint
define wsTransparent (U32)flag13 // I am transparent
define wsVisible (U32)flag14 // I am visible
define wsPaintable (U32)flag15 // I can be painted
define wsSendFile (U32)flag16 // I should be filed

define wsShrinkWrapWidth (U32)flag17 // I shrink to fit children
define wsShrinkWrapHeight (U32)flag18 // I shrink to fit children
define wsLayoutDirty (U32)flag19 // My layout is dirty
#define wsCaptureLayout ((U32)flag20) // child wsVisible changes

define wsSendLayout ((U32)flag21) // I'm dirty if I change size or
#define wsSendLayoutWidth ((U32)flag22) // wsShrinkWrapWidth/Height or
#define wsSendLayoutHeight ((U32)flag23) // wsMaskWrapWidth/Height changes

define wsHeightFromWidth ((U32)flag24) // height is computed from width
#define wsWidthFromHeight ((U32)flag25) // width is computed from height

typedef struct
{
  U32 input,                  // see input.h
  style;                     // see ws* flags above
} WIN_FLAGS, *P_WIN_FLAGS;  // part of WIN_METRICS

#define WinShrinkWrapWidth(style) \(! ((style) & wsMaskWrapWidth) \&\& ((style) & wsShrinkWrapWidth))

#define WinShrinkWrapHeight(style) \(! ((style) & wsMaskWrapHeight) \&\& ((style) & wsShrinkWrapHeight))

#define WinShrinkWrap(style) \((WinShrinkWrapWidth(style) || WinShrinkWrapHeight(style)))

You can use these WinShrinkWrap macros are test if a window has shrink-wrap-width or
shrink-wrap-height enabled. If wsMaskWrapWidth/Height is on, the shrink wrapping will be off in
that dimension. clsGrabBox will turn on wsMaskWrapWidth/Height if the user resizes a window and
changes the width/height. clsFrame will clear the wsMaskWrapWidth/Height bits and re-layout when
the user triple-taps on the title bar.

Enum16(WIN_OPTIONS)
{
  wsPosTop = 0,
  wsPosBottom = flag0,     // In: to msgWinInsert...
  wsPosInFront = wsPosTop,
  wsPosBack = wsPosBottom,
  wsWinMoved = flag9,      // Out: from msgWinDelta
  wsWinSized = flag10,     // Out: from msgWinDelta
  wsParentMoved = flag12,  // Out: from msgWinDelta
  wsParentSized = flag13,  // Out: from msgWinDelta
  wsLayoutResize = flag11, // In: to msgWinLayout...
  wsLayoutMinPaint = flag14, // In: to msgWinLayout...
  wsLayoutNoCache = flag8,  // Out: from msgWinLayoutSelf
  wsLayoutDefault = wsLayoutResize // In: to msgWinLayout...
};
typedef struct
{
    WIN parent,
    child;
    RECT32 bounds;
    WIN_DEV device;
    WIN_FLAGS flags;
    TAG tag;
    WIN_OPTIONS options;
} WIN_METRICS, *P_WIN_METRICS;

A P_WIN_METRICS is the argument to most of the messages defined by clsWin. However, for most of these messages, not all of the fields are used. In the discussion of each message below, fields which are not mentioned are not used; and they don’t have to be initialized before sending the message. This is not to say that these "unused" fields are not modified during the call; they will be during the processing of some messages.

**Messages Sent to a Window**

**msgNew**

Creates a window.

Takes P_WIN_METRICS, returns STATUS. Category: class message.

typedef WIN_METRICS WIN_NEW_ONLY, *P_WIN_NEW_ONLY;
#define winNewFields \  
   objectNewFields \  
   WIN_NEW_ONLY win;

Arguments
typedef struct WIN_NEW
{
   winNewFields
} WIN_NEW, *P_WIN_NEW;

Message
Arguments
typedef struct
{
    WIN parent,
    child;
    RECT32 bounds;
    WIN_DEV device;
    WIN_FLAGS flags;
    TAG tag;
    WIN_OPTIONS options;
} WIN_METRICS, *P_WIN_METRICS;

Comments
If pArgs->parent is not objNull, clsWin will create the window on the specified parent's window device. Note that the new window will not be inserted as a child of the specified parent. You must send msgWinInsert to the new window after creating it to insert it into its parent.

If pArgs->parent is objNull, the window will be created on pArgs->device. If pArgs->device is objNull, clsWin will create the window on OSThisWinDev().

Returns
stsWinParentBad if pArgs->parent is not objNull or a valid window
stsWinDevBad if pArgs->device is not objNull or a valid window device
stsWinDevFull if the window device window array can't be grown

See Also
msgWinInsert
msgNewDefaults
Initializes the WIN_NEW structure to default values.
Takes P_WIN_NEW, returns STATUS. Category: class message.

typedef struct WIN_NEW
{
    winNewFields
    WIN_NEW, *P_WIN_NEW;
    object.cap = objCapCall;
    win.parent = objNull;
    win.child = objNull;
    win.device = objNull;
    win.flags.style = wsDefault;
    win.flags.input = 0;
    win.tag = 0;
    win.options = wspPosTop;
    win.bounds.origin.x = 0;
    win.bounds.origin.y = 0;
    win.bounds.size.w = 0;
    win.bounds.size.h = 0;
}

msgWinInsert
Inserts or changes z-order of a window.
Takes P_WIN_METRICS, returns STATUS.

#define msgWinInsert
typedef struct
{
    WIN parent,
    child;
    RECT32 bounds;
    WIN_DEV device;
    WIN_FLAGS flags;
    TAG tag;
    WIN_OPTIONS options;
} WIN_METRICS, *P_WIN_METRICS;

You send this message to the child that you want to insert or change z-order.
In parameters are:
    pArgs->parent  = child's new parent or objNull;
    pArgs->options = either wspPosTop or wspPosBottom;

If pArgs->parent is not objNull or self's current parent, clsWin will insert self as a child of the specified parent. If pArgs->options has wspPosTop on, self will be inserted as the top-most child; if wspPosBottom is on, self will be inserted as the bottom-most child.

If pArgs->parent is objNull or self's current parent, clsWin will change the z-order of self according to pArgs->options. If pArgs->options has wspPosTop on, self will be altered in z-space to be the top-most child; if wspPosBottom is on, self will be altered to be the bottom-most child. If the z-order of self is changed, wsWinMoved will be or-ed into pArgs->options as an out parameter.

If the receiver's parent has wsCaptureLayout on, wsLayoutDirty will be set on the receiver's parent.
Returns
    stsWinParentBad if pArgs->parent is not objNull or a valid window on the same window device as self
stsWinHasParent if self already has a parent and pArgs->parent is not either objNull or self's current parent.

See Also
msgWinInsertSibling

msgWinInsertSibling

Inserts or changes z-order of a window (relative to a sibling).

Takes P_WIN_METRICS, returns STATUS.

#define msgWinInsertSibling MakeMsg(clsWin, 2)

Message

typedef struct
{
    WIN parent,
    child;
    RECT bounds;
    WIN_DEV device;
    WIN_FLAGS flags;
    TAG tag;
    WIN_OPTIONS options;
    } WIN_METRICS, *P_WIN_METRICS;

Comments

You send this message to the child that you want to insert or change z-order. This message is similar to msgWinInsert, except pArgs->parent should be the intended sibling of the receiver.

In parameters are:

pArgs->parent = receiver's new sibling
pArgs->options = either wsPosTop or wsPosBottom;

clsWin will insert self as a sibling of the specified sibling. If pArgs->options has wsPosTop on, self will be inserted as in front of pArgs->parent; if wsPosBottom is on, self will be inserted behind pArgs->parent.

If pArgs->parent is already self's sibling, clsWin will change the z-order of self according to pArgs->options. If pArgs->options has wsPosTop on, self will be altered in z-space to be in front of pArgs->parent; if wsPosBottom is on, self will be altered to be behind pArgs->parent. If the z-order of self is changed, wsWinMoved will be or-ed into pArgs->options as an out parameter.

If the receiver's parent has wsCaptureLayout on, wsLayoutDirty will be set on the receiver's parent.

Returns

stsWinParentBad if pArgs->parent is not a valid window on the same window device as self
stsWinHasParent if self already has a parent and pArgs->parent is not a sibling of self

See Also
msgWinInsert

msgWinExtract

Extracts a window from its parent.

Takes P_WIN_METRICS or pNull, returns STATUS.

#define msgWinExtract MakeMsg(clsWin, 3)

Comments

If a P_WIN_METRICS is passed instead of pNull the same information returned by msgWinGetMetrics is returned in the WIN_METRICS structure. This will include the parent field BEFORE the extract is performed. If the window is already extracted, stsWinParentBad is returned, and any passed WIN_METRICS field is unmodified.

If the receiver's parent has wsCaptureLayout on, wsLayoutDirty will be set on the receiver's parent.
**msgWinDelta**

Moves and/or resizes a window. `pArgs->bounds` should be the newly desired bounds (size AND position).

Takes `P_WIN_METRICS`, returns `STATUS`.

```
#define msgWinDelta MakeMsg(clsWin, 4)
```

```
typedef struct {
    WIN RECT32 parent,
    child;
    RECT32 bounds;
    WIN_DEV device;
    WIN_FLAGS flags;
    TAG tag;
    WIN_OPTIONS options;
} WIN_METRICS, *P_WIN_METRICS;
```

If the receiver is involved in a layout episode (msgWinLayout is being processed in the receiver's window tree), the new bounds will be remembered for use at the end of the layout episode. If the new bounds has a new width or height, and a cached desired size is being remembered for the receiver, the desired size will be discarded if either of the following is true:

- the new bounds has a new width and the receiver has `wsHeightFromWidth` on or does not have `wsShrinkWrapWidth` on
- the new bounds has a new height and the receiver has `wsWidthFromHeight` on or does not have `wsShrinkWrapHeight` on

If the receiver is involved in a layout episode this is all that is done and `stsOK` is returned.

If the receiver's parent has `wsCaptureGeometry` on, the parent will be sent `msgWinDeltaOK`. If the parent responds with anything other than `stsOK`, that status will be returned and nothing else is done. Otherwise, the (possibly modified) bounds returned by the parent will be used. If the parent modified the proposed child origin, `wsParentMoved` will be or-ed into `pArgs->options` as an out parameter. If the parent modified the proposed child size, `wsParentSized` will be or-ed into `pArgs->options` as an out parameter.

If the receiver is visible and paintable (`wsVisible` and `wsPaintable` are on for the receiver and all of its ancestors), valid portions of the receiver's window may be copied to their new location to avoid damage and repaint of those portions.

If the receiver has any of the grow bits on (`wsGrowBottom/Top/Left/Right`), the appropriate grow semantics will be applied to determine how to move the receiver's children and what portions of the receiver's window to damage for subsequent repaint.

If `pArgs->bounds` is a new bounds and the receiver's parent has `wsCaptureLayout` on, `wsLayoutDirty` will be set on the receiver's parent.

If `pArgs->bounds.size` is a new size and the receiver has `wsSendLayout` on, `wsLayoutDirty` will be set on the receiver.

Subclasses that want to know when their position or size has changed should not expect that `msgWinDelta` is the only way for this to happen. If you need to know this information, you should turn on `wsSendGeometry` and catch `msgWinMoved` or `msgWinSized`. `clsWin` may change a window's bounds without sending `msgWinDelta` to the window.
msgWinLayout

Tells a window sub-tree to layout.

Takes P_WIN_METRICS, returns STATUS.

```c
#define msgWinLayout MakeMsg(clsWin, 41)
```

typedef struct
{
    WIN parent,
    child;
    RECT32 bounds;
    WIN DEV device;
    WIN_FLAGS flags;
    TAG tag;
    WIN_OPTIONS options;
} WIN_METRICS, *P_WIN_METRICS;

You should send msgWinLayout to a window after you have altered the window in such a way that its bounds or its descendants bounds must be recomputed.

For example, if you create an instance of clsTableLayout (a subclass that lays out its children in rows and columns) and insert children into it, you must send msgWinLayout to the table layout window to force it to "layout" itself and its children.

After msgWinLayout has been sent, every window in the receiver's tree will be positioned and sized as required. You can then use msgWinInsert to insert the root of the tree on the display and allow the windows to paint.

In parameters:

- `bounds` = new final bounds for receiver if wsLayoutResize is not on in pArgs->options
- `options` = wsLayoutDefault, 0, or any combination of wsLayoutResize, wsLayoutMinPaint

Subclasses must not catch msgWinLayout. clsWin will respond by beginning a "layout episode" during which the windows in the receiver's tree will be layed out.

The algorithm for a layout episode is as follows:

for the receiver and each of its descendants
   If the window has wsLayoutDirty on
      If the bounds of the window have been fixed by a previous msgWinDelta during the layout episode
         send the window msgWinLayoutSelf with the following WIN_METRICS parameters:
         bounds.size = current bounds.size;
         options = 0;
      Otherwise,
         send the window msgWinLayoutSelf with the following WIN_METRICS parameters:
         options = wsLayoutResize;
         copy back WIN_METRICS.bounds.size as the new size for the window.
   Otherwise,
      If the window's parent has wsLayoutDirty on, switch to the window's parent and continue down the tree from there.

After the entire tree has been traversed, traverse the tree again and process wsCaptureGeometry and wsSendGeometry requests as follows:
WIN.H 303

Messages Sent to a Window

For each window

If the origin or size has changed

If the window's parent has wsCaptureGeometry on
send msgWinDeltaOK to the window's parent;

If the window has wsSendGeometry on
send msgWinMoved and/or msgWinSized to the window.

After the geometry notifications have been done, apply all of the new
bounds for each window in the tree as in msgWinDelta.

If wsLayoutResize is NOT set in pArgs->options, then you must set pArgs->bounds to the new
rectangle that the receiver must fit into -- it will lay out accordingly; otherwise the receiver will lay out to
its desired size.

If wsLayoutMinPaint is not on, window damage will not be computed during the layout episode -- all
of the windows in the window tree will be damaged and repaint after the layout episode. This will result
in faster layout, at the expense of some (possibly) unnecessary repaints. If wsLayoutMinPaint is on, the
true damaged area will be computed. This may take longer, but will result in the minimal amount of
repaint after the layout episode.

In general this message should not be handled by subclasses. However, it results in the sending of
msgWinLayoutSelf, which does need to be handled by subclasses.

Returns

stsWinInfiniteLayout the layout episode does not appear to terminate
msgWinLayoutSelf

See Also

msgWinLayoutSelf

tells a window to layout its children (sent during layout).

Takes P_WIN_METRICS, returns STATUS.

#define msgWinLayoutSelf MakeMsg(clsWin, 42)

This message is sent by clsWin during a layout episode. It can be handled by knowledgeable window
classes.

When sent, pArgs->bounds.size contains the present size. If pArgs->options is 0 then the window
cannot change pArgs->bounds.size, it must lay out its children, as best it can, within those bounds. If
pArgs->option is wsLayoutResize then it may change pArgs->bounds.size to its desired size.

After pArgs->bounds.size is determined, the window should msgWinDelta each child to its final
position and size.

In order to determine its desired size and layout, a window may need to send msgWinGetDesiredSize to
some, or all, of its children first.
**msgWinGetDesiredSize**

Gets the desired size of a window (sent during layout).

Takes P_WIN_METRICS, returns STATUS.

```c
#define msgWinGetDesiredSize MakeMsg(clsWin, 43)
```

**Message Arguments**

```c
typedef struct {
    WIN parent,
    child;
    RECT32 bounds;
    WIN_DEV device;
    WIN_FLAGS flags;
    TAG tag;
    WIN_OPTIONS options;
} WIN_METRICS, *P_WIN_METRICS;
```

**Comments**

This message should not be handled by a subclass.

If the receiver is not in a layout episode, clsWin responds by returning the receiver's current bounds. Otherwise, if the desired size has already been computed (cached) for the receiver, that value will be returned.

Otherwise, `msgWinLayoutSelf` will be self-sent with the following WIN_METRICS parameters:

```c
    options = wsLayoutResize;
```

Subclasses should catch `msgWinLayoutSelf`, layout to their desired size and return the desired size in WIN_METRICS.bounds.size. The computed desired size will be remembered in the window's cache for future use and will be passed back in `pArgs->bounds.size`.

**See Also**

`msgWinLayout`

---

**msgWinGetBaseline**

Gets the desired x,y alignment of a window.

Takes P_WIN_METRICS, returns STATUS.

```c
#define msgWinGetBaseline MakeMsg(clsWin, 46)
#define wsNoXBaseline ((U16)flag0)
#define wsNoYBaseline ((U16)flag1)
```

**Message Arguments**

```c
typedef struct {
    WIN parent,
    child;
    RECT32 bounds;
    WIN_DEV device;
    WIN_FLAGS flags;
    TAG tag;
    WIN_OPTIONS options;
} WIN_METRICS, *P_WIN_METRICS;
```

**Comments**

Subclasses can set `pArgs->bounds.origin` to reflect the window's desired baseline position. clsWin will set both x and y to 0.0.
pArgs->bounds.size should contain the size of the window. This is useful for windows whose alignment is a function of window size (like centered).

If the receiver does not have either an x or y baseline, wsNoXBaseline and/or wsNoYBaseline can be or-ed into pArgs->options as an out parameter.

clsWin will always set pArgs->options to wsNoXBaseline | wsNoYBaseline (i.e. the default is the window has no x or y baseline).

---

### msgWinSetLayoutDirty

Turns wsLayoutDirty bit on or off, returns previous value.

Takes BOOLEAN, returns BOOLEAN.

```c
#define msgWinSetLayoutDirty MakeMsg(clsWin, 44)
```

**Comments**

If the window has a cached desired size, and wsLayoutDirty comes on, the desired size will be discarded.

---

### msgWinSetLayoutDirtyRecursive

Turns wsLayoutDirty bit on for every window in subtree.

Takes BOOLEAN, returns nothing.

```c
#define msgWinSetLayoutDirtyRecursive MakeMsg(clsWin, 45)
```

---

### msgWinSend

Sends a message up a window ancestry chain.

Takes P_WIN_SEND, returns STATUS.

```c
#define msgWinSend MakeMsg(clsWin, 36)
```

**Arguments**

```c
Enum16 (WIN_SEND_FLAGS)
{
    wsSendDefault = 0,
    wsSendIntraProcess = flagO,   // stop at process transition
};
```

```c
typedef struct
{
    U32 lenSend;             // length of message,
    WIN_SEND_FLAGS flags;   // SizeOf(WIN_SEND) minimum
    MESSAGE msg;             // the "message"
    P_UNKNOWN data[1];      // an argument to the message
    // clients can put
    // more data here
    // if needed
} WIN_SEND, *P_WIN_SEND;
```

**Comments**

The receiver may reply to the message or forward the message up the window parent chain. clsWin will forward the message to the parent using ObjectSendUpdate. If the message reaches the root window stsMessageIgnored is returned. If the wsSendIntraProcess flag is true the message will not be propagated past a process transition (based on the owner of the window object); in this case stsMessageIgnored may also be returned.

lenSend must be at least SizeOf(WIN_SEND) but may be larger to move more data to a window owned by another process. A single unit of data, data[0] is defined in WIN_SEND as a convenience. The message a window receives when msgWinSend is forwarded is msgWinSend, NOT msg. The field msg is provided so the receiving client can properly interpret the purpose of the msgWinSend.
msgWinGetMetrics

Gets full window metrics.

Takes P_WIN_METRICS, returns STATUS.

```c
#define msgWinGetMetrics
typedef struct
{
    WIN parent,
    child;
    RECT32 bounds;
    WIN_DEV device;
    WIN_FLAGS flags;
    TAG tag;
    WIN_OPTIONS options;
} WIN_METRICS, *P_WIN_METRICS;
```

Comments

pArgs->parent passes back the receiver’s parent pArgs->child passes back self pArgs->bounds passes back size and parent relative position pArgs->device passes back self’s device pArgs->flags passes back self’s window and input flags pArgs->tag passes back self’s tag

msgWinGetFlags

Like msgWinGetMetrics but passes back flags only.

Takes P_WIN_METRICS, returns STATUS.

```c
#define msgWinGetFlags
typedef struct
{
    WIN parent,
    child;
    RECT32 bounds;
    WIN_DEV device;
    WIN_FLAGS flags;
    TAG tag;
    WIN_OPTIONS options;
} WIN_METRICS, *P_WIN_METRICS;
```

Comments

pArgs->flags passes back self’s window and input flags.

msgWinSetFlags

Sets the window flags.

Takes P_WIN_METRICS, returns STATUS.

```c
#define msgWinSetFlags
typedef struct
{
    WIN parent,
    child;
    RECT32 bounds;
    WIN_DEV device;
    WIN_FLAGS flags;
    TAG tag;
    WIN_OPTIONS options;
} WIN_METRICS, *P_WIN_METRICS;
```

Comments

pArgs->flags should be set to the new window and input flags.
If `wsVisible` is changed and the receiver's parent has `wsCaptureLayout` on, `wsLayoutDirty` will be set on the receiver's parent.

If the new flags result in a new value for `WinShrinkWrap()` (e.g. `wsShrinkWrapWidth` changes) and the receiver has `wsSendLayout` on, `wsLayoutDirty` will be set on the receiver.

### msgWinGetTag

Like `msgWinGetMetrics` but passes back tag only.

Takes `P_WIN_METRICS`, returns `stsOK`.

```c
#define msgWinGetTag
typedef struct
{
    WIN parent,
    child;
    RECT32 bounds;
    WIN_DEV device;
    WIN_FLAGS flags;
    TAG tag;
    WIN_OPTIONS options;
} WIN_METRICS, * P_WIN_METRICS;

pArgs->tag passes back self's tag.
```

### msgWinSetTag

Sets the window tag.

Takes `P_WIN_METRICS`, returns `STATUS`.

```c
#define msgWinSetTag
typedef struct
{
    WIN parent,
    child;
    RECT32 bounds;
    WIN_DEV device;
    WIN_FLAGS flags;
    TAG tag;
    WIN_OPTIONS options;
} WIN_METRICS, * P_WIN_METRICS;

pArgs->tag should be set to the new window tag.
```

### msgWinIsVisible

Returns `stsOK` if the receiver and all its ancestors have `wsVisible` on.

Takes nothing, returns `STATUS`.

```c
#define msgWinIsVisible
clsWin will traverse the parent chain of the receiver until the parent is `objNull` or the root window of the receiver's device. If the receiver or any of its ancestors have `wsVisible` off in their window flags, `stsFailed` is returned. Otherwise, if the final ancestor is the root window on the receiver's device, `stsOK` is returned.
```
msgWinIsDescendant

Checks if pArgs->child is a descendant of the receiver.

Takes P_WIN_METRICS, returns STATUS.

```c
#define msgWinIsDescendant
MakeMsg(clsWin, 59)
```

In parameters:
- child: child to look for
- options: 0 for direct children, wsEnumRecursive for recursive or-in wsEnumSelf to include self in the search

clsWin will check the receiver's children and return stsWinIsChild if pArgs->child is one of them.

If pArgs->options has wsEnumRecursive on, the search will continue down the window tree until pArgs->child is found or all of the receiver's descendants have been examined. If no match is found, stsNoMatch is returned.

If pArgs->child is self and wsEnumSelf is on in pArgs->options, stsWinIsChild is returned; otherwise stsNoMatch is returned.

Returns Value
- stsWinIsChild if pArgs->child is self or a direct child
- stsWinIsDescendant if pArgs->child is a descendant
- stsNoMatch if pArgs->child is not a descendant

msgWinGetPopup

Gets the popup window.

Takes P_WIN_METRICS, returns stsOK.

```c
#define msgWinGetPopup
MakeMsg(clsWin, 53)
```

pArgs->child passes back self's popup window.

The popup window is traversed during msgWinFindTag only. See msgWinFindTag for more details.
**msgWinSetPopup**

Sets a popup window.

Takes P_WIN_METRICS, returns STATUS.

```c
#define msgWinSetPopup MakeMsg(clsWin, 54)
```

**Message Arguments**

```c
typedef struct {
    WIN parent,
    child;
    RECT32 bounds;
    WIN_DEV device;
    WIN_FLAGS flags;
    TAG tag;
    WIN_OPTIONS options;
} WIN_METRICS, * P_WIN_METRICS;
```

**Comments**

pArgs->child should be set to the popup window.

The popup window is traversed during msgWinFindTag only. See msgWinFindTag for more details.

One example of popup window use is in clsMenuButton. A menu button will set its popup window to be its menu. This allows you to use msgWinFindTag on a menu bar and find a menu button in one of the popup menus.

**msgWinFindAncestorTag**

Searches for a match on argument tag. Returns match or objNull.

Takes U32, returns OBJECT.

```c
#define msgWinFindAncestorTag MakeMsg(clsWin, 49)
```

**Comments**

The search is up the ancestor chain; the first match found is returned. If no match is found, objNull is returned.

**msgWinFindTag**

Searches for a match on argument tag. Returns match or objNull.

Takes U32, returns OBJECT.

```c
#define msgWinFindTag MakeMsg(clsWin, 39)
```

**Comments**

The search is breadth first; but, it starts with the first child of the window, not the window itself. The first match found is returned. If no match is found, objNull is returned. Trees rooted at popup windows (set with msgWinSetPopup) are traversed too. The traversal order is siblings first, then children, then popups.

**msgWinSetVisible**

Turns window visibility bit on or off, returns previous value.

Takes BOOLEAN, returns BOOLEAN.

```c
#define msgWinSetVisible MakeMsg(clsWin, 8)
```

**Comments**

If visibility is changed and the receiver's parent has wsCaptureLayout on, wsLayoutDirty will be set on the receiver's parent.
**msgWinSetPaintable**

Turns window paintability bit on or off, returns previous value.

Takes BOOLEAN, returns BOOLEAN.

```c
#define msgWinSetPaintable MakeMsg(clsWin, 9)
```

**msgWinBeginRepaint**

Sets up window for painting on "dirty" region.

Takes P_RECT32 or pNull, returns STATUS.

```c
#define msgWinBeginRepaint MakeMsg(clsWin, 10)
```

**Comments**

A BeginRepaint/EndRepaint pair bracket an update episode for a window. They should be sent ONLY in response to the receipt of `msgWinRepaint`. If pArgs is not pNull a rectangle describing the bounds of the dirty region is passed back.

**msgWinEndRepaint**

Tells window system that repainting has ended for this window.

Takes nothing, returns STATUS.

```c
#define msgWinEndRepaint MakeMsg(clsWin, 11)
```

**msgWinBeginPaint**

Sets up window for painting on its visible region.

Takes P_RECT32 or pNull, returns STATUS.

```c
#define msgWinBeginPaint MakeMsg(clsWin, 12)
```

**Comments**

A BeginPaint/EndPaint pair can be used to paint on a window at anytime, even if it is not dirty. If pArgs is not pNull a rectangle describing the bounds of the visible region is passed back.

**msgWinEndPaint**

Tells window system that painting has ended for this window.

Takes nothing, returns STATUS.

```c
#define msgWinEndPaint MakeMsg(clsWin, 13)
```

**msgWinDirtyRect**

Marks all or part of a window dirty.

Takes P_RECT32 or pNull, returns STATUS.

```c
#define msgWinDirtyRect MakeMsg(clsWin, 14)
```

**Comments**

If pNull is passed the entire window is marked dirty. If the dirty part is visible, the window will eventually receive `msgWinRepaint` as a side effect of this message.
msgWinUpdate
Forces a window to repaint now, provided that it needs repainting.
Takes nothing, returns STATUS.
#define msgWinUpdate MakeMsg(clsWin, 35)
Comments
The window and all its descendants that need painting are sent msgWinRepaint. However, only windows owned by the current subtask are processed.

msgWinCleanRect
Marks all or part of a window clean.
Takes P_RECT32 or pNull, returns STATUS.
#define msgWinCleanRect MakeMsg(clsWin, 15)
Comments
If pNull is passed the entire window is marked clean. In general it is not a good idea to mark a window clean. Window activity is asynchronous and application software has no way of knowing if the window is really clean.

msgWinCopyRect
Copies pixels within a window.
Takes P_WIN_COPY_RECT, returns STATUS.
#define msgWinCopyRect MakeMsg(clsWin, 16)
Arguments
Enum16 (WIN_COPY_FLAGS)
{ wsCopyNormal = 0, // normal copy of normal planes
  wsPlanePen = flag0, // do pen plane(s) too
  wsPlaneMask = flag1, // use planeMask
  wsSrcNotDirty = flag2, // don’t mark source dirty
  wsDstNotDirty = flag3, // don’t mark dirty dst pixels dirty
  wsChildrenStay = flag4,
  wsCopyRelative = flag5, // xy is a delta on srcRect.origin
};
typedef struct
{ RECT32 srcRect; // rectangle in LWC
  XY32 xy; // new location in LWC
  WIN_COPY_FLAGS flags;
  U16 planeMask;
} WIN_COPY_RECT, * P_WIN_COPY_RECT;
Comments
In general, pixels which are dirty, invisible, or just off the edge of the window, are not copied. Rather, at the destination it is recognized that they did not get copied, and they are marked dirty instead. Also, it is assumed that pixels at the source need to be repainted. (This behavior is controlled by the two flags wsSrcNotDirty and wsDstNotDirty).

The intent of this message is that it be used as an accelerator; to move potentially good pixels to a new location. It should be sent OUTSIDE of an update episode. Then, areas that require repainting will be marked dirty and handled by the next update episode.

If, by mistake, this message is sent inside an update episode it will probably not copy any pixels, because it will assume that all the pixels that are currently being updated are dirty.
The use of wsCopyNormal is recommended to copy the normal planes and skip the pen plane(s). More
precise control over which planes are copied is available with the use of flags wsCopyNormal,
wsPlanePen and wsPlaneMask (in conjunction with the planeMask field).

If wsChildrenStay is not in in pArgs->flags and the receiver has children in the area being copied, the
children will be moved also. Note that even if the receiver has wsCaptureGeometry on, the receiver will
not be sent msgWinDeltaOK when the children are moved. However, each child that has
wsSendGeometry on and is moved will be sent msgWinMoved.

**msgWinTransformBounds**

Transforms bounds from receiver's to another window's LWC.

Takes P_WIN_METRICS, returns STATUS.

```c
#define msgWinTransformBounds MakeMsg(clsWin, 18)
typedef struct
{
  WIN parent,
  child;
  RECT32 bounds;
  WIN_DEV device;
  WIN_FLAGS flags;
  TAG tag;
  WIN_FLAGS options;
} WIN_METRICS, *P_WIN_METRICS;
```

**Comments**

Set the pArgs->parent to a window or use objNull for the receiver's actual parent. pArgs->bounds in the
receiver's LWC are transformed into the equivalent bounds in the parent's LWC.

**msgWinEnum**

Enumerate a window's children.

Takes P_WIN_ENUM, returns STATUS.

```c
#define msgWinEnum MakeMsg(clsWin, 33)
```

```c
Enum16(WIN_ENUM_FLAGS)
{
  wsEnumChildren = 0,  // enum children only
  wsEnumSelf = flag0,  // enum self too
  wsEnumRecursive = flag1, // enum children of children...
  wsEnumFlags = flag2,  // return flags too
  wsEnumBreadthFirst = flag3, // return flags too
  wsEnumSendFile = flag4,  // wsSendFile == TRUE
  wsEnumMetrics = flag5  // return WIN_METRICS
};
```

```c
typedef struct
{
  max,                   // in = size of pWin[] and pFlags[] arrays
  count;                 // in = # to return in arrays
  // if count > max then memory may be allocated
  // out = # of valid entries in arrays
  P_WIN pWin;
  P_WIN_FLAGS pFlags;    // in = ptr to arrays
  // out = if memory was allocated
  // client should free the memory
  U16 next;              // in = 0 to start at beginning
  // OR previous out value to pick up
  // where we left off
  WIN_ENUM_FLAGS flags;  // in = see options
} WIN_ENUM, *P_WIN_ENUM;
```
Here is some sample code for enumerating the direct children of a window:

```c
WIN_ENUM e;
WIN w[10];
U16 i;

// e.pWin is an array of 10 WINs
e.max = 10;
// allocate as much storage as needed
e.count = maxU16;
// put windows in w array
e.pWin = w;
// return only direct children
e.flags = wsEnumChildren;
// start from the first child

s = ObjectCall(msgWinEnum, parent, &e);
// stsEndOfData means we got them all
if (s == stsEndOfData)
  s = stsOK;
// e.count is the actual number of children
for (i = 0; i < e.count; i++) {
  child = e.pWin[i];
  // put code that does something with
  // child here
}
// free any allocated storage
if (e.pWin != w)
  StsWarn(OSHeapBlockFree(e.pWin));
```

If you want to retrieve all of the window metrics for each window, turn on `wsEnumMetrics` in `pArgs->flags` and set `pArgs->pWin` to an array of `WIN_METRICS` structs.

Returns

- `stsEndOfData` if all of the descendants have been returned

See Also

- `WinEachChild`

---

### WinEachChild

Helper macro for enumerating the direct children of a window

Returns nothing.

```c
#define WinEachChild(parent, child, s) \
{ \
  WIN_ENUM _e; \
  WIN _w[10]; \
  U16 _i; \
  \
  _e.max = 10; \
  _e.count = maxU16; \
  _e.pWin = _w; \
  _e.flags = wsEnumChildren; \
  _e.next = 0; \
  \
  s = ObjectCall(msgWinEnum, parent, &_e); \
  \
  if (s == stsEndOfData) \
    s = stsOK; \
  \
  for (_i = 0; _i < _e.count; _i++) \
  { \
    child = _e.pWin[_i]; \
    // put code that does something with \
    // child here
  }
}
```

You can use `WinEachChild` to retrieve the direct children of a window.

See Also

- `WinEndEachChild`
WinEndEachChild
Ending helper macro for most common window enumeration idiom.

Returns nothing.

```c
#define WinEndEachChild
} /* for */
if (_e.pWin != (P_WIN)_w)
    OSHeapBlockFree(_e.pWin);
} /* end scope */
```

WinEachChild and WinEndEachChild

Use WinEachChild(parent,child,status) to start a for loop enumeration of the children of parent. The variable child will be set for each child. Close the enumeration with WinEndEachChild. Here is an example; notice that semicolons are NOT used.

```c
WinEachChild(p,c,s) // send a message to c // break if necessary
    // s is set here
```

The code placed between these macros becomes the body of a for loop. If it is necessary to exit the loop early, use a break statement, not a return or goto, so that WinEndEachChild is reached. If an error in the enum occurs, the for loop will not be executed, and the status value will be set.

msgWinRepaint
Tells a window to repaint itself.

Takes nothing, returns STATUS. Category: descendant responsibility.

```c
#define msgWinRepaint
MakeMsg(clsWin, 21)
```

Comments

Windows only receive this if the wsPaintable flag is true. This message is sent by the window system during an update episode. It should NOT be sent by the application.

If you want a window to be updated immediately (synchronously), use msgWinUpdate.

Upon receipt of this message, applications should NOT perform other windowing operations that are visually significant (msgWinDelta, msgWinInsert, msgWinExtract, etc.). When this message is received; it is too late. The only thing that should happen is repainting.

See Also

msgWinBeginRepaint

msgWinOrphaned
Tells a window its parent has been freed.

Takes nothing, returns STATUS. Category: advisory message.

```c
#define msgWinOrphaned
MakeMsg(clsWin, 22)
```

Comments

Windows only receive this if the wsSendOrphaned flag is true.

msgWinInsertOK
Informs a potential parent of a pending child insertion.

Takes P_WIN_METRICS, returns STATUS. Category: advisory message.

```c
#define msgWinInsertOK
MakeMsg(clsWin, 23)
```
typedef struct {
    WIN parent,
    child;
    RECT32 bounds;
    WIN_DEV device;
    WIN_FLAGS flags;
    TAG tag;
    WIN_OPTIONS options;
} WIN_METRICS, * P_WIN_METRICS;

pArgs->child is the window that is being inserted; pArgs->bounds is its bounds, which the parent can modify. If receiver does not return stsOK the insertion will be denied.

Windows only receive this if wsCaptureGeometry in flags is true.

msgWinExtractOK

Informs parent of a pending child extraction.

Takes P_WIN_METRICS, returns STATUS. Category: advisory message.

#define msgWinExtractOK MakeMsg(clsWin, 24)

typedef struct {
    WIN parent,
    child;
    RECT32 bounds;
    WIN_DEV device;
    WIN_FLAGS flags;
    TAG tag;
    WIN_OPTIONS options;
} WIN_METRICS, * P_WIN_METRICS;

pArgs->child is the window that is being extracted. If receiver does not return stsOK the extraction will be denied.

Windows only receive this if wsCaptureGeometry in flags is true.

msgWinDeltaOK

Informs parent of a pending change in a child window’s size or position.

Takes P_WIN_METRICS, returns STATUS. Category: advisory message.

#define msgWinDeltaOK MakeMsg(clsWin, 25)

typedef struct {
    WIN parent,
    child;
    RECT32 bounds;
    WIN_DEV device;
    WIN_FLAGS flags;
    TAG tag;
    WIN_OPTIONS options;
} WIN_METRICS, * P_WIN_METRICS;

pArgs are the arguments to msgWinDelta. If receiver does not return stsOK the delta will be denied.

Windows only receive this if wsCaptureGeometry in flags is true.
msgWinFreeOK
Informs parent of the pending destruction of a child window.
Takes WIN, returns STATUS. Category: advisory message.

#define msgWinFreeOK MakeMsg(clsWin, 26)

Comments
Windows only receive this if wsCaptureGeometry in flags is true.

msgWinInserted
Advises window that it has been inserted.
Takes WIN, returns STATUS. Category: advisory message.

#define msgWinInserted MakeMsg(clsWin, 27)

Comments
pArgs is the window that actually was inserted, it may be self or an ancestor. If it is an ancestor, the window is being inserted indirectly, as part of a sub-tree insertion.
Windows only receive this if wsSendGeometry in flags is true.

msgWinExtracted
Advises window that it has been extracted.
Takes WIN, returns STATUS. Category: advisory message.

#define msgWinExtracted MakeMsg(clsWin, 28)

Comments
pArgs is the window that actually was extracted, it may be self or an ancestor. If it is an ancestor, the window is being extracted indirectly, as part of a sub-tree extraction.
Windows only receive this if wsSendGeometry in flags is true.

msgWinVisibilityChanged
Advises window that its visibility may have changed.
Takes WIN, returns STATUS. Category: advisory message.

#define msgWinVisibilityChanged MakeMsg(clsWin, 60)

Comments
pArgs is the window that actually was changed, it may be self or an ancestor. If it is an ancestor, the window is being made visible or invisible indirectly, as part of a sub-tree insertion or extraction.
Note that if pArgs is an ancestor, the ancestor’s visibility change may not have changed self’s visibility. Use msgWinIsVisible to determine self’s current visibility.
Windows only receive this if wsSendGeometry in flags is true.

See Also
msgWinIsVisible

msgWinMoved
Advises window that it, or an ancestor, has moved.
Takes P_WIN_METRICS, returns STATUS. Category: advisory message.

#define msgWinMoved MakeMsg(clsWin, 29)
typedef struct
{
    WIN parent,
    child;
    RECT32 bounds;
    WIN_DEV device;
    WIN_FLAGS flags;
    TAG tag;
    WIN_OPTIONS options;
} WIN_METRICS, *P_WIN_METRICS;

Windows only receive this if wsSendGeometry in flags is true. pArgs->bounds.origin is the previous position. pArgs->child is the window that actually moved, it may be self or an ancestor. If it is an ancestor, the window is being moved indirectly, as part of a sub-tree move.

**msgWinSized**
Advises window that it, or an ancestor, has changed size.

Takes P_WIN_METRICS, returns STATUS. Category: advisory message.

#define msgWinSized MakeMsg(clsWin, 30)

Windows only receive this if wsSendGeometry in flags is true. pArgs->bounds.size is the previous size. pArgs->child is the window that actually changed size, it may be self or an ancestor. If it is an ancestor, the window did not actually change size, the ancestor did.

**msgWinStartPage**
Advises window that it is on a printer, and printing is about to commence.

Takes pNull, returns STATUS. Category: advisory message.

#define msgWinStartPage MakeMsg(clsWin, 48)

clsWin does nothing and returns stsOK in response to this message.

This message is sent before a page is about to be printed. The window may want to set a state variable used to change the way the window paints on a printer.

**msgWinSort**
Sorts a window's children into a back to front order determined by a client supplied comparison function.

Takes P_WIN_SORT, returns STATUS.

#define msgWinSort MakeMsg(clsWin, 52)
typedef struct
{
    P_WIN_SORT_PROC pSortProc; // In: comparison callback
    P_UNKNOWN pClientData;    // In: parameter to callback
    BOOLEAN changed;         // Out: did sort cause change in order
} WIN_SORT, *P_WIN_SORT;

The client must create a function of the profile P_WIN_SORT_PROC that takes two windows (A,B) and returns -1 if A < B, 0 if A == B, and +1 if A > B. The comparison will normally be based on information retrieved from the windows (for instance, msgLabelGetString).

### msgWinGetEnv

Gets the current window environment.

Takes P_WIN_ENV, returns STATUS.

```
#define msgWinGetEnv MakeMsg(clsWin, 47)
```

#### Arguments

typedef struct WIN_ENV
{
    U8    scale;     // system font scale
    U16   sysFontId, // system font
           userFontId; // user font
    SIZE32 ppm;      // device pixels per meter
} WIN_ENV, *P_WIN_ENV;

typedef struct WIN_SAVE_ENV
{
    WIN_ENV env;     // environment being saved
    U32    spare1;
    U32    spare2;
} WIN_SAVE_ENV, *P_WIN_SAVE_ENV;

typedef struct WIN_RESTORE_ENV
{
    WIN_ENV env;     // the saved environment
    BOOLEAN scaleChanged,
           sysFontIdChanged, // these are true if the current
           userFontIdChanged, // environment has changed from
           ppmWChanged,
           ppmHChanged;
    U32    spare1;
    U32    spare2;
} WIN_RESTORE_ENV, *P_WIN_RESTORE_ENV;

The window environment is information filed with the root of each filed tree of windows.

This message would not normally be used by application software.

### msgWinDumpTree

In lieu of msgDump. Dumps a dense subset of information for the window and all it's children recursively.

Takes pNull, returns STATUS.

```
#define msgWinDumpTree MakeMsg(clsWin, 51)
```

#### Comments

Debug /DW 2 causes the input flags to be printed, otherwise the window flags are printed.

This function may not work unless the debugging version of win.dll is being used.
**msgWinHitDetect**

Locates the window "under" a point.

Takes P_WIN_METRICS, returns STATUS.

```
#define msgWinHitDetect MakeMsg(clsWin, 58)
```

typedef struct {
    WIN parent,
    child;
    RECT32 bounds;
    WIN_DEV device;
    WIN_FLAGS flags;
    TAG tag;
    WIN_OPTIONS options;
} WIN_METRICS, * P_WIN_METRICS;

Comments

pArgs->bounds.origin is a point relative to the receiver. The window tree, starting with the root window, is searched for a window underneath this point. The result is returned in pArgs->child.

If the search is NOT successful, pArgs->child will be objNull.

---

**Messages from other classes**

**msgSave**

Causes an object to file itself in an object file.

Takes P_OBJ_SAVE, returns STATUS.

Comments

clsWin will save its instance data and file each direct child that has wsSendFile on.

If pArgs->root is self, clsWin will file the window environment along with its instance data. The window environment is retrieved by self-sending msgWinGetEnv. If pArgs->pEnv is not pNull, the current environment info (WIN_SAVE_ENV) will be copied to the storage provided (pArgs->pEnv should either be pNull or a P_WIN_SAVE_ENV). Subclasses of clsWin can make use of pArgs->pEnv to look at the environment under which the window is being saved. The filed window environment will be used during msgRestore to adjust the window bounds and/or dirty the window layout if the restore environment is not the same as the saved environment.

If wsFileNoBounds is on in self's window style flags, the current bounds will not be filed. This will save space in the filed window.

If self's desired size has been computed (via msgWinGetDesiredSize during msgWinLayout processing), the desired size will be filed.

For each child of self that has wsSendFile on, clsWin will do the following:

If wsFileInline is on in the child's window style flags, the class of the child window will be filed, and then the child will be sent msgSave with the following OBJ_SAVE parameters:

```
all fields as in *pArgs,
objSave = pointer to current save environment
```

This will file the child "inline" without the usual resource file object header. This will save storage, but the child will not have its own resid and can only be restored by restoring its parent.
If wsFileInline is not on in the child's window style flags, the child window will be filed by sending msgResPutObject to pArgs->file with the child's uid as pArgs.

Returns

stsWinNoEnv if pArgs->root != self and pArgs->pEnv is pNull

See Also

msgRestore

**msgRestore**

Creates and restores an object from an object file.

Takes P_OBJ_RESTORE, returns STATUS.

**clsWin** will restore its instance data from pArgs->file. Each filed child window will also be restored. The window will be created on the window device returned from OSThisWinDev().

If the window environment was filed when the window was saved, the window environment will be restored and copied to pArgs->pEnv if it is not pNull (pArgs->pEnv must be either pNull or P_WIN_RESTORE_ENV). The current window environment will be retrieved using msgWinGetEnv and compared to the filed window environment.

If wsFileNoBounds is on in self's window style flags, the bounds will be set to (0, 0, 0, 0) and the window will be marked as layout-dirty (wsLayoutDirty will be or-ed into the window's style flags). Otherwise, the filed bounds will be restored and adjusted to compensate for differences in the save/restore-time device resolution and orientation.

**clsWin** will or-in wsLayoutDirty into the window's style flags if any of the following are true (in this context "changed" means that the current window environment values do not match the window environment filed with the window tree):

- wsFileLayoutDirty is on in the window's style flag
- system font or system font scale has changed
- user font has changed
- pixels-per-meter in x or y have changed

Each child that was filed will be restored as follows:

If wsFileInline was on in the child's window style flags, the child's class will be read in from pArgs->file and msgRestoreInstance will be sent to the class with the following OBJ_RESTORE parameters:

- all fields as in *pArgs
- object = msgNewDefaults to clsObject
- object.key = pArgs->object.key;
- object.cap = pArgs->object.cap;
- object.heap = pArgs->object.heap;

If wsFileInline was not on in the child's window style flags, the child's resId will be read in from pArgs->file and the child will be restored by sending msgResReadObject to pArgs->file with the following RES_READ_OBJECT parameters:

- mode = resReadObjectOnce;
- objectNew = same as object in wsFileInline case above

After all of the children have been restored, they will be inserted into the restored parent. Note that the wsCaptureGeometry and wsSendGeometry protocol is not used for these inserts (e.g. the parent will not be sent msgWinInsertOK, even if the parent has wsCaptureGeometry on).

See Also

msgSave
**Messages Sent to a Window Device**

As a rule applications should not send these messages to the `Screen`. They would be used if the application creates image devices.

### msgNew

Creates a windowing device.

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

**Arguments**

typedef struct
{
    UI16 initialWindows; // default window slots to allocate
} WIN_DEV_NEW_ONLY, *P_WIN_DEV_NEW_ONLY;

typedef struct
{
    OBJECT_NEW        object;
    WIN_DEV_NEW_ONLY  winDev;
} WIN_DEV_NEW, *P_WIN_DEV_NEW,

**Message**

typedef struct
{
    OBJECT_NEW        object;
    WIN_DEV_NEW_ONLY  winDev;
} WIN_DEV_NEW, *P_WIN_DEV_NEW,

**win.Dev.initialWindows = 100;**

### msgNewDefaults

Initializes the `WIN_DEV_NEW` structure to default values.

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

**Arguments**

typedef struct
{
    OBJECT_NEW        object;
    WIN_DEV_NEW_ONLY  winDev;
} WIN_DEV_NEW, *P_WIN_DEV_NEW,

### msgWinDevGetRootWindow

Passes back root window for receiver.

Takes `P_OBJECT`, returns `STATUS`.

#define msgWinDevGetRootWindow MakeMsg(clsWinDev, 10)

### msgWinDevBindScreen

Binds window device to a screen.

Takes `P_CHAR`, returns `STATUS`.

#define msgWinDevBindScreen MakeMsg(clsWinDev, 6)

### msgWinDevBindPrinter

Binds window device to an object of `clsPrn`.

Takes `OBJECT`, returns `STATUS`.

#define msgWinDevBindPrinter MakeMsg(clsWinDev, 7)
msgWinDevBindPixelmap
Binds window device to a pixelmap.
Takes P_WIN_DEV_PIXELMAP, returns STATUS.

#define msgWinDevBindPixelmap MakeMsg(clsWinDev, 11)

Comments
Note that you should not file the memory allocated by msgWinDevBindPixelmap, since the memory is
device-dependant and you may be restored on a different screen device or system processor.

msgWinDevSizePixelmap
Computes the amount of memory needed for a single plane.
Takes P_WIN_DEV_PIXELMAP, returns STATUS.

#define msgWinDevSizePixelmap MakeMsg(clsWinDev, 12)

typedef struct
{
  OBJECT            device;  // in = device to be "compatible" with
  SIZE32            size;    // in = w,h of device to allocate
  U16               planeCount;  // in = # planes to allocate
  SIZEOF            planeSize;  // out = amount of memory for one plane
  PP_UNKNOWN        pPlanes;  // in = plane memory
} WIN_DEV_PIXELMAP, * P_WIN_DEV_PIXELMAP;

msgWinDevSetOrientation
Changes orientation of a window device.
Takes PIX_DEV_ORIENT, returns STATUS.

#define msgWinDevSetOrientation MakeMsg(clsWinDev, 8)

Enum16(PIX_DEV_ORIENT)
{
  pdUL        = 0,
  pdUR        = 1,
  pdLR        = 2,
  pdLL        = 3,
  pdOrientLandscapeNormal = pdLL,
  pdOrientPortraitNormal = pdUL,
  pdOrientLandscapeReverse = pdUR,  // not supported on printers
  pdOrientPortraitReverse = pdLR    // not supported on printers
};

msgPixDevGetMetrics
Gets metrics of a pixelmap device.
Takes P_PIX_DEV_METRICS, returns nothing.

#define msgPixDevGetMetrics MakeMsg(clsPixDev, 1)

typedef struct
{
  SIZE32              size,  // size of device in DU4
  ppm;               // pixel per meter in DU4
  U16                 planes;  // # of planes total
  U16                 planeMask;  // mask representing all planes
  U16                 planeNormalCount;  // # of normal (not pen) planes
  U16                 planeNormalMask;  // mask for the normal planes
  U16                 planePenCount;  // # of pen planes
  U16                 planePenMask;  // mask for the pen planes
}
WIN.H

Messages sent to a drawing context

msgWinDevPrintPage
Repaints and outputs a page.
Takes nothing, returns STATUS.
#define msgWinDevPrintPage MakeMsg(clsWinDev, 9)

Messages sent to a drawing context

msgDrwCtxSetWindow
Binds a drawing context to a window, returns old window.
Takes WIN, returns WIN.
#define msgDrwCtxSetWindow MakeMsg(clsDrwCtx, 3)

msgDrwCtxGetWindow
Returns the window to which a drawing context is bound.
Takes nothing, returns WIN.
#define msgDrwCtxGetWindow MakeMsg(clsDrwCtx, 4)
#endif // WIN_INCLUDED
Part 4 / UI Toolkit
This file contains the API for clsBorder. clsBorder inherits from clsEmbeddedWin. clsBorder supports drawing borders, backgrounds and shadows. Support is also provided for resize, drag and top window management.

```c
#ifndef BORDER_INCLUDED
#define BORDER_INCLUDED

#include <win.h>
#endif

#ifndef WIN_INCLUDED
#define WIN_INCLUDED
#endif

#ifndef EWNEW_INCLUDED
#define EWNEW_INCLUDED
#endif

#ifndef INPUT_INCLUDED
#define INPUT_INCLUDED
#endif

Common #defines and typedefs

#define hlpBorderResizeBottom MakeTag(clsBorder, 1)
#define hlpBorderResizeCorner MakeTag(clsBorder, 2)
#define hlpBorderResizeRight MakeTag(clsBorder, 3)

typedef OBJECT BORDER;

Edge Styles

#define bsEdgeNone 0     // no borders
#define bsEdgeLeft flag0 // border on the left
#define bsEdgeRight flag1 // border on the right
#define bsEdgeTop flag2 // border on the top
#define bsEdgeBottom flag3    // border on the bottom

#define bsEdgeAll (bsEdgeLeft | bsEdgeRight | \n               bsEdgeTop | bsEdgeBottom)

Join Styles

#define bsJoinSquare 0 // right-angle rectangle
#define bsJoinRound  1 // round corner rectangle
#define bsJoinEllipse 2 // ellipse instead of rectangle
#define bsJoinReserved 3 // unused (reserved)

Line Styles

#define bsLineSingle 0 // solid ink
#define bsLineDouble  1 // ink-white-ink lines
#define bsLineMarquee 2 // flowing dashed lines
#define bsLineDashed  3 // dashed lines
#define bsLineDoubleMarquee 4 // double flowing dashed lines
#define bsLineDoubleDashed 5 // double dashed lines
#define bsLineDoubleDashed 6 // unused (reserved)
#define bsLineMarquee15 // unused (reserved)
```
# Edge and Background Colors

```c
#define bsInkTransparent 0 // no ink
#define bsInkBlack 1 // black
#define bsInkGray75 2 // 75% gray
#define bsInkGray66 3 // 66% gray
#define bsInkGray50 4 // 50% gray
#define bsInkGray33 5 // 33% gray
#define bsInkGray25 6 // 25% gray
#define bsInkWhite 7 // white
#define bsInkAsIs 8 // use appropriate dc value
#define bsInkRGB 9 // use custom RGB value
#define bsInkBackground 10 // use the background ink
// 11 // unused (reserved)
// .. // unused (reserved)
// 31 // unused (reserved)
```

`bsInkExclusive` can be or'ed into any ink to indicate that the specified ink should only be used if the window exclusively paints its pixels. If the window is transparent or shares clipping with its parent, `bsInkTransparent` will be used (i.e. nothing will be painted).

```c
#define bsInkExclusive flag4
```

`BorderInk` extracts the base ink from a border ink

```c
#define BorderInk(ink) ((ink) & 0xF)
```

# Shadow Styles: drawn on the bottom and right

```c
#define bsShadowNone 0 // no shadow
#define bsShadowThinGray 1 // one line gray
#define bsShadowThickGray 2 // two line gray
#define bsShadowThinBlack 3 // one line black
#define bsShadowThickBlack 4 // two line black
#define bsShadowThinWhite 5 // one line white
#define bsShadowThickWhite 6 // two line white
#define bsShadowCustom 7 // use shadowThickness and shadowInk
// 8 // unused (reserved)
// .. // unused (reserved)
// 15 // unused (reserved)
```

# Units

```c
#define bsUnitsLayout 0 // values are in layout units
#define bsUnitsDevice 1 // values are in device units
#define bsUnitsTwips 2 // values are in twips
#define bsUnitsPoints BorderUnitsCustom(bsUnits20x, bsUnitsTwips)
// values are in points = 20 x twips
#define bsUnitsRules 3 // values are in rules
#define bsUnitsLines BorderUnitsCustom(bsUnits20x, bsUnitsRules)
// values are in lines = 20 x rules
#define bsUnitsMetric 4 // values are in .01 mm
#define bsUnitsMIL 5 // values are in .001 inch
#define bsUnitsFitWindow 6 // values not specified --
// compute to fit window
#define bsUnitsFitWindowProper 7 // values not specified --
// compute to fit window w/proper // aspect ratio
// 8 // unused (reserved)
// .. // unused (reserved)
// 15 // unused (reserved)
```
# Units Multiplier
These values can be used with BorderUnitsCustom() to produce new units e.g.
BorderUnitsCustom(bsUnits20x, bsUnitsTwips) indicates units are 20 x twips

```c
#define bsUnits1x     0 // 1x
#define bsUnits20x    1 // 20x
#define bsUnits100x   2 // 100x
#define bsUnits1000x  3 // 1000x
```

```c
#define BorderUnitsCustom(mult, units) ((mult << 4) | (units))
```

macros to extract base units and multiplier values

```c
#define BorderUnits(units)    ((units) & 0x0F)
#define BorderUnitsMult(units) ((units) >> 4)
```

## Common Margin Values

```c
#define bsMarginNone      0 // no inner margin
#define bsMarginSmall     1 // 1 unit
#define bsMarginMedium   2 // 2 units
#define bsMarginLarge    8 // 8 units
```

## Resize Handles

```c
#define bsResizeNone       0 // no resize handles
#define bsResizeCorner     flag0 // lower-right corner
#define bsResizeBottom     flag1 // center-bottom
#define bsResizeRight      flag2 // center-right
#define bsResizeAll        (bsResizeCorner | bsResizeBottom | \ bsResizeRight)
```

## Drag Styles

```c
#define bsDragNone         0 // no drag
#define bsDragHoldDown     1 // drag on penHoldDown
#define bsDragDown         2 // drag on penDown
#define bsDragMoveDown     3 // drag on penMoveDown beyond range
```

## Top Styles

```c
#define bsTopNone          0 // never top the window
#define bsTopUp             1 // top on penUp
#define bsTopDrag           2 // top after drag
```

## Shadow Gap Styles

```c
#define bsGapNone          0 // no shadow gap
#define bsGapWhite         1 // cleared to white
#define bsGapTransparent   2 // unpainted
```

## Look Styles

```c
#define bsLookActive       0 // usually black foreground color
#define bsLookInactive     1 // usually gray66 foreground color
```

```c
```
# define bsAlterNone 0 // don’t alter anything
# define bsAlterBackground 1 // alter the background ink
# define bsAlterBorders 2 // alter the border ink

// unused (reserved)
typedef struct BORDER_STYLE {
    U16 edge : 4, // edges to border
top : 2, // top style (e.g. bsTopUp)
drag : 2, // drag style (e.g. bsDragDown)
resize : 5, // resize handles (e.g. bsResizeCorner|bsResizeBottom)
maskBorders : 1, // mask out edge, shadow, resize
getDeltaWin : 1, // use msgBorderProvideDeltaWin
spare1 : 1; // unused (reserved)
    U16 leftMargin : 8, // margin in marginUnits
rightMargin : 8; // *
    U16 bottomMargin : 8, // *
topMargin : 8; // *
    U16 borderInk : 6, // edge line color
backgroundInk : 6, // background fill color
previewAlter : 2, // what to alter when previewing
selectedAlter : 2; // what to alter when selected
    U16 marginInk : 6, // ink for margin area (not implemented)
marginUnits : 6, // units for left, right, bottom, top margins
    U16 preview : 1, // true/false
 selected : 1, // true/false
look : 2; // active/inactive
    U16 shadow : 4, // type of shadow
shadowGap : 2, // type of shadow gap
shadowThickness : 8, // custom shadow thickness, in lineUnits
spare4 : 2; // unused (reserved)
    U16 shadowInk : 6, // custom shadow ink
lineStyle : 4, // edge line style (e.g. bsLineSingle)
spare5 : 6; // unused (reserved)
    U16 lineUnits : 6, // units for lineThickness and shadowThickness
lineThickness : 8, // line thickness, in lineUnits
join : 2; // how edges join together
    U16 propagateVisuals : 1, // propagate changes in visuals to children
notifyVisuals : 1, // send msgBorderSetVisuals to observers
spare3 : 14; // unused (reserved)
} Borders_STYLE, *P_BORDER_STYLE;

Default BORDER_STYLE:

    edge = bsEdgeNone
join = bsJoinSquare
lineStyle = bsLineSingle
marginUnits = bsUnitsLayout
resize = bsResizeNone
move = bsDragNone
top = bsTopNone
leftMargin = bsMarginNone
rightMargin = bsMarginNone
bottomMargin = bsMarginNone
topMargin = bsMarginNone
look = bsLookActive
preview = false
selected = false
propagateVisuals = false
notifyVisuals = false
getDeltaWin = false
maskBorders = false
borderInk = bsInkBlack
backgroundInk = bsInkWhite
marginInk = bsInkBackground
shadow = bsShadowNone
shadowGap = bsGapWhite
shadowInk = bsInkBlack
shadowThickness = 1
lineUnits = bsUnitsLines
lineThickness = 1
previewAlter = bsAlterNone
selectedAlter = bsAlterNone

Input event flags returned in INPUT_EVENT.flags indicates event was used to move/resize

#define evBorderTaken evFlag0

Tags used by resize or drag tracks. These will be the tags in TRACK_METRICS of msgTrackProvideMetrics and msgTrackDone.

#define tagBorderResize MakeTag(clsBorder, 4)
#define tagBorderDrag MakeTag(clsBorder, 5)

**msgNew**

Creates a border window.

Takes P_BORDER_NEW, returns STATUS. Category: class message.

```c
typedef struct BORDER_NEW ONLY {
    BORDER_STYLE style; // overall style
    U32 spare1; // unused (reserved)
    U32 spare2; // unused (reserved)
} BORDER_NEW ONLY, BORDER_METRICS,
*P_BORDER_NEW ONLY, *P_BORDER_METRICS;
#define borderNewFields \
    embeddedWinNewFields \
    BORDER_NEW ONLY border;
```

**Comments**

If `pArgs->border.style.maskBorders` is true, style.resize is treated as though it is `bsResizeNone`, style.edge is treated as though it is `bsEdgeNone`, and style.shadow is treated as though it is `bsShadowNone`.

If `pArgs->style.resize` is not `bsResizeNone`, `pArgs->win.flags.input` is altered to enable events needed for resizing.

If `pArgs->style.drag` is not `bsDragNone`, `pArgs->win.flags.input` is altered to enable events needed for dragging.

If `pArgs->style.top` is not `bsTopNone`, `pArgs->win.flags.input` is altered to enable events needed for topping.

**msgNewDefaults**

Initializes the BORDER_NEW structure to default values.

Takes P_BORDER_NEW, returns STATUS. Category: class message.

```c
typedef struct {
    borderNewFields
} BORDER_NEW, *P_BORDER_NEW;
```

**Comments**

Zeroes out `pNew->border` and sets...

```c
pArgs->win.flags.style |= wsSendFile;
```
msgBorderGetStyle

Passes back the current style values.

Takes P_BORDER_STYLE, returns STATUS.

```c
#define msgBorderGetStyle MakeMsg(clsBorder, 1)
```

```c
typedef struct BORDER_STYLE {
    U16 edge : 4,  // edges to border
    top : 2,      // top style (e.g. bsTopUp)
    drag : 2,     // drag style (e.g. bsDragDown)
    resize : 5,   // resize handles (e.g. bsResizeCorner|bsResizeBottom)
    maskBorders : 1,  // mask out edge, shadow, resize
    getDeltaWin : 1,  // use msgBorderProvideDeltaWin
    spare1 : 1,    // unused (reserved)
    leftMargin : 8, // margin in marginUnits
    rightMargin : 8, //
    topMargin : 8,  //
    bottomMargin : 8, //
    borderlnk : 6,  // edge line color
    backgroundlnk : 6, // background fill color
    previewAlter : 2,  // what to alter when previewing
    selectedAlter : 2;  // what to alter when selected
    marginlnk : 6, // marginlnk
    marginUnits : 6, // units for left, right, bottom, top margins
    preview : 1,    // true/false
    selected : 1,   // true/false
    look : 2;       // active/inactive
    shadow : 4,     // type of shadow
    shadowGap : 2,  // type of shadow gap
    shadowThickness : 8, // custom shadow thickness, in lineUnits
    spare4 : 2;     // unused (reserved)
    shadowlnk : 6,  // custom shadow ink
    lineStyle : 4,  // edge line style (e.g. bsLineSingle)
    spare5 : 6;     // unused (reserved)
    lineUnits : 6,  // units for lineThickness and shadowThickness
    lineThickness : 8, // line thickness, in lineUnits
    join : 2;       // how edges join together
    propagateVisuals : 1, // propagate changes in visuals to children
    notifyVisuals : 1; // send msgBorderSetVisuals to observers
    spare3 : 14;    // unused (reserved)
} BORDER_STYLE, *P_BORDER_STYLE;
```

msgBorderSetStyle

Sets all of the style values.

Takes P_BORDER_STYLE, returns STATUS.

```c
#define msgBorderSetStyle MakeMsg(clsBorder, 2)
```
typedef struct BORDER_STYLE {
    U16 edge : 4;  // edges to border
    top : 2;     // top style (e.g. bsTopUp)
    drag : 2;    // drag style (e.g. bsDragDown)
    resize : 5;  // resize handles (e.g. bsResizeCorner|bsResizeBottom)
    maskBorders : 1;  // mask out edge, shadow, resize
    getDeltaWin : 1;  // use msgBorderProvideDeltaWin
    spare1 : 1;     // unused (reserved)
    U16 leftMargin : 8;  // margin in marginUnits
    rightMargin : 8;   // "
    U16 bottomMargin : 8; // "
    topMargin : 8;    // "
    U16 borderInk : 6;  // edge line color
    backgroundInk : 6; // background fill color
    previewAlter : 2; // what to alter when previewing
    selectedAlter : 2; // what to alter when selected
    U16 marginInk : 6;  // ink for margin area (not implemented)
    marginUnits : 6;  // units for left, right, bottom, top margins
    preview : 1;  // true/false
    selected : 1;  // true/false
    look : 2;   // active/inactive
    U16 shadow : 4;   // type of shadow
    shadowGap : 2;  // type of shadow gap
    shadowThickness : 8;  // custom shadow thickness, in lineUnits
    spare4 : 2;     // unused (reserved)
    U16 lineStyle : 4;  // edge line style (e.g. bsLineSingle)
    spare5 : 6;     // unused (reserved)
    U16 lineUnits : 6;  // units for lineThickness and shadowThickness
    lineThickness : 8;  // line thickness, in lineUnits
    join : 2;    // how edges join together
    U16 propagateVisuals : 1;  // propagate changes in visuals to children
    notifyVisuals : 1;  // send msgBorderSetVisuals to observers
    spare3 : 14;  // unused (reserved)
} BORDER_STYLE, *P_BORDER_STYLE;

Self-sends msgWinDirtyRect(pNull) if painting styles change. If only the edge painting style changes, self-sends msgWinDirtyRect with pArgs specifying the rectangle around each border.

Self-sends msgWinSetLayoutDirty(true), if new style results in new layout.

If style.propagateVisuals is true, and propagateVisuals or any of the visual styles (look, backgroundInk, previewAlter, selectedAlter, preview, or selected) change, msgBorderSetVisuals(pArgs) is sent to each child of self.

If style.notifyVisuals is true and notifyVisuals or any of the visual styles change, msgNotifyObservers is self-sent with the following OBJ_NOTIFY_OBSERVERS parameters:

msg = msgBorderSetVisuals;
pArgs = pointer to new style struct;
lenSend = SizeOf(BORDER_STYLE);

**msgBorderSetStyleNoDirty**

Sets all of the style values.

Takes P_BORDER_STYLE, returns STATUS.

#define msgBorderSetStyleNoDirty    MakeMsg(clsBorder, 26)
typedef struct BORDER_STYLE {
    U16 edge : 4,   // edges to border 
    top : 2,       // top style (e.g. bsTopUp)
    drag : 2,      // drag style (e.g bsDragDown)
    resize : 5,    // resize handles (e.g. bsResizeCorner|bsResizeBottom)
    maskBorders : 1, // mask out edge, shadow, resize
    getDeltaWin : 1, // use msgBorderProvideDeltaWin
    spare : 1;     // unused (reserved)
    U16 leftMargin : 8,   // margin in marginUnits
    rightMargin : 8;      //"
    U16 bottomMargin : 8,  //"
    topMargin : 8;        //"
    U16 borderInk : 6,     // edge line color
    backgroundInk : 6,    // background fill color
    previewAlter : 2,     // what to alter when previewing
    selectedAlter : 2;    // what to alter when selected
    U16 marginInk : 6,     // ink for margin area (not implemented)
    marginUnits : 6,      // units for left, right, bottom, top margins
    preview : 1,          // true/false 
    selected : 1,         // true/false 
    look : 2;             // active/inactive
    U16 shadow : 4,       // type of shadow
    shadowGap : 2,        // type of shadow gap
    shadowThickness : 8,  // custom shadow thickness, in lineUnits
    spare2 : 2;           // unused (reserved)
    U16 shadowInk : 6,    // custom shadow ink
    lineStyle : 4,        // edge line style (e.g. bsLineSingle)
    spare5 : 6;           // unused (reserved)
    U16 lineUnits : 6,    // units for lineThickness and shadowThickness
    lineThickness : 8,    // line thickness, in lineUnits
    join : 2;             // how edges join together
    U16 propagateVisuals : 1, // propagate changes in visuals to children
    notifyVisuals : 1,    // send msgBorderSetVisuals to observers
    spare3 : 14;}         // unused (reserved)

This message is the same as msgBorderSetStyle, except msgWinDirtyRect or msgWinSetLayoutDirty will not be self-sent, even if they new style parameters require repaint or relayout.

---

**msgBorderGetLook**

Passes back value of style.look.

Takes P_U16, returns STATUS.

#define msgBorderGetLook MakeMsg(clsBorder, 13)

---

**msgBorderSetLook**

Sets style.look as in msgBorderSetStyle.

Takes U16 (bsLook...), returns STATUS.

#define msgBorderSetLook MakeMsg(clsBorder, 12)

---

**msgBorderSetPreview**

Sets style.preview as in msgBorderSetStyle.

Takes BOOLEAN, returns STATUS.

#define msgBorderSetPreview MakeMsg(clsBorder, 8)
msgBorderGetPreview
Passes back value of style.preview.
Takes P_BOOLEAN, returns STATUS.
#define msgBorderGetPreview MakeMsg(clsBorder, 9)

msgBorderSetSelected
Sets style.selected as in msgBorderSetStyle.
Takes BOOLEAN, returns STATUS.
#define msgBorderSetSelected MakeMsg(clsBorder, 16)

msgBorderGetSelected
Passes back value of style.selected.
Takes P_BOOLEAN, returns STATUS.
#define msgBorderGetSelected MakeMsg(clsBorder, 17)

msgBorderPropagateVisuals
Propagates visuals to children.
Takes nothing, returns STATUS.
#define msgBorderPropagateVisuals MakeMsg(clsBorder, 15)

Comments
Sends msgBorderSetVisuals(&style), where style is self's current style, to each child.

msgBorderSetDirty
Sends msgBorderSetDirty(pArgs) to each child.
Takes BOOLEAN, returns STATUS.
#define msgBorderSetDirty MsgNoError(MakeMsg(clsBorder, 37))

Comments
clsBorder will pass this message along to each of its children. Child windows can alter their visuals to display a clean/dirty look. For example, clsControl will self-send msgControlSetDirty(pArgs) when receiving this message.

msgBorderGetDirty
Passes back true if any child responds to msgBorderGetDirty with true; otherwise passes back false.
Takes P_BOOLEAN, returns STATUS.
#define msgBorderGetDirty MsgNoError(MakeMsg(clsBorder, 38))

Comments
clsBorder will pass this message along to each of its children. The first child that responds with true will result in an answer of true. If no children are dirty, or there are no children, false will be returned. This message can be used to check the overall dirty/clean visual state of a tree of border windows. clsControl will respond by passing back the value of visual dirty bit, style.dirty.
**msgBorderGetForegroundRGB**

Passes back foreground RGB to use given current visuals.

Takes P_SYSDC_RGB, returns STATUS.

```
define msgBorderGetForegroundRGB MakeMsg(clsBorder, 27)
```

Comments: Subclasses should use this message to determine the correct foreground color to use. For example, clsLabel will self-send msgBorderGetForegroundRGB in its response to msgWinRepaint to make sure and get the correct foreground color.

**msgBorderGetBackgroundRGB**

Passes back background RGB to use given current visuals.

Takes P_SYSDC_RGB, returns STATUS.

```
define msgBorderGetBackgroundRGB MakeMsg(clsBorder, 28)
```

**msgBorderInkToRGB**

Maps ink value (bsInkGray66, etc.) to RGB.

Takes P_SYSDC_RGB, returns STATUS.

```
define msgBorderInkToRGB MakeMsg(clsBorder, 29)
```

Comments: For example, bsInkGray66 maps to sysDcRGBGray66.

**msgBorderRGBToInk**

Maps RGB value to ink (bsInkGray66, etc).

Takes P_SYSDC_RGB, returns STATUS.

```
define msgBorderRGBToInk MakeMsg(clsBorder, 30)
```

Comments: For example, sysDcRGBGray66 maps to bsInkGray66.

If pArgs has no matching ink value, bsInkTransparent is passed back.

**msgBorderConvertUnits**

Category: class or instance message

Converts values from one unit to another.

Takes P_BORDER_UNITS, returns STATUS.

```
define msgBorderConvertUnits MakeMsg(clsBorder, 39)
```

**Arguments**

```
typedef struct BORDER_UNITS {
    WIN win;  // in: window on target device
    U16 fromUnits;  // in: units for initial size.w/h
    U16 toUnits;  // in: units for final size.w/h
    SIZE32 size;  // in/out: initial/converted value
    U32 spare;  // unused (reserved)
} BORDER_UNITS, *P_BORDER_UNITS;
```

Comments: This message can be sent to clsBorder or an instance of clsBorder. clsBorder will convert pArgs->size from pArgs->fromUnits to pArgs->toUnits. If bsUnitsDevice is specified, pArgs->win should be set to a window on the corresponding device.
msgBorderSetVisuals

Sets only the visual fields from pArgs.

Takes P_BORDER_STYLE, returns STATUS.

```
#define msgBorderSetVisuals
typedef struct BORDER_STYLE {
    U16 edge : 4, // edges to border
    top : 2, // top style (e.g. bsTopUp)
    drag : 2, // drag style (e.g. bsDragDown)
    resize : 5, // resize handles (e.g. bsResizeCorner|bsResizeBottom)
    maskBorders : 1, // mask out edge, shadow, resize
    getDeltaWin : 1, // use msgBorderProvideDeltaWin
    spare1 : 1; // unused (reserved)
    U16 leftMargin : 8, // margin in marginUnits
    rightMargin : 8; //
    U16 bottomMargin : 8, //
    topMargin : 8; //
    U16 borderlnk : 6, // edge line color
    backgroundlnk : 6, // background fill color
    previewAlter : 2, // what to alter when previewing
    selectedAlter : 2; // what to alter when selected
    U16 marginlnk : 6, // ink for margin area (not implemented)
    marginUnits : 6, // units for left, right, bottom, top margins
    preview : 1, // true/false
    selected : 1, // true/false
    look : 2; // active/inactive
    U16 shadow : 4, // type of shadow
    shadowGap : 2, // type of shadow gap
    shadowThickness : 8, // custom shadow thickness, in lineUnits
    spare4 : 2; // unused (reserved)
    U16 shadowlnk : 6, // custom shadow ink
    lineStyle : 4, // edge line style (e.g. bsLineSingle)
    spare5 : 6; // unused (reserved)
    U16 lineUnits : 6, // units for lineThickness and shadowThickness
    lineThickness : 8, // line thickness, in lineUnits
    join : 2; // how edges join together
    U16 propagateVisuals : 1, // propagate changes in visuals to children
    notifyVisuals : 1, // send msgBorderSetVisuals to observers
    spare3 : 14; // unused (reserved)
} BORDER_STYLE, *P_BORDER_STYLE;
```

Sets style.look, style.preview, and style.selected from pArgs as in msgBorderSetStyle.

If style.backgroundlnk is not currently bsInkTransparent, sets style.backgroundlnk from pArgs as in msgBorderSetStyle.

msgBorderGetBorderRect

Passes back the rect on the border.

Takes P_RECT32, returns STATUS.

```
#define msgBorderGetBorderRect
MakeMsg(clsBorder, 3)
```

The first pixel of this rect is on the border. This is the rectangle on which the border edges will be drawn, which is outside the inner margin. pArgs is in device units.
**msgBorderInsetToBorderRect**
Assumes given rect is window bounds, insets to border rect as in `msgBorderGetBorderRect`.
Takes `P_RECT32`, returns `STATUS`.

```c
#define msgBorderInsetToBorderRect MakeMsg(clsBorder, 7)
```

Comments
You can send this message to determine where the border rect would be with the given bounds.

`clsBorder` will self-send this message during `msgWinRepaint` to determine the rect on which the border edges should be drawn.

`pArgs` should be in device units.

**msgBorderGetInnerRect**
Passes back the rect after the inner margin.
Takes `P_RECT32`, returns `STATUS`.

```c
#define msgBorderGetInnerRect MakeMsg(clsBorder, 4)
```

Comments
The first pixel of this rect is inside the shadow, border edges and margin area. This is the outer-most usable area. `pArgs` is in device units. Subclasses should use this message to determine the area available to draw in after `clsBorder` has drawn all the shadows and borders.

**msgBorderInsetToInnerRect**
Assumes given rect is window bounds, insets to inner rect as in `msgBorderGetInnerRect`.
Takes `P_RECT32`, returns `STATUS`.

```c
#define msgBorderInsetToInnerRect MakeMsg(clsBorder, 18)
```

**msgBorderGetMarginRect**
Passes back the rect after the border.
Takes `P_RECT32`, returns `STATUS`.

```c
#define msgBorderGetMarginRect MakeMsg(clsBorder, 31)
```

Comments
The first pixel of this rect is the start of the margin area. `pArgs` is in device units.

**msgBorderInsetToMarginRect**
Assumes given rect is window bounds, insets to margin rect as in `msgBorderGetMarginRect`.
Takes `P_RECT32`, returns `STATUS`.

```c
#define msgBorderInsetToMarginRect MakeMsg(clsBorder, 35)
```

**msgBorderGetOuterSize**
Passes back the sum of the border, margin and shadow sizes for width and height.
Takes `P_SIZE32`, returns `STATUS`.

```c
#define msgBorderGetOuterSize MakeMsg(clsBorder, 5)
```

Comments
Values are in device units. Subclasses can use this message to determine the space needed for the border area. For example, `clsLabel` will use this number to compute its total shrink-wrap size.
**msgBorderGetOuterSizes**
Passes back the breakdown of the outer size requirements.
Takes P_RECT32, returns STATUS.

```c
#define msgBorderGetOuterSizes MakeMsg(clsBorder, 36)
```

**Comments**
OuterSizes are insets from outer edge to inner rect. Note that this is not a true rectangle, each field (x, y, w, h) is a distance from the outer edge. The sum x+w is equivalent to the OuterSize w, the sum y+h is equivalent to the OuterSize h. Values are in device units.

**msgBorderGetOuterOffsets**
Passes back the distance from the outer edge to the border rect in each dimension.
Takes P_RECT32, returns STATUS.

```c
#define msgBorderGetOuterOffsets MakeMsg(clsBorder, 25)
```

**Comments**
OuterOffsets are insets from outer edge to inner rect. Note that this is not a true rectangle, each field (x, y, w, h) is a distance from the outer edge.
Values are in device units.
This message may be subclassed to return the visual outer offsets. For example, clsFrame will return the outer offsets to the frame border window.

**msgBorderXOR**
Sets the raster-op to XOR and paints the background.
Takes U16, returns STATUS.

```c
#define msgBorderXOR MakeMsg(clsBorder, 33)
```

**Comments**
The U16 passed in is used as backgroundInk. Using `pArgs` of bsInkWhite yields a true XOR, bsInkGray66 gives a graying effect.

**msgBorderPaint**
Paints the border, background, shadow, etc. using msgWinBeginPaint.
Takes VOID, returns STATUS.

```c
#define msgBorderPaint MakeMsg(clsBorder, 34)
```

**See Also**
msgBorderXOR

**msgBorderProvideDeltaWin**
Category: ancestor window responsibility Receiver must provide window to be dragged, resized or topped.
Takes P_WIN, returns STATUS.

```c
#define msgBorderProvideDeltaWin MakeMsg(clsBorder, 23)
```

**Comments**
clsBorder will respond by self-sending msgWinSend with the following WIN_SEND parameters:

```c
ws.flags = wsSendDefault;
w.sendSend = SizeOf(WIN_SEND);
ws.msg = msgBorderProvideDeltaWin;
w.data[0] = objNull;
```
*pArgs will be set to ws.data[0].

This message is used by clsBorder if style.getDeltaWin is true to determine which window to drag/resize/top.

See Also

msgWinSend

**msgBorderProvideBackground**

**catagory:** subclass responsibility Receiver must provide rect and ink for drawing background.

Takes P_BORDER_BACKGROUND, returns STATUS.

**Arguments**

typedef struct BORDER_BACKGROUND {
    RECT32 rect; // in/out: background rect to fill
    U16 ink; // in/out: ink to fill with (e.g. bsInkWhite)
    U16 borderInk; // in/out: ink to draw border with (e.g. bsInkBlack)
    U32 spare; // unused (reserved)
} BORDER_BACKGROUND, *P_BORDER_BACKGROUND;

#define msgBorderProvideBackground MakeMsg(clsBorder, 24)

**Comments**

Self-sent during msgWinRepaint if style.preview or style.selected is true. pArgs defaults to current border rect, background and border inks.

A subclass can catch this message and change any of the parameters. For example, clsMenuButton will alter the background rect if the menu button has a top or bottom border on, to back away previewing feedback from the border edge.

**msgBorderPaintForeground**

**catagory:** subclass window responsibility Receiver must paint the foreground, if any.

Takes VOID, returns STATUS.

#define msgBorderPaintForeground MakeMsg(clsBorder, 32)

**Comments**

clsBorder never sends this message. A subclass may send this message to force an ancestor class (e.g. clsLabel) to paint the foreground.

clsBorder responds by doing nothing and returning stsOK.

See Also

msgBorderPaint

**msgBorderFlash**

Flashes self's window by drawing a thick border and erasing it.

Takes VOID, returns STATUS.

#define msgBorderFlash MakeMsg(clsBorder, 40)

**Comments**

clsBorder will flash a border around self's window. This is used by msgBorderTop to hilight a window that is already on top.

See Also

msgBorderTop

**msgBorderTop**

Tops the border window with optional UI feedback and/or bottoming.

Takes U32, returns STATUS.

#define bsTopFlash ((U32)flag0) // msgBorderFlash if already on top
#define bsTopBottom ((U32)flag1) // send to bottom if already on top
#define msgBorderTop MakeMsg(clsBorder, 41)
If self is not already on top of its siblings, clsBorder will bring self to top.

If pArgs has bsTopFlash on and self is already on top, clsBorder will self-send msgBorderFlash to flash a border around self.

If pArgs has bsTopBottom on and self is already on top, clsBorder will re-insert self at the "bottom".

The bottom is defined as the first child of the mainWin of theDesktop. If theDesktop does not exist, or it has no mainWin, self is placed at the bottom of its sibling stack. If self is not a sibling of the mainWin of theDesktop, nothing is done.

msgBorderFlash

Messages from other classes

msgWinSend

Sends a message up a window ancestry chain.

Takes WIN_SEND, returns STATUS.

If pArgs->msg is msgBorderProvideDeltaWin and style.getDeltaWin is true, clsBorder will set pArgs->data[0] to self and return stsOK. This will result in self being the window that is dragged/resized/topped.

msgInputEvent

Notification of an input event.

Takes P_INPUT_EVENT, returns STATUS.

clsBorder will respond to input events that trigger dragging, resizing, or topping.

If pArgs->devCode is msgPenHoldTimeout and style.drag is bsDragHoldDown, or pArgs->devCode is msgPenDown and style.drag is bsDragDown, or pArgs->devCode is msgPenMoveDown and style.drag is bsDragMoveDown and the pen has moved beyond a small threshold since the last msgPenDown, the following is done:

msgGWinAbort(pNull) is self-sent to terminate any gesture in progress.

If style.getDeltaWin is true, msgBorderProvideDeltaWin is self-sent msgWinSend to determine the window to be dragged. Otherwise, is used as the window to be dragged.

msgTrackProvideMetrics is sent to the window to be dragged with _METRICS parameters as follows:

msgNew Defaults is sent to clsTrack to initialize a TRACK _METRICS struct and then:

style.startThickness = tsThicknessDouble;
win = parent of window to be dragged;
client = self;
clientData = window to be dragged;
initRect = bounds of window to be dragged;
constrainRect.size = size of window to be dragged;
keepRect = rect around grabbed point;
tag = tagBorderDrag;

An instance of clsTrack is created and started via msgTrackStart.

If pArgs->devCode is msgPenUp and style.top is bsTopUp and gWin.style.gestureEnable is false, a window to be topped is determined as in the window to be dragged above, and
msgBorderTop(bsTopBottom) is sent to the window to bring it to top (or take it to bottom if already on top).

If pArgs->devCode is one of msgPenInProxUp, msgPenEnterUp, or msgPenMoveUp, and style.resize is not bsResizeNone, and pArgs->xy is in one of the resize handle areas, the following is done:

msgGWinAbort(pNull) is self-sent to terminate any gesture in progress.

A window to be resized is determined as in the window to be dragged above, an instance of clsGrabBox is created with this window as its client. grabBox is sent msgGrabBoxShow(true) to start the resize feedback.

If a drag or resize is done, pArgs->flags will have evBorderTaken turned on to indicate that clsBorder "took" the event.

---

**msgTrackDone**

Sent by a tracker when it's done.

Takes P_TRACK_METRICS, returns STATUS. Category: client notification.

**Comments**

If pArgs->tag is not tagBorderDrag, nothing is done and the message is passed to ancestor.

Otherwise, clsBorder assumes pArgs->clientData is a window to be dragged and sends msgWinDelta to this window to change its origin to one based on pArgs->rect.origin.

If style.top is bsTopDrag, the window to be dragged is also topped (brought to front) by sending it msgBorderTop(0).

---

**msgTimerNotify**

Notifies the client that the timer request has elapsed.

Takes P_TIMER_NOTIFY, returns nothing. Category: advisory message.

**Comments**

If self's lineStyle is bsLineMarquee or bsLineDoubleMarquee, clsBorder will animate the marquee and set the timer again.

---

**msgSelSelect**

Sets self to be the selection.

Takes nothing, returns STATUS.

**Comments**

clsBorder responds by self-sending msgBorderSetSelected(true).

---

**msgSelYield**

The Selection Manager requires the release of the selection.

Takes BOOLEAN, returns STATUS.

**Comments**

clsBorder responds by self-sending msgBorderSetSelected(false).

---

**msgGWinGesture:**

Called to process the gesture.

Takes P_GWIN_GESTURE, returns STATUS.
If `pArgs->msg` is `xgs1Tap` and `style.top` is `bsTopUp`, a window to be topped is determined and topped as in response to the input event `msgPenUp`.

If `pArgs->msg` is `xgsQuestion` and `style.resize` is not `bsResizeNone` and `pArgs->hotPoint` falls over one of the resize handle areas, quick help for the resize handle is shown.

**msgWinRepaint**

Tells a window to repaint itself.

Takes nothing, returns `STATUS`. Category: descendant responsibility.

**clsBorder** responds by painting the background, shadow, resize handles, and border edges.

**msgBorderInsetToBorderRect** will be self-sent with a default of the current window bounds to allow the subclass to alter the rect on which the border will be drawn.

If `style.preview` or `style.selected` are true, `msgBorderProvideBackground` is self-sent with the following `BORDER_BACKGROUND` parameters:

- `rect` = rectangle on which the border will be drawn, in device units;
- `ink` = `backgroundInk` to be used;

The resulting rect and ink are used during painting.

If any of the specified inks have `bsInkExclusive` or-ed in, and the border window does not exclusively paint the pixels in its window, `bsInkTransparent` will be used. The test for a window exclusively painting the pixels in its window is as follows:

```c
define selfStyle to be self's window style flags
define parentStyle to be parent's window style flags
if (selfStyle & wsTransparent) return false;
if (selfStyle & (wsClipSiblings | wsClipChildren)) return true;
if (! (selfStyle & wsParentClip)) return true;
if (parentStyle & wsTransparent) return true;
if (parentStyle & wsClipChildren) return true;
return false;
```

If any of the specified inks are `bsInkTransparent`, nothing will be painted for that feature (e.g. `backgroundInk` of `bsInkTransparent` results in no paint on the background).

**msgScrollWinProvideDelta**

Self-sent when `scrollWin.style.getDelta` is set to true so that descendant or client can normalize the scroll if desired.

Takes `P_SCROLL_WIN_DELTA`, returns `STATUS`. Category: descendant/client responsibility.
clsBorder responds by computing a new origin based on pArgs->action and normalizing to prevent scrolling into part of a row or column.

clsBorder will enumerate the leaf-level children and try to compute the row/column structure from the placement of the children.

Normalization will occur in the direction of the scroll. For example, if the scroll action is moving upward (e.g sbLineUp), normalization will occur at the top of the view.
This file contains the API for clsBusy and theBusyManager.

clsBusy inherits from clsObject.

theBusyManager is typically the only instance of clsBusy. theBusyManager puts up and takes down a visual indication that the system is busy.

Debugging Flags

The clsBusy debugging flag is 'K'. Defined values are:

flag0 (0x0001) general busy on/off/inhibit
flag10 (0x0400) never put up the busy UI

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

ifndef CLSMGR_INCLUDED
#endif

#define msgBusyDisplay MakeMsg(clsBusy, 9)
#define busyOff 0     // turn the busy UI off
#define busyOn 1      // turn the busy UI on
// these can be or-ed into busyOn or busyOff
#define busyNoRefCount flag1 // don't increment/decrement the ref count
#define busyNoDelay flag2  // don't wait for timer to display

You send this message to theBusyManager.

theBusyManager maintains a reference count. Requests of busyOn increment the count, and requests of busyOff decrement the count. theBusyManager will put up the UI when the count goes from 0 to 1, and take the UI down when the count goes from 1 to 0.

If pArgs is busyOn | busyNoRefCount, and the reference count is already 1 or greater (i.e. the busy UI is already being displayed), nothing is done.

If pArgs is busyOn | busyNoDelay, the busy UI will be displayed immediately, skipping the usual delay time.

If pArgs is busyOff | busyNoRefCount, the reference count is set to 0 and the busy UI is taken down.

The busy UI will be displayed (i.e. hot spot at) the last xy specified via msgBusySetXY. If this is (minS32, minS32), the xy specified via msgBusySetDefaultXY will be used.

When the busy UI is taken down, the xy for the next display of the busy UI is set to (minS32, minS32).

See Also msgBusyInhibit
msgBusyInhibit
Inhibits/allows display of the busy UI.
Takes BOOLEAN, returns STATUS.

#define msgBusyInhibit MakeMsg(clsBusy, 10)

You send this message to theBusyManager.

theBusyManager maintains an inhibit reference count. Requests of TRUE increment the count, and
requests of FALSE decrement the count. theBusyManager will take down the UI when the count goes
from 0 to 1, and allow subsequent displays of the busy UI (via msgBusyDisplay(busyOn)) when the
count is zero.

You can use msgBusyInhibit to prevent the busy UI from being displayed, even if requested by other
parts of the system.

msgBusyDisplay

See Also

msgBusySetXY
Specifies the position for the busy UI the next time it is shown.
Takes P_XY32, returns STATUS.

#define msgBusySetXY MakeMsg(clsBusy, 11)

You send this message to theBusyManager. The UI will be centered at pArgs the next time
msgBusyDisplay(busyOn) is sent.
pArgs should be in root window space.
If the busy UI is currently being shown, this message is ignored.

msgBusySetDefaultXY
Specifies the default position for the busy UI the next time it is shown.
Takes P_XY32, returns STATUS.

#define msgBusySetDefaultXY MakeMsg(clsBusy, 12)

The input system sends this message to theBusyManager when an input event has not been processed
within the default time limit. The UI will be centered at pArgs the next time msgBusyDisplay(busyOn)
is sent, if msgBusySetXY has not been used to specify a position.
pArgs should be in root window space.

msgBusyGetSize
Passes back the size of the busy UI.
Takes P_SIZE32, returns STATUS.

#define msgBusyGetSize MakeMsg(clsBusy, 3)

theBusyManager will set *pArgs to the size of the default UI.
This file contains the API definition for clsButton.

clsButton inherits from clsLabel.

Buttons are labels, but with input behavior. Buttons also have a state value: on or off. Buttons notify their client when certain input events occur. clsButton make extensive use of its ancestors display capabilities, particularly clsBorder and clsLabel.

```c
ifndef BUTTON_INCLUDED
#define BUTTON_INCLUDED

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

### Common #defines and typedefs

typedef OBJECT BUTTON;

### Contact Styles

Use one of these values in button's style.contact.

```c
#define bsContactMomentary 0 // push-on, release-off
#define bsContactToggle 1 // push-on, push-off
#define bsContactLockOn 2 // push-on, stays on
// 3 // unused (reserved)
```

### Feedback Styles

Use one of these values in button's style.feedback.

```c
#define bsFeedbackInvert 0 // invert on/off
#define bsFeedbackDecorate 1 // use onDecoration/offDecoration
#define bsFeedbackNone 2 // no feedback
#define bsFeedback3D 3 // 3D shadow effect
#define bsFeedbackBox 4 // boxed outline
// 5 // unused (reserved)
// 6 // unused (reserved)
// 7 // unused (reserved)
```

### pArgs Styles

Use one of these values in button's style.pArgs.

```c
#define bsPargsData 0 // pArgs is data
#define bsPargsValue 1 // pArgs is current value
#define bsPargsUID 2 // pArgs is button's UID
// 3 // unused (reserved)
```
Manager Styles

Use one of these values in button's stylemanager.

```c
#define bsManagerNone 0 // no manager
#define bsManagerParent 1 // parent is the manager
#define bsManagerClient 2 // client is the manager
//
typedef struct BUTTON_STYLE {
  U16 contact : 2, // push behavior
  feedback : 4, // invert, decorate, etc.
  notifyDetail : 1, // notify manager of BeginPreview etc.
  notifyWithMsg : 1, // send specified msg & data
  on : 1, // button state: true == on
  manager : 2, // button manager style
  pArgs : 2, // which pArgs to send with msg
  halfHeight : 1, // half-height borders
  parcel : 2; // unused (reserved)
  U16 onDecoration : 5, // label decoration when on (see label.h)
  offDecoration : 5, // label decoration when off (see label.h)
  spare : 6; // unused (reserved)
} BUTTON_STYLE, *P_BUTTON_STYLE;
```

Default BUTTON_STYLE:

```c
contact = bsContactMomentary
feedback = bsFeedbackInvert
onDecoration = bsDecorationNone
offDecoration = bsDecorationNone
notifyDetail = false
notifyWithMsg = true
pArgs = bsPargsData
on = false
halfHeight = false
```

typedef struct BUTTON_NOTIFY {
  OBJECT button; // uid of sender
  MESSAGE msg; // defined message or some other data
  U32 data; // pArgs for message or some other data
  MESSAGE detail; // msgButtonBeginPreview, etc.
  U32 spare; // unused (reserved)
} BUTTON_NOTIFY, *P_BUTTON_NOTIFY;
```

Messages

msgNew

Creates a button window.

Takes P_BUTTON_NEW, returns STATUS. Category: class message.

Arguments
typedef struct BUTTON_NEW_ONLY {
  BUTTON_STYLE style; // overall style
  MESSAGE msg; // message to send or other data
  U32 data; // pArgs for msg or other data
  U16 onCustomGlyph; // glyph to use for
  // lsDecorationCustomLeft/Right
  U16 offCustomGlyph; // glyph to use for
  // lsDecorationCustomLeft/Right
  U32 spare; // unused (reserved)
} BUTTON_NEW_ONLY, BUTTON_METRICS,
*P_BUTTON_NEW_ONLY, *P_BUTTON_METRICS;
```

#define buttonNewFields
labelNewFields
BUTTON Nhi ONLY 

typedef struct BUTTON Nhi {
    button NhiFields
} BUTTON Nhi, *P BUTTON Nhi;

Comments

The rest of this description uses the following abbreviations:

on = pArgs->button.style.on;
pButton = &pArgs->button.style;
pBorder = &pArgs->border.style;
plabe = &pArgs->label.style;

If pButton->feedback is bsFeedbackInvert, sets
pBorder->preview = on;

If pButton->feedback is bsFeedback3D, sets
pBorder->join = bsJoinSquare;
pBorder->previewAlter = bsAlterNone;
pBorder->edge = bsEdgeTop | bsEdgeLeft;
pBorder->shadowGap = bsGapNone;
pBorder->preview = on;
if (on) {
    pBorder->borderlnk = bslnkBlack;
pBorder->backgroundlnk = bslnkGray66;
    pBorder->shadow = bsShadowThinWhite;
} else {
    pBorder->borderlnk = bslnkWhite;
pBorder->backgroundlnk = bslnkGray33;
    pBorder->shadow = bsShadowThinGray;
}

If pButton->feedback is bsFeedbackDecorate, sets

plabe->decoration = on ?
pArgs->button.style.onDecoration : pArgs->button.style.offDecoration;

msgNewDefaults

Initializes the BUTTON Nhi structure to default values.

Takes P BUTTON Nhi, returns STATUS. Category: class message.

Message Arguments

typedef struct BUTTON Nhi {
    button NhiFields
} BUTTON Nhi, *P BUTTON Nhi;

Comments

Zeroes out pArgs->button and sets:

pArgs->win.flags.input |= inputTip | inputEnter | inputExit;
pArgs->win.flags.style |= wsFileInline;
pArgs->border.style.edge = bsEdgeAll;
pArgs->border.style.join = bsJoinSquare;
pArgs->border.style.shadow = bsShadowThinBlack;
pArgs->border.style.borderlnk = bslnkGray66;
pArgs->control.style.previewEnable = true;
pArgs->label.style.xAlignment = lsAlignCenter;
pArgs->label.style.yAlignment = lsAlignCenter;
pArgs->button.style.notifyWithMsg = true;
**msgButtonGetMetrics**

Passes back the current metrics.

Takes PFLICT_BUTTON, returns STATUS.

```c
#define msgButtonGetMetrics MakeMsg(clsButton, 1)
```

**msgButtonSetMetrics**

Sets the metrics.

Takes PFLICT_BUTTON, returns STATUS.

```c
#define msgButtonSetMetrics MakeMsg(clsButton, 2)
```

**Comments**

If style.on changes, the button does the following:

- If `style.contact != bsContactMomentary`, self-sends `msgControlSetDirty`, true.
- Self-sends `msgButtonNotifyManager` with `msg = msgButtonDone`.
- Self-sends `msgButtonNotify` with detail of `msgButtonAcceptPreview`. This results in either `msgButtonNotify` or a client-specified message to the client. Alters border and label styles to reflect the new "on" value (see `msgNew` description).

Changes to style.feedback and style.on/offDecoration result in appropriate changes to the Border and Label styles.

**msgButtonGetStyle**

Passes back the current style values.

Takes PFLICT_BUTTON, returns STATUS.

```c
#define msgButtonGetStyle MakeMsg(clsButton, 3)
```

**Message Arguments**

```c
typedef struct BUTTON_STYLE {
  U16 contact : 2,  // push behavior
  feedback : 4,  // invert, decorate, etc.
  notifyDetail : 1,  // notify manager of BeginPreview etc.
  notifyWithMsg : 1,  // send specified msg & data
  on : 1,  // button state: true == on
  manager : 2,  // button manager style
  pArgs : 2,  // which pArgs to send with msg
  halfHeight : 1,  // half-height borders
  spare1 : 2;  // unused (reserved)
  U16 onDecoration : 5,  // label decoration when on (see label.h)
  offDecoration : 5,  // label decoration when off (see label.h)
  spare : 6;  // unused (reserved)
} BUTTON_STYLE, *P_BUTTON_STYLE;
```

**msgButtonSetStyle**

Sets the style values.

Takes PFLICT_BUTTON, returns STATUS.

```c
#define msgButtonSetStyle MakeMsg(clsButton, 4)
```

**Message Arguments**

```c
typedef struct BUTTON_STYLE {
  U16 contact : 2,  // push behavior
  feedback : 4,  // invert, decorate, etc.
  notifyDetail : 1,  // notify manager of BeginPreview etc.
  notifyWithMsg : 1,  // send specified msg & data
  on : 1,  // button state: true == on
  manager : 2,  // button manager style
  pArgs : 2,  // which pArgs to send with msg
  halfHeight : 1,  // half-height borders
  spare1 : 2;  // unused (reserved)
  U16 onDecoration : 5,  // label decoration when on (see label.h)
  offDecoration : 5,  // label decoration when off (see label.h)
  spare : 6;  // unused (reserved)
} BUTTON_STYLE, *P_BUTTON_STYLE;
```
on          : 1,  // button state: true == on
manager     : 2,  // button manager style
pArgs       : 2,  // which pArgs to send with msg
halfHeight  : 1,  // half-height borders
spare       : 2;  // unused (reserved)
U16 onDecoration : 5,  // label decoration when on (see label.h)
offDecoration: 5,  // label decoration when off (see label.h)
spare       : 6;  // unused (reserved)
} BUTTON_STYLE, *P_BUTTON_STYLE;

Comments
Reacts to changes in style.on and other style values as in msgButtonSetMetrics.

msgButtonGetMsg
Passes back metrics.msg.
Takes P_MESSAGE, returns STATUS.

#define msgButtonGetMsg   MakeMsg(clsButton, 5)

msgButtonSetMsg
Sets metrics.msg.
Takes MESSAGE, returns STATUS.

#define msgButtonSetMsg   MakeMsg(clsButton, 6)

msgButtonGetData
Passes back metrics.data.
Takes P_U32, returns STATUS.

#define msgButtonGetData   MakeMsg(clsButton, 7)

msgButtonSetData
Sets metrics.data.
Takes U32, returns STATUS.

#define msgButtonSetData   MakeMsg(clsButton, 8)

msgButtonSetNoNotify
Sets the value of the button (i.e. style.on) without notifying.
Takes BOOLEAN, returns STATUS.

#define msgButtonSetNoNotify   MakeMsg(clsButton, 17)

Comments
pArgs must be true or false. The button's manager and client are not notified. Alters border and label styles to reflect new on value (see msgNew description).

msgButtonShowFeedback
Shows the feedback for an on/off button if pArgs is true/false.
Takes BOOLEAN, returns STATUS. Category: self-sent.

#define msgButtonShowFeedback   MakeMsg(clsButton, 19)

Comments
This message is self-sent by clsButton to change the on/off feedback shown to the user. For example, when a button with a contact style of bsContactToggle is pressed and msgControlBeginPreview is
received, clsButton self-sends msgButtonShowFeedback(!style.on) to show the user what will happen when the pen is lifted.

Subclasses can handle the message and show the appropriate feedback for the new state.

### Messages Sent to Button’s Manager

#### msgButtonDone

Sent via msgWinSend to the manager when button receives msgControlAcceptPreview. Takes UID, returns STATUS. Category: manager notification.

```c
#define msgButtonDone MakeMsg(clsButton, 16)
```

#### msgButtonBeginPreview

Sent via msgWinSend to the manager when button receives msgControlBeginPreview. Takes UID, returns STATUS. Category: manager notification.

```c
#define msgButtonBeginPreview MakeMsg(clsButton, 9)
```

Comments

Only sent if style.notifyDetail is true.

#### msgButtonUpdatePreview

Sent via msgWinSend to the manager when button receives msgControlUpdatePreview. Takes UID, returns STATUS. Category: manager notification.

```c
#define msgButtonUpdatePreview MakeMsg(clsButton, 10)
```

Comments

Only sent if style.notifyDetail is true.

#### msgButtonRepeatPreview

Sent via msgWinSend to the manager when button receives msgControlRepeatPreview. Takes UID, returns STATUS. Category: manager notification.

```c
#define msgButtonRepeatPreview MakeMsg(clsButton, 11)
```

Comments

Only sent if style.notifyDetail is true.

#### msgButtonCancelPreview

Sent via msgWinSend to the manager when button receives msgControlCancelPreview. Takes UID, returns STATUS. Category: manager notification.

```c
#define msgButtonCancelPreview MakeMsg(clsButton, 12)
```

Comments

Only sent if style.notifyDetail is true.

#### msgButtonAcceptPreview

Sent via msgWinSend to the manager when button receives msgControlAcceptPreview. Takes UID, returns STATUS. Category: manager notification.

```c
#define msgButtonAcceptPreview MakeMsg(clsButton, 13)
```

Comments

Only sent if style.notifyDetail is true.
**msgButtonNotifyManager**

Sent to self when button wants to notify its manager.

Takes P_BUTTON_NOTIFY, returns STATUS. Category: self-sent.

```c
#define msgButtonNotifyManager MakeMsg(clsButton, 18)
```

```c
typedef struct BUTTON_NOTIFY {
    OBJECT button;     // uid of sender
    MESSAGE msg;       // defined message or some other data
    U32 data;         // pArgs for message or some other data
    MESSAGE detail;   // msgButtonBeginPreview, etc.
    U32 spare;        // unused (reserved)
} BUTTON_NOTIFY, *P_BUTTON_NOTIFY;
```

Comments

A button responds to this by sending `msgWinSend` with the following WIN_SEND parameters to its manager:

```c
deflags = wsSendDefault;
lenSend = SizeOf(WIN_SEND);
msg = pArgs->msg;
data[0] = pArgs->data;
```

---

**msgButtonNotify**

Sent to self when button wants to notify its client.

Takes P_BUTTON_NOTIFY, returns STATUS. Category: client notification.

```c
#define msgButtonNotify MakeMsg(clsButton, 14)
```

```c
typedef struct BUTTON_NOTIFY {
    OBJECT button;     // uid of sender
    MESSAGE msg;       // defined message or some other data
    U32 data;         // pArgs for message or some other data
    MESSAGE detail;   // msgButtonBeginPreview, etc.
    U32 spare;        // unused (reserved)
} BUTTON_NOTIFY, *P_BUTTON_NOTIFY;
```

Comments

If `style.notifyWithMessage` is true, `pArgs->msg` is sent to the button’s client with the `pArgs` of `pArgs->data` or as specified by `style.pArgs`.

Otherwise, `msgButtonNotify` is sent to the button’s client with the following `BUTTON_NOTIFY` parameters:

```c
  button = self;
  msg = pArgs->msg;
  data = pArgs->data;
  detail = pArgs->detail;
```

---

**Messages Defined by Other Classes**

**msgBorderGetForegroundRGB**

Passes back foreground RGB to use given current visuals.

Takes P_SYSDC_RGB, returns STATUS.

Comments

If `style.feedback` is `bsFeedback3D` and `border.style.look` is `bsLookInactive`, passes back `sysDcRGBGray66`. Otherwise, calls ancestor.
**msgControlBeginPreview**
Self-sent when `msgPenDown` is received.

**Comments**
Button computes new on value according to `style.feedback` (e.g. `bsContactToggle` results in `on = !style.on`).
Alters border and label styles to reflect new on value and self-sends `msgWinUpdate` to repaint immediately. `style.on` is not changed.
If `style.contact` is not `bsContactMomentary`, self sends `msgControlSetDirty`, true.
If `style.notifyDetail` is true, self-sends `msgButtonNotifyManager`, which results in `msgWinSend` to the manager. Also, if `control.style.previewRepeat` is true, self-sends `msgButtonNotify` which results in client notification.

**msgControlUpdatePreview**
Self-sent when `msgPenMoveDown` is received.

**Comments**
If `style.notifyDetail` is true, notifies manager and client as in `msgControlBeginPreview`.

**msgControlRepeatPreview**
Self-sent if `style.repeatPreview` is true. Initial delay is 600ms, then immediate repeat until `msgPenUp`.

**Comments**
If `style.notifyDetail` is true, notifies manager and client as in `msgControlBeginPreview`.

**msgControlCancelPreview**
Self-sent when `control.style.previewGrab` is false and `msgPenExitDown` is received.

**Comments**
Clients or subclasses can send this to a control to cancel existing preview.
Alters border and label styles to reflect current `style.on` value and self-sends `msgWinUpdate` to repaint immediately. This undoes the visual effects of `msgControlBeginPreview`.
If `style.notifyDetail` is true, notifies manager and client as in `msgControlBeginPreview`.

**msgControlAcceptPreview**
Self-sent when `msgPenUp` is received.

**Comments**
If gestures are enabled this message is not sent until `msgGWinGesture` is received with xgs!Tap.
Self-sends `msgControlSetValue` with on value computed as in `msgControlBeginPreview`.

**msgControlSetValue**
Sets `style.on`.
Takes `S32`, returns `STATUS`. 
Comments

Updates visuals to reflect new on value as in msgButtonSetMetrics.

Self-sends msgButtonNotifyManager with the following BUTTON_NOTIFY parameters (this results in msgWinSend to the manager):

```c
button = self;
msg   = msgButtonDone;
data  = self;
```

Self-sends msgButtonNotify with the following BUTTON_NOTIFY parameters (this results in client notification):

```c
button = self;
msg    = metrics.msg;
data    = metrics.data;
detail = msgButtonAcceptPreview;
```

msgControlGetValue

Passes back the style.on.

Takes P_S32, returns STATUS.
This file contains the API for clsChoiceMgr. clsChoiceMgr inherits from clsManager. Choice managers serve as tkTable managers in tables of buttons. A choice manager, when plugged in as the manager of a tkTable of buttons, responds to the msgWinSend's generated by the buttons and makes the entire group perform as a choice.

**Debugging Flags**

The clsChoiceMgr debugging flag is 'K'. Defined values are:

```c
#define CHMGR_INCLUDED
#define CHMGR_INCLUDED
#include <manager.h>
#undef MANAGER_INCLUDED
```

**Common #defines and typedefs**

```c
typedef OBJECT CHOICE_MGR;
```

**msgNew**

Creates a choice manager.

Takes P_CHOICE_MGR_NEW, returns STATUS. Category: class message.

```c
typedef struct CHOICE_MGR_NEW_ONLY {
  U32 spare; // unused (reserved)
} CHOICE_MGR_NEW_ONLY, *P_CHOICE_MGR_NEW_ONLY;
#define choiceMgrNewFields \
  managerNewFields \
  CHOICE_MGR_NEW_ONLY choiceMgr;
```

```c
typedef struct CHOICE_MGR_NEW {
  choiceMgrNewFields
} CHOICE_MGR_NEW, *P_CHOICE_MGR_NEW;
```

**msgNewDefaults**

Initializes the CHOICE_MGR_NEW structure to default values.

Takes P_CHOICE_MGR_NEW, returns STATUS. Category: class message.

```c
typedef struct CHOICE_MGR_NEW {
  choiceMgrNewFields
} CHOICE_MGR_NEW, *P_CHOICE_MGR_NEW;
```

clsChoiceManager has no instance data of its own.
msgChoiceMgrGetOnButton

Gets the current on button. Passes back objNull if no button is on.
Takes P_UID, returns STATUS.

#define msgChoiceMgrGetOnButton MakeMsg(clsChoiceMgr, 1)

msgChoiceMgrSetOnButton

Sets the current on button.
Takes UID, returns STATUS.

#define msgChoiceMgrSetOnButton MakeMsg(clsChoiceMgr, 2)

Comments

Since the choiceMgr will use msgControlSetValue to turn the button on, that button's normal notification protocol will be invoked.
All buttons are turned off if message argument is objNull.

msgChoiceMgrSetNoNotify

Like msgChoiceMgrSetOnButton, but no notifications are generated.
Takes UID, returns STATUS.

#define msgChoiceMgrSetNoNotify MakeMsg(clsChoiceMgr, 3)

Messages from Other Classes

msgWinSend

Sends a message up a window ancestry chain.
Takes P_WIN_SEND, returns STATUS.

Comments

clsChoiceMgr responds when pArgs->msg is msgButtonBeginPreview, msgButtonCancelPreview, or msgButtonDone. If pArgs->msg is anything else, clsChoiceMgr just returns stsManagerContinue.
For these three messages, clsChoiceMgr will make the set of entry windows act as a group.

Return Value

stsManagerContinue clsChoiceMgr always returns this so that the caller will continue to propagate the msgWinSend.
This file contains the API for clsChoice.

clsChoice inherits from clsTkTable.

Choices are tkTables of buttons that act as exclusive choices.

Note that msgNewDefaults to clsChoice results in a prototypical new struct whose values describe a button of contact style bsContactLockOn. This is correct for choices that always have one button on, but this won’t work if you want a choice that can have 0 or 1 buttons on. In this case, making each button child have a contact style of bsContactToggle will achieve the desired effect. Here is the appropriate code.

```
ObjCallWarn(MsgNewDefaults, clsChoice, &choiceNew);
choiceNew.tkTable.pButtonNew->button.style.contact = bsContactToggle;
ObjCallRet(msgNew, clsChoice, &choiceNew, s);
```

See the documentation for msgTkTableChildDefaults below.

```
#ifndef CHOICE_INCLUDED
#define CHOICE_INCLUDED

#include <tktable.h>

Common #defines and typedefs

typedef OBJECT CHOICE;
typedef struct CHOICE_STYLE {
    U16 spare;  // unused (reserved)
} CHOICE_STYLE, *P_CHOICE_STYLE;

Informational return status returned by msgControlGetValue if choice has no value
#define stsChoiceNoValue  MakeWarning(clsChoice, 1)
```

**msgNew**

Creates a choice (and its nested button windows).

Takes P_CHOICE_NEW, returns STATUS. Category: class message.

```
typedef struct CHOICE_NEW_ONLY {
    CHOICE_STYLE style;  // overall style
    U32 value;  // tag of on button
    U32 spare;  // unused (reserved)
} CHOICE_NEW_ONLY, *P_CHOICE_NEW_ONLY;
#define choiceNewFields
    tkTableNewFields
    CHOICE_NEWONLY choice;

typedef struct CHOICE_NEW {
    choiceNewFields
} CHOICE_NEW, *P_CHOICE_NEW;
```
Will create a default instance of clsChoiceMgr if the incoming pArgs->tkTable.manager is null. The uid of the created manager will be an out parameter.

After the manager has been set up, clsChoice will use msgControlGetValue to find the button that is 'on', and then send msgChoiceMgrSetNoNotify to the manager to tell the manager which button is 'on'.

**msgNewDefaults**

Initializes the CHOICE_NEW structure to default values.

Takes P_CHOICE_NEW, returns STATUS. Category: class message.

```c
typedef struct CHOICE_NEW {
    choiceNewFields
} CHOICE_NEW, *P_CHOICE_NEW;
```

Sets up tkTable.pButtonNew to create buttons by default. Zeroes out pNew.choice and sets:

- pArgs->gWin.style.gestureEnable = false;
- pArgs->tableLayout.style.growChildHeight = false;
- pArgs->tableLayout.style.growChildWidth = true;
- pArgs->tableLayout.numCols.constraint = tlAbsolute;
- pArgs->tableLayout.numCols.value = 1;
- pArgs->tableLayout.numRows.constraint = tlAbsolute;
- pArgs->tableLayout.colWidth.constraint = tlChildrenMax;
- pArgs->tableLayout.colWidth.gap = 0;
- pArgs->tableLayout.rowHeight.constraint = tlGroupMax;
- pArgs->tableLayout.rowHeight.gap = 0;
- pArgs->tkTable.manager = objNull;

### Instance Messages

**msgChoiceGetStyle**

Gets the style of the receiver.

Takes P_CHOICE_STYLE, returns STATUS.

```c
#define msgChoiceGetStyle MakeMsg(clsChoice, 1)
```

```c
typedef struct CHOICE_STYLE {
    U16 spare; // unused (reserved)
} CHOICE_STYLE, *P_CHOICE_STYLE;
```

**msgChoiceSetStyle**

Sets the style of the receiver.

Takes P_CHOICE_STYLE, returns STATUS.

```c
#define msgChoiceSetStyle MakeMsg(clsChoice, 2)
```

```c
typedef struct CHOICE_STYLE {
    U16 spare; // unused (reserved)
} CHOICE_STYLE, *P_CHOICE_STYLE;
```
**msgChoiceSetNoNotify**

Like *msgControlSetValue* (see below), but without button notifications.

Takes TAG, returns STATUS.

```c
#define msgChoiceSetNoNotify MakeMsg(clsChoice, 3)
```

Comments

Using this message avoids button notifications being sent out to their clients.

---

**Messages from Other Classes**

**msgFree**

Sent as the last of three msgs to destroy an object.

Takes OBJ_KEY, returns STATUS.

Comments

If the choice had created its own TK_TABLE_NEWONLY.manager at *msgNew* time, the manager will be sent *msgDestroy*.

**msgRestore**

Creates and restores an object from an object file.

Takes P_OBJ_RESTORE, returns STATUS.

Comments

*clsChoice* responds by restoring its instance data. If the choice had created its own TK_TABLE_NEWONLY.manager at *msgNew* time, a new one is created from *clsChoiceMgr*.

**msgSave**

Causes an object to file itself in an object file.

Takes P_OBJ_SAVE, returns STATUS.

Comments

*clsChoice* responds by filing away its instance data. It will remember whether *clsChoice* created its own TK_TABLE_NEWONLY.manager at *msgNew* time.

**msgWinSend**

Sends a message up a window ancestry chain.

Takes P_WIN_SEND, returns STATUS.

Comments

*clsChoice* responds when *pArgs-*msg is *msgButtonBeginPreview* or *msgButtonDone* by using *msgControlSetDirty(true)* to mark its children as dirty. This is done as follows:

*clsChoice* calls its ancestor and remembers the returned status. It then tests whether *pArgs-*msg is *msgButtonDone*. If so, then if one of the child buttons is currently previewing, *clsChoice* just returns the saved status (because it was when the previewing started that the choice marked its children as dirty). If, however, the msg is *msgButtonDone* and no button is previewing, the choice will go ahead and mark its children dirty (this case can happen if a child button is changing value programmatically and so isn't previewing), then return *stsManagerContinue*.

If the *pArgs-*msg is *msgButtonBeginPreview*, the choice will mark its children dirty and then return *stsManagerContinue*.

If the *pArgs-*msg is anything else, *clsChoice* will return the status saved from the call to its ancestor.

Return Value

*stsManagerContinue* tell the caller to continue to propagate the *msgWinSend*
msgControlGetDirty
Sets *pArgs true if any child control is dirty, false otherwise.
Takes P_BOOLEAN, returns STATUS.

msgControlGetEnable
Sets *pArgs true if any child control is enabled, false otherwise.
Takes P_BOOLEAN, returns STATUS.

msgControlGetValue
Gets the tag of the child button that is currently on.
Takes P_TAG, returns STATUS.
Comments
Returns stsChoiceNoValue if no child button is on.

msgControlSetDirty
Forwards this message and pArgs on to each child control in the choice.
Takes BOOLEAN, returns STATUS.

msgControlSetEnable
Forwards this message and pArgs on to each child control in the choice.
Takes BOOLEAN, returns STATUS.

msgControlSetValue
Turns on the child button having the passed tag.
Takes TAG, returns STATUS.
Comments
If another child button was on, it is turned off.

msgTkTableAddAsFirst
Adds specified window as the first child in the table.
Takes WIN, returns STATUS.
Comments
c1sChoice first calls its ancestor, then gets its manager via msgTkTableGetManager. If it has no manager, clsChoice returns stsOK. Otherwise, clsChoice gets the BUTTON_STYLE.on value of the new button and, if that is true, uses msgChoiceMgrSetOnButton to change the choice's 'on' button to the one just added.

msgTkTableAddAsLast
Adds specified window as the last child in the table.
Takes WIN, returns STATUS.
Comments
c1sChoice first calls its ancestor, then gets its manager via msgTkTableGetManager. If it has no manager, clsChoice returns stsOK. Otherwise, clsChoice gets the BUTTON_STYLE.on value of the new button and, if that is true, uses msgChoiceMgrSetOnButton to change the choice's 'on' button to the one just added.
**msgTkTableAddAsSibling**
Inserts specified window in front of or behind an existing child.
Takes P_TK_TABLE_ADD_SIBLING, returns STATUS.

Comments

clsChoice first calls its ancestor, then gets its manager via msgTkTableGetManager. If it has no manager, clsChoice returns stsOK. Otherwise, clsChoice gets the BUTTON_STYLE.on value of the new button and, if that is true, uses msgChoiceMgrSetOnButton to change the choice's 'on' button to the one just added.

**msgTkTableAddAt**
Inserts specified window table at specified index.
Takes P_TK_TABLE_ADD_AT, returns STATUS.

Comments

clsChoice first calls its ancestor, then gets its manager via msgTkTableGetManager. If it has no manager, clsChoice returns stsOK. Otherwise, clsChoice gets the BUTTON_STYLE.on value of the new button and, if that is true, uses msgChoiceMgrSetOnButton to change the choice's 'on' button to the one just added.

**msgTkTableRemove**
Extracts pArgs from the table.
Takes WIN, returns STATUS.

Comments

clsChoice first calls its ancestor, then gets its manager via msgTkTableGetManager. If it has no manager, clsChoice returns stsOK. Otherwise, clsChoice checks to see if the button being removed is the one that is currently 'on' (by sending msgChoiceMgrGetOnButton to its manager). If so, the choice will either set the manager's 'on' button to the first remaining child (if the button's BUTTON_STYLE.contact is bsContactLockOn), or to null (if no children remain or the button's BUTTON_STYLE.contact is anything else). Put simply, the choice repairs its state according to whether the choice is always exactly one value, or can have no value.

**msgTkTableChildDefaults**
Sets the defaults in P_ARGS for a common child.
Takes P_UNKNOWN, returns STATUS.

Comments

This can be sent to either an instance of clsChoice or to clsChoice itself. Here is the response for either case:

```c
if (<pArgs->object.class inherits from clsGWin>
    pArgs->gWin.style.gestureEnable = false;
else

if (<pArgs->object.class inherits from clsBorder> {
    pArgs->border.style.edge = bsEdgeNone;
    pArgs->border.style.topMargin = 1;
    pArgs->border.style.bottomMargin = 1;
}

if (<pArgs->object.class inherits from clsLabel>
    pArgs->label.style.xAlignment = lsAlignLeft;
```
if (<pArgs->object.class inherits from clsButton> { 
  pArgs->button.style.notifyDetail = true;
  pArgs->button.style.contact = bsContactLockOn;
  pArgs->button.style.feedback = bsFeedbackDecorate;
  pArgs->button.style.offDecoration =
    lsDecorationExclusiveOff;
  pArgs->button.style.onDecoration =
    lsDecorationExclusiveOn;
})
This file contains the API definition for clsCustomLayout. clsCustomLayout inherits from clsBorder.
Provides container windows which position and size their child windows according to complex constraints you specify for each child.

See Also

clsTableLayout

Debugging Flags

The clsCustomLayout debugging flag is 'W'. Defined values are:
flag1 (0x0002) msgWinLayoutSelf info
You can also set the '0/0' flag to:
flag8 (0x0100) layout timing

 ifndef CLAYOUT_INCLUDED
 define CLAYOUT_INCLUDED
 ifndef BORDER_INCLUDED
 define BORDER_INCLUDED

 ifndef STRING_H_INCLUDED
 define STRING_H_INCLUDED
 endif

 ifndef _STRING_H_INCLUDED
 define _STRING_H_INCLUDED
 endif

Common #defines and typedefs

typedef OBJECT CSTM_LAYOUT;
typedef struct CSTM_LAYOUT_STYLE {
 U16 limitToRootWin : 1; // limit bounds to stay within theRootWindow
 U16 spare : 15; // unused (reserved)
} CSTM_LAYOUT_STYLE, *P_CSTM_LAYOUT_STYLE;

Default CSTM_LAYOUT_STYLE:

 limitToRootWin = false
typedef struct CSTM_LAYOUT_METRICS {
 CSTM_LAYOUT_STYLE style; // overall style
 U32 spare1; // unused (reserved)
 U32 spare2; // unused (reserved)
} CSTM_LAYOUT_METRICS, *P_CSTM_LAYOUT_METRICS;

Constraints for Custom layout. For each of these, relWin of pNull and relWinTag of zero maps to parent.

Enum16(CSTM_LAYOUT_CONSTRAINT) {
 // for x, y, width, height
 cliAsIs = 0, // x, y: leave unchanged; w, h: use desired size
 cliAbsolute = 1, // use absolute value specified in spec
 // for x, y, width, height: all relative to relWin
 cliBefore = 2, // cliBefore clMinEdge is one pixel less than
 // the border rect; cliBefore clMaxEdge is
 // on the border inner rect
 cliSameAs = 3, // same as relWin
 cliAfter = 4, // cliAfter clMaxEdge is one pixel after max edge
 // cliAfter clMinEdge is on the border inner rect
 cliPctOf = 5 // value * relWin / 100
};
possible edge specifications

#define clMinEdge 0 // x: left edge, y: bottom edge
#define clCenterEdge 1 // x, y: mid point
#define clMaxEdge 2 // x: right edge, y: top edge
#define clBaselineEdge 3 // x: horiz. baseline, y: vertical baseline

macro for defining an x or y constraint to align two edges

define ClAlign(childEdge, constraint, relWinEdge) \
    ( ((childEdge) « 6) | ((relWinEdge) « 4) | (constraint) )

macro for defining a w or h constraint to extend to an edge

define ClExtend(constraint, relWinEdge) \
    ClAlign(clMaxEdge, constraint, relWinEdge)

can be or'ed into any constraint (except clAsIs or clAbsolute) to refer to opposite dimension.

define clOpposite flag 8

can be or'ed into any constraint to compute new value as Max(as-is value, constraint-computed value) useful for children which need to be at least their desired size, but can be bigger (e.g. extend to parent's edge)

define clAtLeastAsIs flag 9

can be or'ed into any constraint to compute new value as specified constraint or clAsIs if the custom layout window has wsShrinkWrapWidth/Height on. This allows a child to be shrink wrapped around if the custom layout window is computing its own size; or, for example, have the child's width extend to the edge of the parent if the parent is forced to a bigger size.

define clAsIsIfShrinkWrap flag10

can be or'ed into width or height constraint to exclude a child's width or height from the shrink-wrap computation. This is useful for children which align to parent's max edge and overlap other children.

define clShrinkWrapExclude flag11

macros to extract the parts of a constraint

define ClChildEdge(c) (((c) >> 6) & 0x3)
define ClRelWinEdge(c) (((c) >> 4) & 0x3)
define ClConstraint(c) ((c) & 0x7)

typedef struct CSTM_LAYOUT_DIMENSION {
    CSTM_LAYOUT_CONSTRAINT constraint;
    S32 value; // offset or absolute value
    WIN relWin; // relative window
    U32 relWinTag; // tag of relative window
    U16 valueUnits 6, // units for value (e.g. bsUnitsLayout)
        spare 10; // unused (reserved)
    U32 spare; // unused (reserved)
} CSTM_LAYOUT_DIMENSION, *P_CSTM_LAYOUT_DIMENSION;

typedef struct CSTM_LAYOUT_SPEC {
    CSTM_LAYOUT_DIMENSION x, y, w, h;
} CSTM_LAYOUT_SPEC, *P_CSTM_LAYOUT_SPEC;

typedef struct CSTM_LAYOUT_CHILD_SPEC {
    WIN child;
    CSTM_LAYOUT_SPEC metrics;
    BOOLEAN parentShrinkWrapWidth;
    BOOLEAN parentShrinkWrapHeight;
    U32 spare; // unused (reserved)
} CSTM_LAYOUT_CHILD_SPEC, *P_CSTM_LAYOUT_CHILD_SPEC;
msgNew
Creates a custom layout window.
Takes P_CSTM_LAYOUT_NEW, returns STATUS. Category: class message.

typedef CSTM_LAYOUT_METRICS CSTM_LAYOUT_NEW_ONLY, *P_CSTM_LAYOUT_NEW_ONLY;
#define customLayoutNewFields
   borderNewFields
CSTM_LAYOUT_NEW_ONLY customLayout;

Arguments
typedef struct CSTM_LAYOUT_NEW {
   customLayoutNewFields
} CSTM_LAYOUT_NEW, *P_CSTM_LAYOUT_NEW;

msgNewDefaults
Initializes the CSTM_LAYOUT_NEW structure to default values.
Takes P_CSTM_LAYOUT_NEW, returns STATUS. Category: class message.

typedef struct CSTM_LAYOUT_NEW {
   customLayoutNewFields
} CSTM_LAYOUT_NEW, *P_CSTM_LAYOUT_NEW;

Zeroes out pNew->customLayout.

msgCstmLayoutGetMetrics
Passes back the current metrics.
Takes P_CSTM_LAYOUT_METRICS, returns STATUS.

#define msgCstmLayoutGetMetrics MakeMsg(clsCustomLayout, 1)

typedef struct CSTM_LAYOUT_METRICS {
   CSTM_LAYOUT_STYLE style; // overall style
   U32 spare1; // unused (reserved)
   U32 spare2; // unused (reserved)
} CSTM_LAYOUT_METRICS, *P_CSTM_LAYOUT_METRICS;

msgCstmLayoutSetMetrics
Sets the current metrics.
Takes P_CSTM_LAYOUT_METRICS, returns STATUS.

#define msgCstmLayoutSetMetrics MakeMsg(clsCustomLayout, 2)

typedef struct CSTM_LAYOUT_METRICS {
   CSTM_LAYOUT_STYLE style; // overall style
   U32 spare1; // unused (reserved)
   U32 spare2; // unused (reserved)
} CSTM_LAYOUT_METRICS, *P_CSTM_LAYOUT_METRICS;

If style.limitToRootWin is changed, msgWinSetLayoutDirty(true) will be self-sent.

msgCstmLayoutGetStyle
Passes back current style values.
Takes P_CSTM_LAYOUT_STYLE, returns STATUS.

#define msgCstmLayoutGetStyle MakeMsg(clsCustomLayout, 5)
typedef struct CSTM_LAYOUT_STYLE {
    U16 limitToRootWin : 1; // limit bounds to stay within theRootWindow
    U16 spare : 15; // unused (reserved)
} CSTM_LAYOUT_STYLE, *P_CSTM_LAYOUT_STYLE;

msgCstmLayoutSetStyle
Sets style values.
Takes P_CSTM_LAYOUT_STYLE, returns STATUS.
#define msgCstmLayoutSetStyle MakeMsg(clsCustomLayout, 6)
Comments
If style.limitToRootWin is changed, msgWinSetLayoutDirty(true) will be self-sent.

CstmLayoutSpecInit
Zeros the P_CSTM_LAYOUT_SPEC.
Returns VOID.
#define CstmLayoutSpecInit(lm) memset((lm), 0, sizeof(CSTM_LAYOUT_SPEC))
Comments
This is equivalent to the following:
x, y, w, h constraint = clAsIs
You should use CstmLayoutSpecInit to initialize the layout spec that you pass in to msgCstmLayoutSetChildSpec. For example:

CSTM_LAYOUT_CHILD_SPEC cs;
CstmLayoutSpecInit(&cs.metrics);
cs.child = child;
cs.metrics.x.constraint = CAlign(clMinEdge, clSameAs, clMinEdge);
cs.metrics.y.constraint = CAlign(clMinEdge, clSameAs, clMinEdge);
ObjCallRet(msgCstmLayoutSetChildSpec, clayout, &cs, s);

msgCstmLayoutSetChildSpec
Sets layout spec for given child.
Takes P_CSTM_LAYOUT_CHILD_SPEC, returns STATUS.
#define msgCstmLayoutSetChildSpec MakeMsg(clsCustomLayout, 3)
Comments
Storage will be allocated for the spec. The child specification will be used in response to msgCstmLayoutGetChildSpec, which is self-sent during msgWinLayoutSelf. clsCustomLayout will self-send msgWinSetLayoutDirty(true).
See Also
CstmLayoutSpecInit
msgCstmLayoutRemoveChildSpec

Removes the spec for the specified child (pArgs).
Takes WIN, returns STATUS.

#define msgCstmLayoutRemoveChildSpec MakeMsg(clsCustomLayout, 7)

Comments
If a child is extracted or destroyed, and msgCstmLayoutSetChildSpec was used to set the child layout constraints, you must use this message to remove the child layout constraints.

See Also
msgCstmLayoutSetChildSpec

msgCstmLayoutGetChildSpec

Passes back the current spec for the specified child.
Takes P_CSTM_LAYOUT_CHILD_SPEC, returns STATUS. Category: self-sent, subclass responsibility.

#define msgCstmLayoutGetChildSpec MakeMsg(clsCustomLayout, 4)
#define stsCstmLayoutBadRelWin MakeStatus(clsCustomLayout, 1)
#define stsCstmLayoutBadRelWinTag MakeStatus(clsCustomLayout, 2)
#define stsCstmLayoutLoop MakeStatus(clsCustomLayout, 3)
#define stsCstmLayoutBadConstraint MakeStatus(clsCustomLayout, 3)

typedef struct CSTM_LAYOUT_CHILD_SPEC {
    WIN child;
    CSTM_LAYOUT_SPEC metrics;
    BOOLEAN parentShrinkWrapWidth;
    BOOLEAN parentShrinkWrapHeight;
    U32 spare;     // unused (reserved)
} CSTM_LAYOUT_CHILD_SPEC, *P_CSTM_LAYOUT_CHILD_SPEC;

Comments
Self-sent during msgWinLayout to retrieve the current spec from subclasses. clsCustomLayout responds by returning the stored spec, or an initialized spec (CstmLayoutSpecInit()) if none is found.
Subclasses can catch this message, look at pArgs->child and return the layout constraints for known children.
If pArgs->relWin is not objNull, this uid will be used as the relative window. Otherwise, if pArgs->relWinTag will be used to find the relative window (i.e. relWinTag should be the window tag of the relative window). The relative window must be objNull (in which case the parent is used) or a sibling of pArgs->child.

status values

Messages from other classes

msgSave

Causes an object to file itself in an object file.
Takes P_OBJ_SAVE, returns STATUS.

Comments
clsCustomLayout will save the constraints for each child that has wsSendFile on in its WIN_METRICS.flags.style. If a child's constraint specifies a relWin that does not file, the relWin will be filed as objNull.
**msgRestore**

Creates and restores an object from an object file.

Takes `P_OBJ_RESTORE`, returns `STATUS`.

**Comments**

`clsCustomLayout` will restore the constraints for each child that was filed.

`clsCustomLayout` will self-send `msgWinSetLayoutDirty(true)` if the system font or system font scale changed since the table was filed. `pArgs->pEnv` is cast to a `P_WIN_RESTORE_ENV` and must be a valid window environment pointer.

**msgWinLayoutSelf**

Tell a window to layout its children (sent during layout).

Takes `P_WIN_METRICS`, returns `STATUS`.

**Comments**

`clsCustomLayout` responds by laying out its children. For each child, the following is done:

- `msgCstmLayoutGetChildSpec` is self-sent with the following `CSTM_LAYOUT_CHILD_SPEC` parameters:
  - `child` = child to be layed out;
  - `metrics` = result of `CstmLayoutSpecInit();`
  - `parentShrinkWrapWidth` = true if self can shrink wrap width;
  - `parentShrinkWrapHeight` = true if self can shrink wrap height;

Self can shrink wrap width/height if `pArgs->options` has `wsLayoutResizeOn` and self's `WinShrinkWrapWidth/Height(WIN_METRICS.flags.style)` is true.

The passed-back `pArgs` will be used as the child's layout spec.

- `msgBorderGetOuterOffsets` is sent to the child with a default `pArgs` (RECT32) of (1, 1, 1, 1). The outer offsets are used to define "after min edge" or "before max edge" constraints.

- The `x`, `y`, `w`, `h` of the child is computed based on its constraints. If the either `w` or `h` constraints are `clAsIs`, `msgWinGetDesiredSize` is sent to the child to determine its desired size.

If `pArgs->options` has `wsLayoutResizeOn` and self has shrink wrap width/height on, the bounding box around the layed out children will be computed and passed back in `pArgs->bounds.size`. If `style.limitToRootWin` is true, and self has no parent or self's parent is the `RootWindow`, the computed size will be limited to insure that self will fit on the `RootWindow` and self's origin may be altered (via `msgWinDelta`) to insure the window is fully on screen.

**Return Value**

- `stsCstmLayoutBadRelWin` The `relWin` specified for a child spec was not the uid of a sibling window.
- `stsCstmLayoutBadRelWinTag` The `relWinTag` specified for a child spec was not the tag of a sibling window.
- `stsCstmLayoutLoop` The specified set of child constraints results in a circular layout loop. For example, child A's width `clSameAs` child B's width and child B's width `clSameAs` child A's width.
- `stsCstmLayoutBadConstraint` A constraint specified for a child is not a valid value.
CLOSEBOX.H

This file contains the API definition for clsCloseBox.

clsCloseBox inherits from clsMenuButton.

Close boxes are frame decorations that let you close the frame. Close boxes paint as a triangle in the upper-left hand corner.

```c
ifndef CLOSEBOX_INCLUDED
define CLOSEBOX_INCLUDED
#include <rnbutton.h>
#endif
```

### Common #defines and typedefs

```c
#define hlpCloseBoxGeneral MakeTag(clsCloseBox, 1)
typedef OBJECT CLOSE_BOX;
typedef struct CLOSE_BOX_STYLE {
    U16 spare     : 16; // unused (reserved)
} CLOSE_BOX_STYLE, *P_CLOSE_BOX_STYLE;
```

### Messages

#### msgNew

Creates a closebox window.

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

```c
typedef struct CLOSE_BOX_NEW_ONLY {
    CLOSE_BOX_STYLE style;
    U32     spare; // unused (reserved)
} CLOSE_BOX_NEW_ONLY, *P_CLOSE_BOX_NEWONLY;
#define closeBoxNewFields 
    menuButtonNewFields \
    CLOSE_BOX_NEWONLY closeBox;
typedef struct CLOSE_BOX_NEW {
    closeBoxNewFields
    } CLOSE_BOX_NEW, *P_CLOSE_BOX_NEW;
```

#### msgNewDefaults

Initializes the `CLOSE_BOX_NEW` structure to default values.

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

```c
typedef struct CLOSE_BOX_NEW {
    closeBoxNewFields
} CLOSE_BOX_NEW, *P_CLOSE_BOX_NEW;
```

Comments

Zeroes out `pArgs->closeBox` and sets

```c
pArgs->win.flags.style &= ~(U32)(wsShrinkWrapWidth | wsShrinkWrapHeight);
```
pArgs->gWin.style.gestureEnable = false;
pArgs->gWin.helpId = hlpCloseBoxGeneral;

pArgs->border.style.edge = bsEdgeBottom;
pArgs->border.style.shadow = bsShadowNone;
pArgs->border.style.join = bsJoinSquare;
pArgs->border.style.leftMargin = bsMarginNone;
pArgs->border.style.rightMargin = bsMarginNone;
pArgs->border.style.bottomMargin = bsMarginNone;
pArgs->border.style.topMargin = bsMarginNone;

pArgs->button.style.feedback = bsFeedbackNone;

---

**msgCloseBoxGetStyle**

Passes back the current style values.

Takes **P_CLOSE_BOX_STYLE**, returns **STATUS**.

```c
#define msgCloseBoxGetStyle MakeMsg(clsCloseBox, 1)
```

### Arguments

```c
typedef struct CLOSE_BOX_STYLE {
    U16 spare : 16; // unused (reserved)
} CLOSE_BOX_STYLE, *P_CLOSE_BOX_STYLE;
```

---

**msgCloseBoxSetStyle**

Sets the style values.

Takes **P_CLOSE_BOX_STYLE**, returns **STATUS**.

```c
#define msgCloseBoxSetStyle MakeMsg(clsCloseBox, 2)
```

### Arguments

```c
typedef struct CLOSE_BOX_STYLE {
    U16 spare : 16; // unused (reserved)
} CLOSE_BOX_STYLE, *P_CLOSE_BOX_STYLE;
```
This file contains the API definition for clsCommandBar.
clsCommandBar inherits from clsTkTable.
Command bars are tkTables of buttons used in option sheets and frames.

```c
#ifndef CMDBAR_INCLUDED
#define CMDBAR_INCLUDED
#endif

#ifndef TKTABLE_INCLUDED
#include <tktable.h>
#endif
```

## Common #defines and typedefs

typedef OBJECT COMMAND_BAR;
typedef struct COMMAND_BAR_STYLE {
  U16 spare : 16; // unused (reserved)
} COMMAND_BAR_STYLE, *P_COMMAND_BAR_STYLE;

## Messages

### msgNew

Creates a command bar window.
Takes P_COMMAND_BAR_NEW, returns STATUS. Category: class message.

#### Arguments

typedef struct COMMAND_BAR_NEW_ONLY {
  COMMAND_BAR_STYLE style; // overall style
  U32 spare; // unused (reserved)
} COMMAND_BAR_NEW_ONLY, *P_COMMAND_BAR_NEW_ONLY;
#define commandBarNewFields \
  tkTableNewFields \
  COMMAND_BAR_NEW_ONLY commandBar;
typedef struct COMMAND_BAR_NEW {
  commandBarNewFields
} COMMAND_BAR_NEW, *P_COMMAND_BAR_NEW;

### msgNewDefaults

Initializes the COMMAND_BAR_NEW structure to default values.
Takes P_COMMAND_BAR_NEW, returns STATUS. Category: class message.

#### Arguments

typedef struct COMMAND_BAR_NEW {
  commandBarNewFields
} COMMAND_BAR_NEW, *P_COMMAND_BAR_NEW;

#### Comments

Sets
pArgs->gWin.style.gestureEnable = false;
pArgs->border.style.backgroundInk = bsInkGray33;
pArgs->border.style.topMargin = bsMarginMedium;
pArgs->border.style.bottomMargin = bsMarginMedium;
pArgs->border.style.leftMargin = bsMarginSmall;
pArgs->border.style.rightMargin = bsMarginSmall;

pArgs->tableLayout.style.tblXAlignment = tlAlignCenter;
pArgs->tableLayout.style.tblYAlignment = tlAlignCenter;
pArgs->tableLayout.style.childXAlignment = tlAlignCenter;
pArgs->tableLayout.style.childYAlignment = tlAlignCenter;
pArgs->tableLayout.style.growChildWidth = false;
pArgs->tableLayout.style.growChildHeight = true;

pArgs->tableLayout.numCols.constraint = tlInfinite;
pArgs->tableLayout.numRows.constraint = tlAbsolute;
pArgs->tableLayout.numRows.value = 1;
pArgs->tableLayout.colWidth.constraint = tlGroupMax;
pArgs->tableLayout.colWidth.gap = defaultColGap;
pArgs->tableLayout.rowHeight.constraint = tlChildrenMax;
pArgs->tableLayout.rowHeight.gap = 0;

Alters pArgs->tkTable.pButtonNew as in msgTkTableChildDefaults.

**msgCommandBarGetStyle**

Passes back the current style values.

Takes P_COMMAND_BAR_STYLE, returns STATUS.

```c
#define msgCommandBarGetStyle MakeMsg(clsCommandBar, 1)
```

**msgCommandBarSetStyle**

Sets the style values.

Takes P_COMMAND_BAR_STYLE, returns STATUS.

```c
#define msgCommandBarSetStyle MakeMsg(clsCommandBar, 2)
```

**Messages from Other Classes**

**msgTkTableChildDefaults**

Sets the defaults in pArgs for a common child.

Takes P.UNKNOWN, returns STATUS.

Comments

clsCommandBar sets up defaults for each child as follows:

If the child is a descendant of clsGWin, then

```c
pArgs->gWin.style.gestureEnable = false;
```

If the child is a descendant of clsButton, then

```c
pArgs->button.style.feedback = bsFeedback3D;
```
CONTROL.H

This file contains the API definition for clsControl.

clsControl inherits from clsBorder.

clsControl implements the previewing and client notification behavior of several UI components. clsControl is an abstract class -- it is never instantiated directly.

ציה Debugging Flags

The clsControl debugging flag is 'Char'. Defined values are:

flag8 (0x0100) msgControlEnable info
#ifndef CONTROL INCLUDED
#define CONTROL INCLUDED

#include <border.h>
#endif

Common #defines and typedefs

typedef OBJECT CONTROL;

Dynamic Enable Styles

Use one of these values in control’s style.dynamicEnable.

#define csDynamicNone 0 // no dynamic determination of "enabled"
#define csDynamicClient 1 // send msgControlProvideEnable to client
#define csDynamicObject 2 // send msgControlProvideEnable to "object"
#define csDynamicPargs 3 // set "enabled" from pargs

typedef struct CONTROL STYLE {
  U16 enable : 1, // if enabled, a control responds to input
  previewGrab : 1, // grab input when previews start
  previewRepeat : 1, // previews repeat on time-out
  previewing : 1, // msgControlBeginPreview has been sent out
  dirty : 1, // dirty status
  previewEnable : 1, // self-send msgControlBeginPreview, etc.
  showDirty : 1, // visuals reflect dirty state
  dynamicEnable : 2, // how "enable" value is determined
  private1 : 1, // reserved
  spare : 6; // unused (reserved)
} CONTROL STYLE, *P_CONTROL_STYLE;

Default CONTROL_STYLE:

enable = true
previewGrab = true
previewRepeat = false
previewing = false
dirty = false
previewEnable = false
showDirty = true
typedef struct CONTROL_STRING {
    P_CHAR pString;
    U16 len;
    U32 spare;  // unused (reserved)
} CONTROL_STRING, *P_CONTROL_STRING;

Advisory return values for subclasses
#define stsControlCancelPreview MakeWarning(clsControl, 13)
#define stsControlCancelRepeat MakeWarning(clsControl, 1)

msgNew

Creates a control window.

Takes P_CONTROL_NEW, returns STATUS. Category: class message.

Arguments
typedef struct CONTROL_NEWONLY {
    CONTROL_STYLE style;  // overall style
    OBJECT client;  // client to notify
    U32 spare;  // unused (reserved)
} CONTROL_NEWONLY, CONTROL_METRICS,
*P_CONTROL_NEWONLY, *P_CONTROL_METRICS;
#define controlNewFields
    borderNewFields
    CONTROL_NEWONLY control;
typedef struct CONTROL_NEW {
    controlNewFields
} CONTROL_NEW, *P_CONTROL_NEW;

Comments
Note that setting pArgs->control.style.enable to false does not result in pArgs->border.style.look set to bsLookInactive. If you change style.enable after msgNew (via msgControlSetStyle or msgControlSetEnable), the border.style.look will be changed to match.

msgNewDefaults

Initializes the CONTROL_NEW structure to default values.

Takes P CONTROL_NEW, returns STATUS. Category: class message.

Arguments
typedef struct CONTROL_NEW {
    controlNewFields
} CONTROL_NEW, *P_CONTROL_NEW;

Comments
Zeroes pArgs->control and sets
pArgs->win.flags.style |= wsFileInline;
pArgs->border.style.previewAlter = bsAlterBackground;
pArgs->border.style.selectedAlter = bsAlterBackground;
pArgs->control.style.enable = true;
pArgs->control.style.showDirty = true;

msgSave

Causes an object to file itself in an object file.

Takes P_OBJ_SAVE, returns STATUS.

Comments
If the client of the control is OSthisApp(), this is remembered and reinstated in msgRestore. In any case, the client is not saved.
msgRestore
Creates and restores an object from an object file.
Takes P_OBJ_RESTORE, returns STATUS.

Comments
clsControl restores the instance from the file. If the client of the control was OSThisApp() when filed, the client is set to OSThisApp(), otherwise objNull.

Messages Clients Send to Controls

msgControlGetMetrics
Passes back the current metrics.
Takes P_CONTROL_METRICS, returns STATUS.
#define msgControlGetMetrics MakeMsg(clsControl, 1)

msgControlSetMetrics
Sets the metrics.
Takes P_CONTROL_METRICS, returns STATUS.
#define msgControlSetMetrics MakeMsg(clsControl, 2)

msgControlGetStyle
Passes back the current style values.
Takes P_CONTROL_STYLE, returns STATUS.
#define msgControlGetStyle MakeMsg(clsControl, 3)

typedef struct CONTROL_STYLE {
    U16 enable : 1, // if enabled, a control responds to input
    previewGrab : 1, // grab input when previews start
    previewRepeat : 1, // previews repeat on time-out
    previewing : 1, // msgControlBeginPreview has been sent out
    dirty : 1, // dirty status
    previewEnable : 1, // self-send msgControlBeginPreview, etc.
    showDirty : 1, // visuals reflect dirty state
    dynamicEnable : 2, // how "enable" value is determined
    private1 : 1, // reserved
    spare : 6; // unused (reserved)
} CONTROL_STYLE, *P_CONTROL_STYLE;

msgControlSetStyle
Sets the style values.
Takes P_CONTROL_STYLE, returns STATUS.
#define msgControlSetStyle MakeMsg(clsControl, 4)

Message Arguments
typedef struct CONTROL_STYLE {
    U16 enable : 1, // if enabled, a control responds to input
    previewGrab : 1, // grab input when previews start
    previewRepeat : 1, // previews repeat on time-out
    previewing : 1, // msgControlBeginPreview has been sent out
    dirty : 1, // dirty status
    previewEnable : 1, // self-send msgControlBeginPreview, etc.
    showDirty : 1, // visuals reflect dirty state
    dynamicEnable : 2, // how "enable" value is determined
    private1 : 1, // reserved
    spare : 6; // unused (reserved)
} CONTROL_STYLE, *P_CONTROL_STYLE;
If `style.enable` changes, the control does the following:

- self-sends `msgBorderSetLook`, with `pArgs` of `bsLookActive` if `style.enable` is true, `bsLookInactive` otherwise.
- self-sends `msgControlCancelPreview`, `pNull` if `style.enable` is false.

### Comments

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>msgControlGetClient</code></td>
<td>Passes back <code>metrics.client</code>.</td>
</tr>
<tr>
<td><code>msgControlSetClient</code></td>
<td>Sets <code>metrics.client</code>.</td>
</tr>
<tr>
<td><code>msgControlGetDirty</code></td>
<td>Passes back true if the control has been altered since <code>dirty</code> was set false.</td>
</tr>
<tr>
<td><code>msgControlSetDirty</code></td>
<td>Sets <code>style.dirty</code>.</td>
</tr>
<tr>
<td><code>msgControlGetEnable</code></td>
<td>Passes back <code>style.enable</code>.</td>
</tr>
<tr>
<td><code>msgControlSetEnable</code></td>
<td>Sets <code>style.enable</code>.</td>
</tr>
</tbody>
</table>

### Macros

- `#define msgControlGetClient MakeMsg(clsControl, 5)`
- `#define msgControlSetClient MakeMsg(clsControl, 6)`
- `#define msgControlGetDirty MakeMsg(clsControl, 15)`
- `#define msgControlSetDirty MakeMsg(clsControl, 16)`
- `#define msgControlGetEnable MakeMsg(clsControl, 17)`
- `#define msgControlSetEnable MakeMsg(clsControl, 18)`
- `#define msgControlEnable MakeMsg(clsControl, 19)`

### For `msgControlEnable`

The control re-evaluates whether it is enabled.

`#define msgControlEnable MakeMsg(clsControl, 19)`
typedef struct CONTROL_ENABLE {
    WIN root;  // In: originator
    OBJECT object;  // In: object for msgControlProvideEnable
    BOOLEAN enable;  // In: value to use iff csDynamicPargs
    U32 spare;  // reserved (unused)
} CONTROL_ENABLE, *P_CONTROL_ENABLE;

This is commonly used with menu buttons that need to be enabled/disabled according to some constraints known to the sender. For example, clsMenuButton sends msgControlEnable to its menu before showing the menu, which results in each control in the menu receiving msgControlEnable with appropriate parameters. See msgMenuButtonShowMenu (mbutton.h) for sample usage.

clsControl responds to msgControlEnable as follows:

- If style.dynamicEnable is csDynamicNone, simply returns stsOK.
- If style.dynamicEnable is csDynamicPargs, style.enable is set to pArgs->enable.
- If style.dynamicEnable is csDynamicClient and metrics.client is objNull, does not change enable and returns stsOK.
- If style.dynamicEnable is csDynamicObject and pArgs->object is objNull, sets style.enable to false (as in msgControlSetEnable) and returns stsOK.

The cases that remain are style.dynamicEnable of csDynamicClient or csDynamicObject, and a non-null object.

- If the object is not owned by OSThisProcess(), sets style.enable to false (as in msgControlSetEnable) and returns stsOK. Otherwise, sends msgControlProvideEnable with the following CONTROL_PROVIDE_ENABLE parameters:

  root = pArgs->root;
  control = self;
  tag = self's WIN_METRICS.tag;
  enable = current value of style.enable;

- If the object responds to msgControlProvideEnable with stsNotUnderstood, sets style.enable to true (as in msgControlSetEnable) and returns stsOK. Otherwise, sets style.enable to CONTROL_PROVIDE_ENABLE.enable (as in msgControlSetEnable) and returns stsOK.

See Also

msgControlProvideEnable

Subclass Responsibility Messages

msgControlGetValue

Passes back the current "value" of the control.

Takes P_S32, returns STATUS.

#define msgControlGetValue MakeMsg(clsControl, 7)

Comments

In response to this message clsControl returns stsNotUnderstood.

msgControlSetValue

Sets the current "value" of the control.

Takes S32, returns STATUS.

#define msgControlSetValue MakeMsg(clsControl, 8)

Comments

In response to this message clsControl returns stsNotUnderstood.
Messages Controls Send to Self

**msgControlBeginPreview**
Self-sent when msgPenDown is received.
Takes P_INPUT_EVENT, returns STATUS. Category: self-sent.

```c
#define msgControlBeginPreview MakeMsg(clsControl, 10)
```

Comments
clsControl responds with stsOK. pArgs is pNull if the preview is not caused by an input event.

**msgControlUpdatePreview**
Self-sent when msgPenMoveDown is received.
Takes P_INPUT_EVENT, returns STATUS. Category: self-sent.

```c
#define msgControlUpdatePreview MakeMsg(clsControl, 11)
```

Comments
clsControl responds with stsOK. pArgs is pNull if the preview is not caused by an input event.

**msgControlRepeatPreview**
Self-sent if style.repeatPreview is true. Initial delay is 600ms, then immediate repeat until msgPenUp.
Takes P_INPUT_EVENT, returns STATUS. Category: self-sent.

```c
#define msgControlRepeatPreview MakeMsg(clsControl, 12)
```

Comments
clsControl responds with stsOK.
Subclasses can return stsControlCancelRepeat to prevent the next msgControlRepeatPreview.
pArgs is pNull if the preview is not caused by an input event.

**msgControlCancelPreview**
Self-sent when style.previewGrab is false and msgPenExitDown is received. Clients or subclasses can send this to a control to cancel existing preview.
Takes P_INPUT_EVENT, returns STATUS. Category: self-sent.

```c
#define msgControlCancelPreview MakeMsg(clsControl, 13)
```

Comments
Sets style. previewing to false.
pArgs is pNull if the preview is not caused by an input event.

**msgControlAcceptPreview**
Self-sent when msgPenUp is received.
Takes P_INPUT_EVENT, returns STATUS. Category: self-sent.

```c
#define msgControlAcceptPreview MakeMsg(clsControl, 14)
```

Comments
If gestures are enabled this message is not sent until msgGWinGesture is received with xgs1Tap.
clsControl responds with stsOK.
pArgs is pNull if the preview is not caused by an input event.
Messages Controls Send to Client

msgControlProvideEnable

Sent out to client or "object" during processing of msgControlEnable.

Takes P_CONTROL_PROVIDE_ENABLE, returns STATUS.

#define msgControlProvideEnable MakeMsg(clsControl, 20)

typedef struct CONTROL_PROVIDE_ENABLE {
  WIN root;      // In: originator
  CONTROL control; // In: sending control
  TAG tag;       // In: tag of sending control
  BOOLEAN enable; // In/Out: enabled value for control
  U32 spare;     // unused (reserved)
} CONTROL_PROVIDE_ENABLE, *P_CONTROL_PROVIDE_ENABLE;

Messages Defined by Other Classes

msgInputEvent

Notification of an input event.

Takes P_INPUT_EVENT, returns STATUS.

clsControl first calls ancestor, then responds as follows. (In each of these cases, see below for status return value.)

- If pArgs->flags has evBorderTaken set (see border.h), assumes clsBorder used the event and returns status.
- If style.enable is false, or style.previewEnable is false, or the event is not a pen event, returns status returned by ancestor.
- If pArgs->devCode is msgPenDown, self-sends msgControlBeginPreview passing along pArgs. If return status is stsControlCancelPreview, returns status. If style.previewRepeat is true, and return status is not stsControlCancelRepeat, the control repeats preview after 600ms delay. Sets style.previewing to true.
- If pArgs->devCode is msgPenMoveDown, self-sends msgControlUpdatePreview passing along pArgs. If return status is stsControlCancelPreview, sets style.previewing to false and returns status.
- If pArgs->devCode is msgPenUp, checks GWIN_STYLE.gestureEnable. If true, does nothing and returns status. Otherwise, self-sends msgControlAcceptPreview passing along pArgs and returns stsInputTerminate.
- If pArgs->devCode is msgPenExitDown and style.previewGrab is true or style.previewing is false or GWIN_STYLE.gestureEnable is true, does nothing and returns status. Otherwise, self-sends msgControlCancelPreview passing along pArgs and returns stsInputTerminate.

clsControl returns stsInputGrabTerminate if no error was encountered and style.previewing and style.previewGrab are true after processing the input event. Otherwise, the status returned by ObjectCallAncestor() is returned.
**msgGWinGesture**
Called to process the gesture.
Takes P_GWIN_GESTURE, returns STATUS.

Comments
If ObjectCallAncestor() returns stsOK, clsControl self-sends msgControlCancelPreview and returns stsOK.
If pArgs->msg is xgs1Tap and style.previewEnable is true, self-sends msgControlAcceptPreview and returns stsOK.
All other gestures result in msgGWinForwardedGesture to the control client, followed by msgControlCancelPreview to self.

**msgGWinAbort**
Clears the translation state of the GWin.
Takes void, returns STATUS.

Comments
clsControl responds to this by self-sending msgControlCancelPreview if the receiver is currently previewing.

**msgGWinGestureDone**
Sent to signal the end of a gesture.
Takes P_GWIN_GESTURE, returns STATUS. Category: self-sent.

Comments
clsControl responds to this by self-sending msgControlCancelPreview if the receiver is currently previewing.

**msgBorderGetDirty**
Passes back true if any child responds to msgBorderGetDirty with true; otherwise passes back false.
Takes P_BOOLEAN, returns STATUS.

Comments
clsControl responds by self-sending msgControlGetDirty. If the control is dirty, true is passed back. Otherwise, this message is passed on to clsControl's ancestor. clsBorder will respond by passing back true if any child of this control is dirty.

**msgBorderSetDirty**
Sends msgBorderSetDirty(pArgs) to each child.
Takes BOOLEAN, returns STATUS.

Comments
clsControl will call ancestor (to allow clsBorder to dirty any children), then self-send msgControlSetDirty(pArgs).
This file contains the API definition for clsCounter.

clsCounter inherits from clsTableLayout.

Counters are general components which display a current count and provide up/down arrows for the user to alter the count.

Counters are used as notebook frame decorations to provide up/down arrows to move between pages.

```c
#ifndef COUNTER_INCLUDED
#define COUNTER_INCLUDED

#include <tlayout.h>
#endif

Common #defines and typedefs
```

```c
#define tagCounterDecArrow MakeTag(clsCounter, 1)
#define tagCounterLabel MakeTag(clsCounter, 2)
#define tagCounterIncArrow MakeTag(clsCounter, 3)
#define hlpCounterDecArrow tagCounterDecArrow
#define hlpCounterLabel tagCounterLabel
#define hlpCounterIncArrow tagCounterIncArrow
typedef OBJECT COUNTER;
```

Show Style

```c
#define csShowCount 0 // show "count" only (e.g. "24")
#define csShowCountSlashTotal 1 // show "count/total" (e.g. "1/24")
#define csShowCountOfTotal 2 // show "count of total" (e.g. "1 of 24")

typedef struct COUNTER_STYLE {
  U16 numCols : 4, // number of columns for shrink-wrap
  show : 3, // what to show
  spare : 9; // unused (reserved)
} COUNTER_STYLE, *P_COUNTER_STYLE;
```

Messages

msgNew

Creates a counter window.

Takes P_COUNTER_NEW, returns STATUS. Category: class message.

```c
typedef struct COUNTER_NEW_ONLY {
  COUNTER_STYLE style;
  OBJECT client; // client to notify
  S32 value; // initial count
  S32 total; // total to display
  U32 spare1; // unused (reserved)
  U32 spare2; // unused (reserved)
} COUNTER_NEW_ONLY, *P_COUNTER_NEW_ONLY;
```
typedef struct COUNTER_NEW {
    counterNewFields
} COUNTER_NEW, *P_COUNTER_NEW;

msgNewDefaults

Initializes the COUNTER_NEW structure to default values.

Takes P_COUNTER_NEW, returns STATUS. Category: class message.

typedef struct COUNTER_NEW {
    counterNewFields
} COUNTER_NEW, *P_COUNTER_NEW;

msgCounterGetStyle

Passes back the current style values.

Takes P_COUNTER_STYLE, returns STATUS.

#define msgCounterGetStyle MakeMsg(clsCounter, 1)

typedef struct COUNTER_STYLE {
    U16 numCols : 4,  // number of columns for shrink-wrap
    show : 3,       // what to show
    spare : 9;      // unused (reserved)
} COUNTER_STYLE, *P_COUNTER_STYLE;

msgCounterSetStyle

Sets the style values.

Takes P_COUNTER_STYLE, returns STATUS.

#define msgCounterSetStyle MakeMsg(clsCounter, 2)

typedef struct COUNTER_STYLE {
    U16 numCols : 4,  // number of columns for shrink-wrap
    show : 3,       // what to show
    spare : 9;      // unused (reserved)
} COUNTER_STYLE, *P_COUNTER_STYLE;

Comments
If style.numCols requires the counter to be wider, clsCounter will self-send msgWinLayout to relayout.
msgCounterGetClient
Passes back the current counter client.
Takes P_OBJECT, returns STATUS.
#define msgCounterGetClient MakeMsg(clsCounter, 7)

msgCounterSetClient
Sets the client.
Takes OBJECT, returns STATUS.
#define msgCounterSetClient MakeMsg(clsCounter, 8)

msgCounterGetValue
Passes back the current count value.
Takes P_S32, returns STATUS.
#define msgCounterGetValue MakeMsg(clsCounter, 3)

msgCounterSetValue
Sets the current counter value.
Takes S32, returns STATUS.
#define msgCounterSetValue MakeMsg(clsCounter, 4)

Comments
If the new value requires the counter to be wider, clsCounter will self-send msgWinLayout to relayout.

msgCounterGetTotal
Passes back the current total value.
Takes P_S32, returns STATUS.
#define msgCounterGetTotal MakeMsg(clsCounter, 11)

msgCounterSetTotal
Sets the current total value.
Takes S32, returns STATUS.
#define msgCounterSetTotal MakeMsg(clsCounter, 12)

Comments
If the new total value requires the counter to be wider, clsCounter will self-send msgWinLayout to relayout.

msgCounterIncr
Increments the current counter value by adding in pArgs.
Takes S32, returns STATUS.
#define msgCounterIncr MakeMsg(clsCounter, 5)

Comments
If the new value requires the counter to be wider, clsCounter will self-send msgWinLayout to relayout.
**msgCounterGoto**

Sends `msgCounterNotify` to the counter's client to alter the counter's value.

Takes S32, returns STATUS.

```c
#define msgCounterGoto MakeMsg(clsCounter, 9)
```

**Comments**

`clsCounter` will send `msgCounterNotify` to the counter's client with the following `COUNTER_NOTIFY` parameters:

```c
counter = self;
initValue = current counter value;
action = csActionAccept;
value = pArgs;
```

The client can alter the value parameter to goto a different value, if desired.

A common use for this message is to create a menu with individual menu buttons representing particular counter values, and set the (msg, data) pair for each menu button to be (msgCounterGoto, desired value) and set the menu button's client to be the counter.

**msgCounterGetLabel**

Passes back the counter label window uid.

Takes P_WIN, returns STATUS.

```c
#define msgCounterGetLabel MakeMsg(clsCounter, 10)
```

**Comments**

The label is an instance of clsMenuButton, and can be given a menu by setting the CONTROL_STYLE.previewEnable to true and using msgMenuButtonSetMenu.

---

**Messages Counters Send to Clients**

**msgCounterNotify**

Sent to the client when an arrow repeats, finishes or cancels.

Takes P_COUNTER_NOTIFY, returns STATUS. Category: client notification.

```c
#define msgCounterNotify MakeMsg(clsCounter, 6)
```

**Arguments**

Enum16 (COUNTER_ACTION) {
  csActionIncrement = 0, // increment the counter
  csActionDecrement = 1, // decrement the counter
  csActionCancel = 2, // cancel the increment/decrement
  csActionAccept = 3, // accept the increment/decrement
};

typedef struct COUNTER_NOTIFY {
  OBJECT counter; // in: counter calling out
  S32 initValue; // in: initial value before repeat
  COUNTER_ACTION action; // in: what happened
  S32 value; // in/out: current value
  S32 total; // in: current total value
  U32 spare1; // unused (reserved)
  U32 spare2; // unused (reserved)
} COUNTER_NOTIFY, *P_COUNTER_NOTIFY;

**Comments**

If the user presses or continues to hold down on the decrement arrow, pArgs->action will be set to csActionDecrement.
If the user presses or continues to hold down on the increment arrow, pArgs->action will be set to csActionIncrement.

If the user pen's-up over either arrow, pArgs->action will be set to csActionAccept.

If the user drags out of either arrow, pArgs->action will be set to csActionCancel.

For any action, pArgs->value will be the current value of the counter and pArgs->initValue will be the initial value of the counter when the first csActionIncrement/Decrement was sent out.

Clients should change pArgs->value to the new desired value. Note that clsCounter does not change the value of the counter, other than copying back pArgs->value.

If pArgs->value is not changed by the client, the value of the counter will not be changed. This allows clients to use msgCounterIncr or msgCounterSetValue to alter the value during msgCounterNotify.
This file contains the API definition for clsField

clsField inherits from clsLabel.

Implements the UI component to edit, validate and display string data.

Fields implement the basic UI component to edit simple strings of text. The user-interface for fields has been optimized for simple short one row strings of text, although they will function for multiple lines. All display information for translated fields is handled in clsLabel. Typically the label layout is fixed, and shrink wrap will be turned off in the label. Otherwise the field size will change as the value of the string changes, and lead to strange results and behavior. There are three basic User-Interfaces supported through the API to edit fields. These are defined in field.style.editType.

Fields with editType of fstInline support direct writing, appending, and a number of gestural editing operations, including bringing up an IP. Fields with editType of fstPopUp will only allow editing through an IP. Fields with editType of fstOverWrite make the field combed and allow over-writing on individual characters. These fields have very precise stroke targeting due to the character box constraints. This, in combination with only allowing three editing gestures (insert space, delete range, and delete character) allows for highly accurate handwriting and gesture recognition and for quick correction of mistakes. The downside of this style of field is that a specific UI look is implied.

To further increase recognition accuracy, fields require a translator for both inline editing and in the IP. Translators have a rich API to provide various types of contextual information. This greatly increases translation accuracy. See msgNew, msgFieldGetXlate, msgFieldSetXlate, msgFieldCreateTranslator.

Fields can also be run in delayed mode. Delayed fields allow the user to write into an empty field, and not translate the strokes on pen out of proximity. Delayed fields are translated when msgFieldTranslateDelayed is sent to the field. See msgFieldTranslateDelayed, msgFieldSetDelayScribble, and msgFieldGetDelayScribble for more information.

Fields will replace newLines with spaces, and will strip trailing spaces when their value is retrieved. The value should be set via msgLabelSetString and retrieved via msgLabelGetString.

Messages from clsInput, messages from clsGWIn (other then msgGWInGesture), messages from clsWin, messages from clsLabel, messages from clsSelection, messages from clsXfer, messages from xlate, and messages from clsTracker should NOT be overridden by subclasses of clsField.

Finally, fields provide simple hooks to allow clients or subclasses to perform various validation according to a common protocol. See msgFieldValidate for details.

 ifndef FIELD_INCLUDED
 #define FIELD_INCLUDED

 #include <go.h>
 #include <label.h>
 #include <xtemplt.h>

 // Next Up: 31  Recycled: 28
Common \#defines and typedefs

typedef OBJECT FIELD;

Field Editing Types

These define the types of edit User-Interface the field provides, defining the behavior of the field. These are used for style.editType.

\#define fstInline 1 // Direct editing on field, or through IP
\#define fstPopUp 2  // Editing only through an IP
\#define fstOverWrite 3 // Editing in combed overwrite field

Insertion Pad Types

These define the type of Insertion Pad that will be created in msgFieldCreatePopUp when the type parameter is fipReplaceAll. Note: A call to msgFieldCreatePopUp when the type parameter is fipInsert will look at the system preferences to determine the type of IP. These are used for style.popUpType.

//\#define fstEditBox 1 // Obsolete
\#define fstCharBox 2  // The pop-up is an ipsCharBox IP
\#define fstCharBoxButtons 2 // Obsolete

Character Box Memory

For fstOverWrite fields, this defines the number of characters that should be used sent to the translator via msgXlateCharMemorySet. This causes the translator to cycle through choices and not return the same character from a translation. These are use for style.boxMemory.

\#define fstBoxMemoryZero 0  // Box memory is zero characters
\#define fstBoxMemoryOne 1   // Box memory is one character
\#define fstBoxMemoryFour 2  // Box Memory is four characters

Selection/Input Target

These define the interaction the field should have with both the selection manager and the input target when:

- msgFieldKeyboardActivate is called
- the pen is interacting with the field
- msgFieldTranslateDelayed is called
- the field is the recipient of a move/copy operation

These are used for style.focusStyle.

\#define fstInputSelection 1  // Field takes selection and input target
\#define fstInput 2           // Field takes input target only
\#define fstNone 3           // Field takes neither selection nor target

UpperCase Writer Rules

These define the capitalization heuristic rules used by the field translator. These rules do not apply when the translator is provided by the client of the field, or the writer is not an all-caps writer. These are used for style.capOutput.

\#define fstCapAsIs 1
\#define fstCapFirstWord 2
\#define fstCapAllWords 3
\#define fstCapAll 4
# Translator Type

These define the type of translator given to and maintained by the field, and affects the parameters to `msgFieldGetTranslator` and `msgFieldSetTranslator`, the interaction with `msgFieldCreateTranslator`, and `msgNew`. See these messages for more information. These are used for style.xlateType.

```c
#define fstXlateObject 0
#define fstXlateTemplate 1
```

# Field Style Structure

The field style structure defines the overall behavior of the field. Information on the various flags can be found elsewhere. For information on focusStyle, capOutput, popUpType, editType, xlateType, delayed and boxMemory, see above.

For information on noSpace and veto, see `msgFieldCreateTranslator`.

```c
typedef struct FIELD_STYLE {
    U16 focusStyle: 2, // How field does selection and target
    capOutput: 3, // Upper case writer cap rules for xlate
    popUpType: 3, // Insertion pad style for fipReplaceAll
    editType: 2, // Type of editing in field
    xlateType: 1, // 0=xlate object, 1=xtemplate
    clientValidate: 1, // client performs validation
    clientPreValidate: 1, // Notify client before validation
    clientPostValidate: 1, // Notify client after successful valid
    clientNotifyInvalid: 1, // Notify client when invalid
    clientNotifyModified: 1, // Notify client when attempt to modify
    validatePending: 1, // Field not valid since last modification
    delayed: 1, // Delayed translation field. Capture
               // strokes till msgFieldTranslateDelayed
    upperCase: 1, // Field and IP forced to upper case
    noSpace: 1, // Turn on no space in fld created xlate
    privateData1: 1, // Internal use only
    veto: 1, // Turn on veto in fld created xlate
    privateData2: 1, // Internal use only
    boxMemory: 2, // Enable box memory in field and IP
    dataMoveable: 1, // Reserved for future use
    dataCopyable: 1,
    reserved: 5;
} FIELD_STYLE, *P_FIELD_STYLE;
```

# Popup Editing Types

These defines are parameters in `msgFieldCreatePopUp` and `msgFieldActivatePopUp`. They specify what type of edit operation should be performed by this pop-up. Internally, an edit gesture (circle) in an `fstInline` field or pen input into `fstPopUp` field will call these messages with `fipReplaceAll`. An insert caret in an `fstInline` field will call with `fipInsert`.

```c
#define fipReplaceAll 0 // The IP displays/edits the field value
#define fipInsert 1 // The IP inserts new text at the insertion pt
#define fipReplaceRange 2 // Unimplemented
```

# Validation data structure

This data structure is used as a parameter to `msgFieldValidateEdit`, and `msgFieldNotifyInvalid` to capture all validation information.

```c
typedef struct {
    MESSAGE failureMessage; // Reason validation failed
    OBJECT field; // Field to validate
} FIELD_NOTIFY, *P_FIELD_NOTIFY;
```
Messages

msgNew

Creates and initializes a new instance of clsField.

Takes P_FIELD_NEW, returns STATUS. Category: class message.

Arguments

typedef union FIELD_XLATE {
    OBJECT translator;
    pXTM_ARGS pTemplate;
} FIELD_XLATE, *P_FIELD_XLATE;

typedef struct FIELD_XLATE {
    FIELD_STYLE style; // field style, see above
    FIELD_XLATE xlate;  // xlate object or template
    U16 maxLen;        // maximum field string length. 0 means no limit
    U32 reserved;      // reserved for future use, must be 0
} FIELD_XLATE, *P_FIELD_XLATE;

typedef struct FIELD_NEW_ONLY {
    FIELD_XLATE xlate; // xlate object or template
    U16 maxLen;        // maximum field string length. 0 means no limit
} FIELD_NEW_ONLY, *P_FIELD_NEW_ONLY;

#define fieldNewFields    
labelNewFields         
FIELD_NEW_ONLY      field;

typedef struct FIELD_NEW {
    fieldNewFields
} FIELD_NEW, *P_FIELD_NEW;

Comments

Will force the label.style to IsBoxTicks for fields of editType fstOverWrite. Overwrite fields must have label style of IsBoxTicks. Will force gWin.style.gestureEnable to TRUE. Extreme care should be taken if changing either of these. The xlate parameter in conjunction with style.xlateType specifies the type of translator the field uses. If xlateType is 0, and pNew->field.xlate.translator does not equal objNull, the translator will be used for all translations in the field and in the IP, and destroyed when the field is destroyed. If xlateType is 1, pNew->field.xlate.pTemplate is used to create, allocate, and compile a template. It will also be freed when the field is destroyed. A translator will be created and destroyed as needed via msgFieldCreateTranslator from this compiled template. msgFieldCreateTrans will also be used when xlateType is 0 and pNew->field.xlate.translator is objNull.

See Also

msgFieldSetXlate

msgNewDefaults

Initializes the FIELD_NEW structure to default values.

Takes P_FIELD_NEW, returns STATUS. Category: class message.

Message

typedef struct FIELD_NEW {
    fieldNewFields
} FIELD_NEW, *P_FIELD_NEW;

Comments

Initializes the default values. Care should be taken when changing the default values of parent classes. Examples are win.flags.input, or gwin.style.

Zeros out pNew->field and sets

    fld.field.style.dataMoveable = true;
    fld.field.style.dataCopyable = true;
    fld.field.style.focusStyle = fstInputSelection;
    fld.field.style.capOutput = fstCapAsIs;
    fld.field.style.editType = fstInline;
    fld.field.style.popUpType = fstCharBoxButtons;
    fld.field.style.xlateType = fstXlateObject;
    fld.field.style.boxMemory = fstBoxMemoryFour;
    fld.field.maxLen = 64;
fld.border.style.edge = bsEdgeBottom;
fld.gwin.style.firstEnter = TRUE;
fld.gwin.style.askOtherWin = TRUE;
fld.gwin.style.otherWinSaysYes = TRUE;
fld.win.flags.input = inputTip | inputStroke |
   inputOutProx | inputInk | inputEnter |
   inputHoldTimeout | inputLRContinue |
   inputAutoOfTerm | inputTimeout | inputHWTimeout;

msgFieldGetStyle
Passes back the style value held by the field.
Takes P_FIELD_STYLE, returns STATUS.

#define msgFieldGetStyle MakeMsg(clsField, 1)

typedef struct FIELD_STYLE {
  U16 focusStyle: 2,    // How field does selection and target
      capOutput: 3,    // Upper case writer cap rules for xlate
      popUpType: 3,    // Insertion pad style for fipReplaceAll
      editType: 2,     // Type of editing in field
      xlateType: 1,    // 0=xlate object, 1=xtemplate
      clientValidate: 1, // client performs validation
      clientPreValidate: 1, // Notify client before validation
      clientPostValidate: 1, // Notify client after successful valid
      clientNotifyInvalid: 1, // Notify client when invalid
      clientNotifyReadOnly: 1; // Notify client when attempt to modify
                      // readonly field
  U16 clientNotifyModified: 1, // Notify client when field modified
      validatePending: 1,    // Field not valid since last modification
      delayed: 1,    // Delayed translation field. Capture
                     // strokes till msgFieldTranslateDelayed
      upperCase: 1,     // Field and IP forced to upper case
      noSpace: 1,       // Turn on no space in fld created xlate
      privateData1: 1,  // Internal use only
      veto: 1,          // Turn on veto in fld created xlate
      privateData2: 1,  // Internal use only
      boxMemory: 2,     // Enable box memory in field and IP
      dataMoveable: 1,  // Reserved: 5; // Reserved for future use
      dataCopyable: 1,
      reserved: 5;
} FIELD_STYLE, *P_FIELD_STYLE;

msgFieldSetStyle
Sets the style of the field.
Takes P_FIELD_STYLE, returns STATUS.

#define msgFieldSetStyle MakeMsg(clsField, 2)

typedef struct FIELD_STYLE {
  U16 focusStyle: 2,    // How field does selection and target
      capOutput: 3,    // Upper case writer cap rules for xlate
      popUpType: 3,    // Insertion pad style for fipReplaceAll
      editType: 2,     // Type of editing in field
      xlateType: 1,    // 0=xlate object, 1=xtemplate
      clientValidate: 1, // client performs validation
      clientPreValidate: 1, // Notify client before validation
      clientPostValidate: 1, // Notify client after successful valid
      clientNotifyInvalid: 1, // Notify client when invalid
      clientNotifyReadOnly: 1; // Notify client when attempt to modify
                      // readonly field
  U16 clientNotifyModified: 1, // Notify client when field modified
      validatePending: 1,    // Field not valid since last modification
      delayed: 1,    // Delayed translation field. Capture
                     // strokes till msgFieldTranslateDelayed
      upperCase: 1,     // Field and IP forced to upper case
      noSpace: 1,       // Turn on no space in fld created xlate
      privateData1: 1,  // Internal use only
      veto: 1,          // Turn on veto in fld created xlate
      privateData2: 1,  // Internal use only
      boxMemory: 2,     // Enable box memory in field and IP
      dataMoveable: 1,  // Reserved: 5; // Reserved for future use
      dataCopyable: 1,
      reserved: 5;
} FIELD_STYLE, *P_FIELD_STYLE;
### Comments

- **validatePending**: 1, /* Field not valid since last modification
- **delayed**: 1, /* Delayed translation field. Capture strokes till msgFieldTranslateDelayed
- **upperCase**: 1, /* Field and IP forced to upper case
- **noSpace**: 1, /* Turn on no space in fld created xlate
- **privateData1**: 1, /* Internal use only
- **veto**: 1, /* Turn on veto in fld created xlate
- **privateData2**: 1, /* Internal use only
- **boxMemory**: 2, /* Enable box memory in field and IP
- **dataMoveable**: 1,
- **dataCopyable**: 1,
- **reserved**: 5; /* Reserved for future use

### Return Value

- **stsFailed** The field is currently being edited. This is either through the pen, or a pop up IP.

---

### msgFieldGetXlate

Passes back the translator information for the field.

Takes P_UNKNOWN, returns STATUS.

```c
#define msgFieldGetXlate MakeMsg(clsField, 3)
```

### Comments

- If `xlateType` is 0, the parameter is assumed to be a P_OBJECT and the translator object id is returned. Otherwise the parameter is assumed to be a P_UNKNOWN and the COMPILED template is returned.

### See Also

- `xtemplate.h.h`

---

### msgFieldSetTextXlate

Specifies the translator information for the field.

Takes P_UNKNOWN, returns STATUS.

```c
#define msgFieldSetTextXlate MakeMsg(clsField, 4)
```

### Comments

- If `xlateType` is 0, the argument is assumed to be P_OBJECT being a translator. The old translator is not destroyed. If `xlateType` is 1, the argument is assumed to be an uncompiled template (P_XTM_ARGS). The field code will compile the template and use it to create a translator. Any old compiled template will not be freed, and must be done so by a call to XTemplateFreeO by the client. Calling on a delayed field will cancel the delayed field, destroying any scribbles captured by the field.

### Return Value

- **stsFailed** The field is currently being edited with the pen, or through an IP.

### See Also

- `msgFieldCreateTranslator.h.h`

---

### msgFieldGetMaxLen

Passes back the maximum length allowed for input in the field.

Takes P_U16, returns STATUS.

```c
#define msgFieldGetMaxLen MakeMsg(clsField, 5)
```
**msgFieldSetMaxLen**
Sets the Maximum length for input in the field.
Takes P_U16, returns STATUS.

```c
#define msgFieldSetMaxLen MakeMsg(clsField, 6)
```

Comments
Sets the limit for the number of characters that are allowed in a field. If `maxLen` is 0, the `maxLen` is assumed to be a `maxU16`. However, it is not recommended that fields of that size be created. If the value is less than the old value, the value displayed in the field will be truncated to the new value during the next edit.

**msgFieldSetCursorPosition**
Sets the cursor position of the keyboard insertion point in the field.
Takes P_U16, returns STATUS.

```c
#define msgFieldSetCursorPosition MakeMsg(clsField, 7)
```

Comments
The cursor position will not be displayed unless the field has the input target. As a performance optimization, this message is not self-sent to set the cursor position.

**msgFieldGetCursorPosition**
Passes the current keyboard insertion cursor position in the field.
Takes P_U16, returns STATUS.

```c
#define msgFieldGetCursorPosition MakeMsg(clsField, 8)
```

Comments
If no cursor position has been set, 0 is returned. As a performance optimization, this message is not self-sent to inquire cursor position.

---

**Insertion Pad Messages**

**msgFieldActivatePopUp**
Called to cause an Insertion pad to be brought up for the field.
Takes P_FIELD_ACTIVATE_POPUP, returns STATUS.

```c
#define msgFieldActivatePopUp MakeMsg(clsField, 18)
```

Arguments
typedef struct {
  U16  type;
  P_RECT32 pRect;
  U32 reserved;
} FIELD_ACTIVATE_POPUP, * P_FIELD_ACTIVATE_POPUP;

Comments
If `msgFieldActivate` has not been called (due to pen input into the field) it will be called. Will bring the up the IP at the passed in `pRect` location. If NULL, the IP will be centered over the field. The type of IP will be passed to `msgFieldCreatePopUp`. Will return `stsFailed` if the pop-up is not valid given the type and state of the field. For example, an `fipInsert` on a filled to `maxLen` field will return `stsFailed`.

Return Value
`stsFailed` A popup up could not be created given the state of the field.
**msgFieldAcceptPopUp**
Causes the Insertion pad to be accepted.
Takes void, returns STATUS.

```
define msgFieldAcceptPopUp MakeMsg(clsField, 19)
```

**Comments**
Called when the user collapses the insertion pad by hitting the OK button or accepts the IP. Can be called programatically as well.

**msgFieldCancelPopUp**
Cancels the edit in the pop-up insertion pad.
Takes void, returns STATUS.

```
define msgFieldCancelPopUp MakeMsg(clsField, 20)
```

**Comments**
Causes the old value to be preserved unchanged. Called when the user hits the cancel button or cancels the IP. Can be called programatically as well.

**msgFieldCreatePopUp**
Creates and passes back the insertion pad when the pop up is invoked.
Takes P_FIELD_CREATE_POPUP, returns STATUS.

```
define msgFieldCreatePopUp MakeMsg(clsField, 27)
```

**Arguments**
```typedef struct {
    U16    type;
    OBJECT ip;
    U32    reserved;
} FIELD_CREATE_POPUP, *P_FIELD_CREATE_POPUP;```

**Comments**
Will create the insertion pad for use in the field. If type is fipReplaceAll, will look at style.popUpType to determine the type of IP to create. If type is fipInsert, will look at the system preferences for writing style and create the appropriate type of Insertion pad. Will return stsFailed if the type is fipInsert and the field data length is equal to maxLen.

**Return Value**
stsFailed  The pop-up could not be created for the field.

---

**Delayed Field Messages**

**msgFieldTranslateDelayed**
Translates a field with delayed captured strokes.
Takes NULL, returns STATUS.

```
define msgFieldTranslateDelayed MakeMsg(clsField, 25)
```

**Comments**
Causes translation to occur for a field that has style.delayed and has captured strokes pending translation. Returns stsMessageIgnored if style.delayed is not set, or if there is no pending translation.

**Return Value**
stsMessageIgnored  The field did not have a delayed scribble to translate.
**msgFieldGetDelayScribble**

Returns the delayed scribble for delayed fields.

Takes P_OBJECT, returns STATUS.

```c
#define msgFieldGetDelayScribble MakeMsg(clsField, 26)
```

**Return Value**

- `stsMessageIgnore` The field did not have a delayed scribble to translate. Either not a delayed field or no scribbles in the field.

**msgFieldSetDelayScribble**

Puts the field in delayed mode with the given scribble.

Takes P_OBJECT, returns STATUS.

```c
#define msgFieldSetDelayScribble MakeMsg(clsField, 30)
```

**Return Value**

- `stsFailed` The field is currently being edited. This is either through the pen, an IP, or the field contains delayed strokes in delayed mode. Undefined behavior if called on a field with delayed scribbles.

### Miscellaneous Messages

**msgFieldClear**

Clears the value of the field.

Takes NULL, returns STATUS.

```c
#define msgFieldClear MakeMsg(clsField, 29)
```

**Comments**

Clears the delay scribble if one exists, otherwise clears the value of the field.

**msgFieldReadOnly**

Self called when an attempt is made to modify a read only field.

Takes self, returns STATUS.

```c
#define msgFieldReadOnly MakeMsg(clsField, 21)
```

**Comments**

Will send `msgFieldReadOnly` to control.client if `clientNotifyReadOnly` is set. it exists.

**msgFieldModified**

Self called when a field is modified.

Takes self, returns STATUS.

```c
#define msgFieldModified MakeMsg(clsField, 22)
```

**Comments**

If the control.dirty bit is clear and the `clientNotifyModified` bit is set, will send `msgFieldModified` to control.client. Will set the control.dirty bit. It is the clients responsibility to clear this bit. Will also set the `validatePending` bit. This bit is cleared after successful validation.

**msgFieldKeyboardActivate**

Activates field for keyboard use.

Takes void, returns STATUS.

```c
#define msgFieldKeyboardActivate MakeMsg(clsField, 23)
```
Called by client whenever the field is activated for use with the keyboard. Primarily useful for item managers that are dealing with keyboard navigation between fields.

**msgFieldCreateTranslator**

Self called to create a translator. Passes back the translator.

Takes P_OBJECT, returns STATUS.

```
define msgFieldCreateTranslator      MakeMsg(clsField, 15)
```

Comments

Used to create the translator based on the compiled template. Called when xlate.xlateType = 1 or when xlate.xlateType = 0 and xlate.translator = NULL to create the translator. Will create the translator and respect the style.noSpace, style.veto, and style.capOutput settings (for all caps writers). This translator will be destroyed when msgFieldDeactivate is called.

**Validation Messages**

**msgFieldValidate**

Performs the validation protocall for a field.

Takes void, returns STATUS.

```
define msgFieldValidate      MakeMsg(clsField, 9)
```

Comments

Forces validation of a field. Called when the field loses the input target and validatePending is TRUE. Also called when translation is completed in a previously empty field. Returns non-error status for failed validation, or stsOK for a valid field.

- calls msgFieldPreValidate on client if field.style.clientPreValidate
- calls msgFieldValidateEdit on client or on self, depending on style.clientValidate
- calls msgFieldNotifyInvalid if msgFieldValidateEdit returns > stsOK
- calls msgFieldPostValidate on client if field.style.clientPostValidate and msgFieldValidateEdit returns stsOK
- calls msgFieldFormat to format the field if msgFieldValidateEdit returns stsOK.
- sets the validatePending bit to 0

See Also

msgFieldValidateEdit

**msgFieldPreValidate**

Called on client if the field.style.clientPreValidate is set before validation.

Takes self, returns STATUS.

```
define msgFieldPreValidate      MakeMsg(clsField, 10)
```

Comments

Called on the control.client if clientPreValidate is set before validation. Allows clients to pre-process the value of a field before validation occurs.

**msgFieldValidateEdit**

Self call to perform validation on the field.

Takes P_FIELD_NOTIFY, returns STATUS.

```
define msgFieldValidateEdit      MakeMsg(clsField, 11)
```
typedef struct {
    MESSAGE failureMessage; // Reason validation failed
    OBJECT field; // Field to validate
} FIELD_NOTIFY, *P_FIELD_NOTIFY;

Called on self if clientValidate is false, or on the client if clientValidate is set. Returns stsOK when successful. Puts a failure message in the failureMessage field of P_FIELD_NOTIFY if not successful, and returns a non-error return code. Default returns stsOK.

msgFieldNotifyInvalid
Called to notify a field was invalid.
Takes P_FIELD_NOTIFY, returns STATUS.
#define msgFieldNotifyInvalid MakeMsg(clsField, 12)

Called on client if field.style.notifyInvalid bit is set and the msgFieldValidateEdit returns a > stsOK return code. Allows clients to post a failure message for validation.

msgFieldPostValidate
Self call to perform post-validation processing.
Takes self, returns STATUS.
#define msgFieldPostValidate MakeMsg(clsField, 13)

Called on client if field.style.clientPostValidate is set. Only called if msgFieldValidateEdit returns stsOK. Allows client to perform post validation processing.

msgFieldFormat
Self call to perform formatting.
Takes void, returns STATUS.
#define msgFieldFormat MakeMsg(clsField, 14)

Self called after validation to perform any formatted the field requires to display itself correctly. Intended to be overridden by clients to support field formatting. Only called when msgFieldValidateEdit returns stsOK.

Messages from other classes

msgFree
Defined in object.h.
Takes OBJ_KEY, returns STATUS.

Deactivates the field if necessary. Will free the translator if xlateType is 0 and a translator was handed to the field. Will free the compiled template if xlateType is 1. Inherits ancestor behavior.

See Also
object.h
**msgSave**
Defined in object.h.
Takes P_OBJ_SAVE, returns STATUS.

Comments
Inherits ancestor behavior first and then stores in the resource file all information about the current state of the field, including the translator or template information or the delayed strokes the field contains. Fields will not save any information about a current editing operation (through a pop-up, keyboard, or pen) in effect.

See Also
object.h

**msgRestore**
Defined in object.h.
Takes P_OBJ_RESTORE, returns STATUS.

Comments
Inherits ancestor information and restores all information about the field including translator information or the delayed strokes the field contains.

See Also
msgSave.h

**msgIPDataAvailable**
Defined in insert.h.
Takes OBJECT, returns STATUS.

Comments
Sent to the field from an insertion pad when there is data to retrieve from the pop-up pad. Depending on the operation that brought up the pad (an insert or edit gesture), will either insert the text from the pad at the current insertion point, or replace the value of the field with the IP value. Will destroy the pop-up pad created.

See Also
insert.h

**msgIPC Cancelled**
Defined in insert.h.
Takes OBJECT, returns STATUS.

Comments
Sent to the field when the insertion pad has been canceled. Will destroy the pad and any changes to the text in the pad are ignored.

See Also
insert.h

**msgControlSetDirty**
Defined in control.h.
Takes BOOLEAN, returns STATUS.

Comments
Inherits behavior from superclass. Will clear all character box memory stored for an overwrite field, allowing characters to be returned immediately from the translator.

See Also
control.h
FONTLBOX.H

This file contains the API for clsFontListBox.

clsFontListBox inherits from clsStringListBox.

Provides a listbox that is based on the list of currently installed fonts.

```c
#ifndef FONTLBOX_INCLUDED
#define FONTLBOX_INCLUDED

#ifndef STRLBOX_INCLUDED
#include <strlbox.h>
#endif
```

- **Common #defines and typedefs**

  ```c
typedef struct {
    U16 prune : 16; // FIM_PRUNE_CONTROL (see fontmgr.h)
    U16 spare : 16; // reserved
  } FONTLB_STYLE, *P_FONTLB_STYLE;

  Default FONTLB_STYLE:
  prune = fimNoPruning (see fontmgr.h)
  ```

  ```c
  #define fontListBoxNewFields 
      stringListBoxNewFields 
      FONTLB_NEW_ONLY
  #define fontListBox;
  ```

  ```c
typedef struct {
    FONTLB_STYLE style; // overall style
    U32 spare; // reserved
  } FONTLB_NEW_ONLY, *P_FONTLB_NEW_ONLY;

  typedef struct {
    FONTLB_NEW_ONLY, *P_FONTLB_NEW_ONLY
  } FONTLB_NEW, *P_FONTLB_NEW;
  ```

- **msgNew**

  Creates a font list box window.

  Takes P_FONTLB_NEW, returns STATUS. Category: class message.

  Arguments

  ```c
typedef struct {
    FONTLB_STYLE style; // overall style
    U32 spare; // reserved
  } FONTLB_NEW_ONLY, *P_FONTLB_NEW_ONLY;
```

  ```c
  #define fontListBoxNewFields 
      stringListBoxNewFields 
      FONTLB_NEW_ONLY
  #define fontListBox;
  ```

  ```c
typedef struct {
    FONTLB_NEW_ONLY, *P_FONTLB_NEW_ONLY
  } FONTLB_NEW, *P_FONTLB_NEW;
  ```

  Comments

  In response to msgNew, clsFontListBox will set pArgs->listBox.nEntries to zero and then call ancestor.

  It will then use msgFIMGetInstalledIdList to get the list of fonts currently installed in the system. For each font, clsFontListBox will add an entry using msgListBoxInsertEntry that has 'freeEntry' set to lbFreeDataDefault and 'data' set to the IM_HANDLE of the font.

  As a last step, the new listBox instance will be added as an observer of theInstalledFonts.

  We recommend that clients set pArgs->listBox.style.filing = lbFileMin to avoid unexpected results after a font listBox has been restored. See the documentation for msgRestore below.

  See Also

  msgFIMGetInstalledIdList obtain the short IDs of all installed fonts.
msgNewDefaults
Initializes the FONTLB_NEW structure to default values.
Takes P_FONTLB_NEW, returns STATUS. Category: class message.

Message Arguments
typedef struct {
    fontListBoxNewFields
} FONTLB_NEW, *P_FONTLB_NEW;
Comments
Zeroes out pArgs->fontListBox and sets:
pArgs->stringListBox.style.role = slbRoleChoice01;

msgFontListBoxGetStyle
Gets the style of a font listbox.
Takes P_FONTLB_STYLE, returns STATUS.
#define msgFontListBoxGetStyle
MakeMsg(clsFontListBox, 1)
Message Arguments
typedef struct {
    U16 prune : 16; // FIM_PRUNE_CONTROL (see fontmgr.h)
    U16 spare : 16; // reserved
} FONTLB_STYLE, *P_FONTLB_STYLE;

Messages from Other Classes

msgFree
Sent as the last of three msgs to destroy an object.
Takes OBJ_KEY, returns STATUS.
Comments
The receiver will remove itself as an observer of theInstalledFonts.

msgRestore
Creates and restores an object from an object file.
Takes P_OBJ_RESTORE, returns STATUS.
Comments
clsFontListBox responds by restoring its style values and resynchronizing its entries with respect to the
list of installed fonts, as is done in msgNew. The restored instance is added as an observer of
theInstalledFonts.

Note that this new information may differ from that which had been used the last time the listBox was
saved, because the list of fonts installed in the system may have changed. Depending on how clsListBox
filed its entry data, this may lead to odd behavior. The best approach is to use a LIST_BOX_STYLE.filing
of lbFileMin so that clsListBox won’t file any entry information or windows. Because after msgRestore
the value obtained via msgStrListBoxGetValue may no longer match any entry, clients should use
msgStrListBoxSetValue to change the value to a short ID from the new list of installed fonts.

msgSave
Causes an object to file itself in an object file.
Takes P_OBJ_SAVE, returns STATUS.
Comments
clsFontListBox responds by writing out its style values.
msgStrListBoxGetValue
Passes back the value of a string listbox.
Takes P_U32, returns STATUS.

Comments
c1sFontListBox responds by calling ancestor, converting the resulting IM_HANDLE *pArgs into the FIM_SHORT_ID via msgFIMGetld, and setting *pArgs to this short id.

msgStrListBoxSetValue
Sets the value of a string listbox whose role is one of slbRoleChoice*.
Takes U32, returns STATUS.

Comments
c1sFontListBox responds by converting the incoming pArgs from a FIM_SHORT_ID into the IM_HANDLE for the font (msgFIMFindId) and then calling ancestor with this new pArgs.

msgStrListBoxProvideString
This message requests the client (or subclass) to provide a string.
Takes P_STRLB_PROVIDE, returns STATUS. Category: self-sent/client responsibility.

Comments
c1sFontListBox first checks whether pArgs->position is >= the number of fonts described by its cached information. If so, c1sFontListBox returns stsFailed.
Otherwise, c1sFontListBox fills out pArgs->pString with the font name (obtained by using msgIMGetName and the IM_HANDLE pArgs->data) and returns stsOK.

Return Value
stsFailed pArgs->position >= number of fonts

msgIMInstalled
A new item was installed.
Takes P_IM_NOTIFY, returns STATUS. Category: observer notification.

Comments
c1sFontListBox responds by resynchronizing its entries with respect to the list of installed fonts, as is done in msgNew.

msgIMDeinstalled
An item has been deinstalled.
Takes P_IM_DEINSTALL_NOTIFY, returns STATUS. Category: observer notification.

Comments
c1sFontListBox responds by resynchronizing its entries with respect to the list of installed fonts, as is done in msgNew.
This file contains the API definition for clsFrame.

clsFrame inherits from clsShadow.

Frames support a single client window, surrounded by a host of optional "decorations" -- title bar, menu bar, close box, tab bar, command bar, etc.

```c
#ifndef FRAME_INCLUDED
#define FRAME_INCLUDED

#ifndef SHADOW_INCLUDED
#include <shadow.h>
#endif
#endif
```

Common #defines and typedefs

```c
typedef OBJECT FRAME;
typedef struct FRAME_STYLE {
    U16 titleBar : 1, // show/don't show decoration
    menuBar : 1, // "
    closeBox : 1, // "
    cmdBar : 1, // "
    tabBar : 1, // "
    pageNum : 1, // "
    zoomable : 1, // true => zoom is allowed
    clipBoard : 1, // true => look like a clip board
    maskTitleLine : 1, // mask out the closeBox, titleBar, pageNum
    maskMenuLine : 1, // mask out the menuBar
    maskAll : 1, // mask out title, menu and cmd lines
    maskCmdLine : 1, // mask out the cmdBar
    useAltVisuals : 1, // use alternate border visuals
    spare1 : 3; // unused (reserved)
    spare2 : 16; // unused (reserved)
} FRAME_STYLE, *P_FRAME_STYLE;
```

Default FRAME_STYLE:

```c
titleBar = true
menuBar = false
closeBox = true
cmdBar = false
tabBar = false
pageNum = false
zoomable = true
clipBoard = false
maskTitleLine = false
maskMenuLine = false
maskAll = false
useAltVisuals = false
```

for msgFrameZoomOK, msgFrameZoomed

```c
typedef struct FRAME_ZOOM {
    FRAME frame; // in: Frame to zoom.
    BOOLEAN up; // in: True=zoom up, False=zoom down
    WIN toWin; // out: Window to zoom to
    U32 spare; // unused (reserved)
} FRAME_ZOOM, *P_FRAME_ZOOM;
```
msgNew

Creates a frame window. Passes back the resulting FRAME_METRICS in pArgs->frame.

Takes P_FRAME_NEW, returns STATUS. Category: class message.

Arguments

typedef struct FRAME_NEW_ONLY {
    FRAME_STYLE style;
    WIN clientWin;
    WIN titleBar;
    WIN menuBar;
    WIN closeBox;
    WIN cmdBar;
    P_CHAR pTitle;  // in only for msgNew
    OBJECT client;
    WIN tabBar;
    WIN pageNum;
    U32 spare1;  // page number
    U32 spare2;  // unused (reserved)
} FRAME_NEW_ONLY, *P_FRAME_NEW_ONLY,
FRAME_METRICS, *P_FRAME_METRICS;

#define frameNewFields \\
    shadowNewFields  \\
    FRAME_NEW_ONLY frame;

typedef struct FRAME_NEW {
    frameNewFields
} FRAME_NEW, *P_FRAME_NEW;

Comments

clsFrame creates an instance of clsFrameBorder as the frame’s border window to be the parent of all of
the frame decorations (except the tabBar, which is a direct child of the frame). The border window is
inserted as a child of the frame.

If pArgs->frame.style.clipBoard is true, the frame is made opaque and many of the border.style values
are changed to produce a clipboard style look.

For each of the decoration visibility style bits (e.g. style.titleBar), the following is done:

If the style value is true, and the corresponding decoration window (e.g..titleBar) is not objNull, the
window provided is inserted aschild of the frame border window.

If the style value is true and no window is provided (e.g. titleBar objNull), a default instance of the
decoration is created (e.g. msgNew clsTitleBar) and inserted as a child of the frame border window.

If the style value is false, the provided decoration window is remembered when the style value is set
to true.

If style.menuBar is true, the border style of the menuBar is altered to have a bottom edge with thickness
bsThicknessDouble and borderInk bsInkGray66.

If style.titleBar is true, the border style of the titleBar is altered to have a bottom edge with thickness
bsThicknessDouble (if style.menuBar is false) or bsThicknessSingle (if style.menuBar is true) and
borderInk bsInkGray66.

If style.closeBox is true, the border style of the closeBox is altered to match that of the titleBar.

If style.cmdBar is true and style.clipBoard is false, the border style of the cmdBar is altered to have a top
draw with thickness bsThicknessDouble and borderInk bsInkGray33.

If style.maskTitleLine is true, style.closeBox, style.titleBar and style.pageNum are all treated as though
they are false.
If style.maskMenuLine is true, style.menuBar is treated as though it is false.

If style.maskCmdLine is true, style.cmdBar is treated as though it is false.

If style.maskAll is true, style.maskTideLine, style.maskMenuLine, and style.maskCmdLine are all is treated as though they are true.

**msgNewDefaults**

Initializes the FRAME_NEW structure to default values.

Takes P_FRAME_NEW, returns STATUS. Category: class message.

```c
typedef struct FRAME_NEW {
    frameNewFields
} FRAME_NEW, *P_FRAME_NEW;
```

Zeroes out pArgs->frame and sets

- pArgs->win.flags.style &= ~wsParentClip;
- pArgs->win.flags.style |= wsClipChildren | wsClipSiblings;
- pArgs->embeddedWin.style.selection = ewSelect;
- pArgs->frame.style.titleBar = true;
- pArgs->frame.style.closeBox = true;
- pArgs->frame.style.zoomable = true;

**msgSave**

Causes an object to file itself in an object file.

Takes P_OBJ_SAVE, returns STATUS.

Comments

If the client of the frame is OSThisAPP(), this is remembered and reinstated in msgRestore. In any case, the client is not saved.

Each of the frame decorations, including the clientWin, with WIN_METRICS.flags.style.wsSendFile on is filed, even if the corresponding visibility style bit (e.g. style.titleBar) is false.

**msgRestore**

Creates and restores an object from an object file.

Takes P_OBJ_RESTORE, returns STATUS.

Comments

clsFrame restores the instance from the file. If the client of the frame was OSThisApp() when filed, the client is set to OSThisApp(), otherwise objNull.

Each of the filed decoration windows and the clientWin are restored. If the frame was zoomed when filed, the frame is unzoomed as in msgFrameZoom(false).

For each of the following, if the corresponding child windows were not filed (i.e. wsSendFile was not on), and the visibility style is on, default instances will not be created and the visibility style will be set to false: menuBar, cmdBar, and tabBar. For example, if the frame was filed with style.menuBar true and the menuBar did not have wsSendFile on, the restored frame will have style.menuBar false, and the menuBar in FRAME_METRICS set to objNull.
**msgFree**

Sent as the last of three msgs to destroy an object.

Takes OBJ_KEY, returns STATUS.

Comments

All children of the frame border window are destroyed. Decoration windows with visibility style bits off are also destroyed.

**msgFrameGetMetrics**

Passes back the metrics.

Takes P_FRAME_METRICS, returns STATUS.

#define msgFrameGetMetrics MakeMsg(clsFrame, 1)

**msgFrameSetMetrics**

Sets the metrics.

Takes P_FRAME_METRICS, returns STATUS.

#define msgFrameSetMetrics MakeMsg(clsFrame, 2)

Comments

clsFrame replaces existing decoration windows with those provided. For example, if pArgs->titleBar specifies a new titleBar, the existing titleBar is extracted from the window tree and the new titleBar inserted as a child of the frame border window.

Note that the old decoration windows are not destroyed and are no longer referenced by the frame (the client is free to destroy them at this point).

Frame style values are changed as in msgFrameSetStyle.

**msgFrameGetStyle**

Passes back the current style values.

Takes P_FRAME_STYLE, returns STATUS.

#define msgFrameGetStyle MakeMsg(clsFrame, 22)

typedef struct FRAME_STYLE {
  U16 titleBar : 1, // show/don’t show decoration
  menuBar : 1, // ”
  closeBox : 1, // ”
  cmdBar : 1, // ”
  tabBar : 1, // ”
  pageNum : 1, // ”
  zoomable : 1, // true => zoom is allowed
  clipboard : 1, // true => look like a clip board
  maskTitleLine : 1, // mask out the closeBox, titleBar, pageNum
  maskMenuLine : 1, // mask out the menuBar
  maskAll : 1, // mask out title, menu and cmd lines
  maskCmdLine : 1, // mask out the cmdBar
  useAltVisuals : 1, // use alternate border visuals
  spare1 : 3; // unused (reserved)
  U16 spare2 : 16; // unused (reserved)
} FRAME_STYLE, *P_FRAME_STYLE;
**msgFrameSetStyle**

Sets the style.

Takes `P_FRAME_STYLE`, returns `STATUS`.

```c
#define msgFrameSetStyle MakeMsg(clsFrame, 23)
```

```c
typedef struct FRAME_STYLE {
    U16 titleBar  : 1, // show/don't show decoration
    menuBar     : 1, //
    closeBox    : 1, //
    cmdBar      : 1, //
    tabBar      : 1, //
    pageNum     : 1, //
    zoomable    : 1, // true => zoom is allowed
    clipBoard   : 1, // true => look like a clip board
    maskTitleLine : 1, // mask out the closeBox, titleBar, pageNum
    maskMenuLine : 1, // mask out the menuBar
    maskAll     : 1, // mask out title, menu and cmd lines
    maskCmdLine : 1, // mask out the cmdBar
    useAltVisuals : 1, // use alternate border visuals
    spare1      : 3; // unused (reserved)
    U16 spare2   : 16; // unused (reserved)
} FRAME_STYLE, *p FRAME_STYLE;
```

**Comments**

The new decoration visibility style bits (e.g. `style.titleBar`) are treated as in `msgNew`. Setting a visibility bit to false results in extracting the corresponding decoration window (e.g. `metrics.titleBar`) from the frame border window. Note that the extracted decoration window is not destroyed; but remembered for later use when the visibility bit is set to true.

If `style.useAltVisuals` is changed from false to true, the alternate frame border visuals are applied to the frame's border style.

If `style.useAltVisuals` is changed from true to false, the normal frame border visuals are applied to the frame's border style.

Note that changing `style.clipBoard` is not implemented.

---

**msgFrameGetClientWin**

Passes back `metrics.clientWin`.

Takes `P_WIN`, returns `STATUS`.

```c
#define msgFrameGetClientWin MakeMsg(clsFrame, 24)
```

---

**msgFrameSetClientWin**

Sets `metrics.clientWin`.

Takes `WIN`, returns `STATUS`.

```c
#define msgFrameSetClientWin MakeMsg(clsFrame, 25)
```

**Comments**

The old `clientWin`, if any, is not destroyed and is no longer referenced by the frame.

---

**msgFrameGetMenuBar**

Passes back `metrics.menuBar`.

Takes `P_WIN`, returns `STATUS`.

```c
#define msgFrameGetMenuBar MakeMsg(clsFrame, 26)
```
msgFrameSetMenuBar
Sets metrics.menuBar; also sets style.menuBar to true if pArgs is not objNull, else false.
Takes WIN, returns STATUS.
#define msgFrameSetMenuBar MakeMsg(clsFrame, 27)
Comments
The menuBar is changed as in msgFrameSetMetrics.

msgFrameDestroyMenuBar
Sets style.menuBar to false and destroys the existing menu bar, if any.
Takes VOID, returns STATUS.
#define msgFrameDestroyMenuBar MakeMsg(clsFrame, 28)

msgFrameSetTitle
Sets the string in the metrics.titleBar.
Takes P_CHAR, returns STATUS.
#define msgFrameSetTitle MakeMsg(clsFrame, 3)
Comments
This results in msgLabelSetString to metrics.titleBar.

msgFrameGetClient
Passes back metrics.client.
Takes P_OBJECT, returns STATUS.
#define msgFrameGetClient MakeMsg(clsFrame, 4)

msgFrameSetClient
Sets metrics.client.
Takes OBJECT, returns STATUS.
#define msgFrameSetClient MakeMsg(clsFrame, 5)

msgFrameGetAltVisuals
Passes back the alternate border visuals.
Takes P_BORDER_STYLE, returns STATUS.
#define msgFrameGetAltVisuals MakeMsg(clsFrame, 29)

msgFrameSetAltVisuals
Sets the alternate border visuals.
Takes P_BORDER_STYLE, returns STATUS.
#define msgFrameSetAltVisuals MakeMsg(clsFrame, 30)
Comments
If style.useAltVisuals is true, the new alternate visuals are applied to the frame's border style.
**msgFrameGetNormalVisuals**
Passes back the normal border visuals.
Takes P_BORDER_STYLE, returns STATUS.

\[
\#define \textit{msgFrameGetNormalVisuals} \text{MakeMsg}(clsFrame, 31)
\]

Comments
This is equivalent to \textit{msgBorderGetStyle} if style.useAltVisuals is false.

**msgFrameSetNormalVisuals**
Sets the normal border visuals.
Takes P_BORDER_STYLE, returns STATUS.

\[
\#define \textit{msgFrameSetNormalVisuals} \text{MakeMsg}(clsFrame, 32)
\]

Comments
If style.useAltVisuals is false, the new normal visuals are applied to the frame's border style.

**msgFrameShowSelected**
Makes the frame look selected or not.
Takes BOOLEAN, returns STATUS.

\[
\#define \textit{msgFrameShowSelected} \text{MakeMsg}(clsFrame, 17)
\]

**msgFrameMoveEnable**
Enables or disables UI for moving.
Takes BOOLEAN, returns STATUS.

\[
\#define \textit{msgFrameMoveEnable} \text{MakeMsg}(clsFrame, 19)
\]

Comments
\(\text{clsFrame}\) alters the border.style.drag of the metrics.titleBar to be bsDragHoldDown if pArgs is true, bsDragNone otherwise.

**msgFrameResizeEnable**
Enables or disables UI for resizing.
Takes BOOLEAN, returns STATUS.

\[
\#define \textit{msgFrameResizeEnable} \text{MakeMsg}(clsFrame, 20)
\]

Comments
\(\text{clsFrame}\) alters the border.style.resize of self to be bsResizeCorner if pArgs is true, bsResizeNone otherwise.

**msgFrameIsZoomed**
Passes back true if the frame is currently zoomed.
Takes P_BOOLEAN, returns STATUS.

\[
\#define \textit{msgFrameIsZoomed} \text{MakeMsg}(clsFrame, 21)
\]

**msgFrameDelete**
Asks the frame's client to delete the frame.
Takes nothing, returns STATUS.

\[
\#define \textit{msgFrameDelete} \text{MakeMsg}(clsFrame, 7)
\]

Comments
\(\text{clsFrame}\) forwards this message to the client with self as the pArgs.
msgFrameClose
Asks the frame's client to close the frame.
Takes nothing, returns STATUS.
#define msgFrameClose MakeMsg(clsFrame, 8)
clsFrame forwards this message to the client with self as the pArgs.

msgFrameFloat
Asks the frame's client to float the frame.
Takes VOID, returns STATUS.
#define msgFrameFloat MakeMsg(clsFrame, 9)
clsFrame forwards this message to the client with self as the pArgs.

msgFrameZoom
Zooms the frame up or down.
Takes BOOLEAN, returns STATUS.
#define msgFrameZoom MakeMsg(clsFrame, 6)
If style.zoomable is false, nothing is done and stsOK is returned.
Otherwise, msgFrameZoomOK is sent to the client with the following FRAME_ZOOM parameters:

frame   = self;
up       = pArgs;
toWin    = objNull;

If the client returns stsRequestDenied or does not set the FRAME_ZOOM.toWin, the client's status is returned.
If the frame is already zoomed as pArgs requests, nothing is done and stsOK is returned.
If pArgs is true and style.clipBoard is false, the frame is zoomed up as follows:
• The frame is made opaque by turning off wsTransparent in WIN_METRICS.flags.style and turning off inputTransparent in WIN_METRICS.flags.input.
• The border edges, shadow, margin and resize handles on the frame are all turned off.
• The current frame window bounds and parent are remembered for restoration in unzoom.
• The frame is extracted from its current parent and inserted as a child of the FRAME_ZOOM.toWin with a window bounds computed to zoom the inner rect of the frame into the FRAME_ZOOM.toWin. The inner rect is computed using msgBorderGetOuterOffsets on the frame.

If pArgs is false and style.clipBoard is false, the frame is zoomed down as follows:
• The frame is made transparent by turning on wsTransparent in WIN_METRICS.flags.style and turning on inputTransparent in WIN_METRICS.flags.input.
• The border edges, shadow, margin and resize handles on the frame are all restored to their values before the zoom.
• The frame is extracted from its current parent and inserted in its original parent with its original window bounds.

After the frame is zoomed/unzoomed it is layed out via msgWinLayout to self.
clsFrame then sends the following notifications of the zoom/unzoom:
• self-send msgFrameZoomed with the FRAME_ZOOM as pArgs.
• msgFrameZoomed to its client with the FRAME_ZOOM as pArgs.
• self-sends msgNotifyObservers with the following OBJ_NOTIFY_OBSERVERS parameters:

```c
msg = msgFrameZoomed;
pArgs = address of FRAME_ZOOM used to zoom/unzoom;
lenSend = sizeof(FRAME_ZOOM);
```

### msgFrameSelect
Selects the frame.
Takes VOID, returns STATUS.

```c
#define msgFrameSelect MakeMsg(clsFrame, 18)
msgFrameSelectOK(self) is sent to the client.
```

### msgFrameZoomOK
Sent to the client when msgFrameZoom is received.
Takes P_FRAME_ZOOM, returns STATUS. Category: client notification.

```c
#define msgFrameZoomOK MakeMsg(clsFrame, 11)
```

```c
typedef struct FRAME_ZOOM {
    FRAME frame; // in: Frame to zoom.
    BOOLEAN up; // in: True=zoom up, False=zoom down
    WIN toWin; // out: Window to zoom to
    U32 spare; // unused (reserved)
} FRAME_ZOOM, *P_FRAME_ZOOM;
```

### msgFrameSelectOK
Sent to the client when msgFrameSelect is received.
Takes FRAME, returns STATUS. Category: client notification.

```c
#define msgFrameSelectOK MakeMsg(clsFrame, 16)
msgFrameSelectOK(self) is sent to the client.
```

### msgFrameZoomed
Sent to client and observers after frame is zoomed.
Takes P_FRAME_ZOOM, returns STATUS. Category: client & observer notification.

```c
#define msgFrameZoomed MakeMsg(clsFrame, 12)
```
**msgFrameClosed**

Sent to client and observers after frame is closed. **pArgs** is the frame.

Takes WIN, returns STATUS. Category: client & observer notification.

```c
#define msgFrameClosed MakeMsg(clsFrame, 13)
```

**Comments**

Note: not implemented.

---

**msgFrameFloated**

Sent to client and observers after frame is floated.

Takes VOID, returns STATUS. Category: client & observer notification.

```c
#define msgFrameFloated MakeMsg(clsFrame, 14)
```

**Comments**

Note: not implemented.

---

**msgFrameTopped**

Sent to client and observers after frame is brought to top.

Takes VOID, returns STATUS. Category: client & observer notification.

```c
#define msgFrameTopped MakeMsg(clsFrame, 15)
```

**Comments**

Note: not implemented.

---

### Messages from Other Classes

**msgGWinForwardedGesture:**

Called to process the gesture.

Takes P_GWIN_GESTURE, returns STATUS.

**Commands**

- clsFrame maps certain gestures forwarded from the frame’s titleBar into self-sent messages. Other gestures are forwarded to the frame’s client.
- If the **pArgs->uid** is not metrics.titleBar or a direct child of metrics.titleBar, **msgGWinForwardedGesture(pArgs)** will be sent to the frame’s client. clsFrame will return the client’s return status from this message.

The value of **pArgs->msg** is processed as follows:

- If xgsFlickUp/Down and the system preference with tag tagPrDocZooming is prDocZoomingOn, **msgFrameZoom(true/false)** is self-sent.
- If xgsCross, **msgFrameDelete(pNull)** is self-sent.
- If xgsPlus, **msgFrameSelect(pNull)** is self-sent.
- If xgs2Tap, **msgFrameFloat(pNull)** is self-sent.
- If xgs3Tap, the frame’s WIN_METRICS.flags.style.wsMaskWrapWidth/Height flags are cleared and **msgWinLayout(WIN_METRICS.options=wsLayoutDefault)** is self-sent. This results in a re-layout to the frame’s desired size.
• If `xgsTrplFlickUp` and the DEBUG version of tk.dll is installed, `msgWinDumpTree` is self-sent with `pArgs` of self or `theRootWindow` if the 'l' debug flag has value 1. Note that `msgWinDumpTree` requires the debug version of win.dll to be installed. This is useful for debugging window layout problems.

• All other gestures result in `msgGWinForwardedGesture(pArgs)` to the frame's client.

---

**msgTrackProvideMetrics**

Sent to a tracker client before tracker is created.


Comments

If `pArgs->minWH` and `pArgs->maxWH` allow the width to change, `pArgs->minWH.w` is set to a small value to prevent the frame from being resized to zero.

If `pArgs->minWH` and `pArgs->maxWH` allow the height to change, `pArgs->minWH.h` is set to prevent the frame from being resized smaller than the sum of the metrics.titleBar and metrics.menuBar heights.

The value of `pArgs->style.draw` is altered to present the proper visual given the frame's style.tabBar and style.cmdBar.

`msgTrackProvideMetrics(pArgs)` is sent to the frame's client.

---

**msgWinSetFlags**

Sets the window flags.

Takes `P_WIN_METRICS`, returns `STATUS`.

Comments

`clsFrame` alters the metric.clientWin's window flags to match the wsShrinkWrapWidth/Height flags of the frame.

---

**msgCstmLayoutGetChildSpec**

Passes back the current spec for the specified child.


Comments

`clsFrame` responds by providing the custom layout constraints for metrics.tabBar, metrics.cmdBar, and the frame's border window.

Note that the decoration windows and the metrics.clientWin are actually children of the frame's border window, which is an instance of `clsFrameBorder`. `clsFrameBorder` responds to `msgCstLayoutGetChildSpec` by providing the custom layout constraints for its children (e.g. `titleBar` at the top, `menuBar` below `titleBar`, etc.).

---

**msgWinSend**

Sends a message up a window ancestry chain.

Takes `WIN_SEND`, returns `STATUS`.

Comments

If `pArgs->msg` is `msgBorderProvideDeltaWin` and the frame is zoomed, `clsFrame` returns `stsOK`. This prevents a zoomed frame from being resized.
This file contains the API definition for clsGrabBox.

clsGrabBox inherits from clsObject.

Provides popup grab handles; uses clsTrack internally.

GrabBoxes are used primarily by clsBorder to display resize handles, although other uses are possible.

```c
 ifndef GRABBOX_INCLUDED
 define GRABBOX_INCLUDED

 include <clsmgr.h>
 include <sysgraf.h>

 ifndef CLSMGR_INCLUDED
 endif
 ifndef SYSGRAF_INCLUDED
 endif
```

### Common #defines and typedefs

```c
 typedef OBJECT GRAB_BOX;
```

### Type styles

```c
 define gbTypeResize 0  // resize
    1  // unused (reserved)
    ...  // unused (reserved)
```

### Locations styles

```c
 define gbLocULCorner 0  // upper-left corner
 define gbLocURCorner 1  // upper-right corner
 define gbLocLRCorner 2  // lower-right corner
 define gbLocLLCorner 3  // lower-left corner
 define gbLocLeftEdge 4  // left edge
 define gbLocRightEdge 5  // right edge
 define gbLocBottomEdge 6  // bottom edge
 define gbLocTopEdge 7  // top edge
 define gbLocNone 8  // no edge
```

```c
typedef struct GRAB_BOX_STYLE {
   U16 type,  // type of grab box
          2,  // type of grab box
   loc : 4,  // location of grab box
   autoDestroy : 1,  // destroy self on take down
   autoTakeDown : 1,  // take down if pen is outside grab box
   spare : 8;  // unused (reserved)
} GRAB_BOX_STYLE, *P_GRAB_BOX_STYLE;
```

Default GRAB_BOX_STYLE:

```c
 type = gbTypeResize
 loc = gbLocULCorner
 autoDestroy = true
 autoTakeDown = true
```
typedef struct GRAB_BOX_INFO {
WIN win; // window over which grab box will be drawn
U16 thickness; // thickness of visible grab area, in twips
U16 length; // length of visible grab area, in twips
RECT32 outerMargin; // thickness of invisible grab area, in twips
BOOLEAN includeOuter; // true to include invisible area
BOOLEAN penIsDown; // true if pen is down (for msgGrabBoxShow)
XY32 downXY; // xy on pen down in win space (for msgGrabBoxShow)
U16 visualInset; // amount to inset length for visual, in twips
U16 cornerRadius; // radius for round corners (zero for square), in twips
U32 spare1; // unused (reserved)
U32 spare2; // unused (reserved)
} GRAB_BOX_INFO, *P_GRAB_BOX_INFO;

msgNew
Creates a grab box object.
Takes P_GRAB_BOX_NEW, returns STATUS. Category: class message.

typedef struct GRAB_BOX_NEW_ONLY {
GRAB_BOX_STYLE style; // overall style
WIN client; // window to grab
XY32 xy; // unused
WIN xyWin; // unused
U8 margin; // unused
U32 spare; // unused (reserved)
} GRAB_BOX_NEW_ONLY, *P_GRAB_BOX_NEW_ONLY,
GRAB_BOX_METRICS, *P_GRAB_BOX_METRICS;
#define grabBoxNewFields \
objectNewFields \
GRAB_BOX_NEW_ONLY grabBox;
typedef struct {
grabBoxNewFields
} GRAB_BOX_NEW, *P_GRAB_BOX_NEW;

msgNewDefaults
Initializes the GRAB_BOX_NEW structure to default values.
Takes P_GRAB_BOX_NEW, returns STATUS. Category: class message.

typedef struct {
grabBoxNewFields
} GRAB_BOX_NEW, *P_GRAB_BOX_NEW;

Comments
Zeroes out pArgs->grabBox and sets
pArgs->grabBox.style.autoDestroy = true;
pArgs->grabBox.style.autoTakeDown = true;

msgGrabBoxGetStyle
Passes back current style values.
Takes P_GRAB_BOX_STYLE, returns STATUS.

#define msgGrabBoxGetStyle MakeMsg(clsGrabBox, 1)

typedef struct GRAB_BOX_STYLE {
U16 type : 2, // type of grab box
loc : 4, // location of grab box
autoDestroy : 1, // destroy self on take down
autoTakeDown : 1, // take down if pen is outside grab box
spare : 8; // unused (reserved)
} GRAB_BOX_STYLE, *P_GRAB_BOX_STYLE;
msgGrabBoxSetStyle
Sets style values.
Takes P_GRAB_BOX_STYLE, returns STATUS.

```c
#define msgGrabBoxSetStyle MakeMsg(clsGrabBox, 2)
```

**Message**

typedef struct GRAB_BOX_STYLE {
    U16 type : 2, // type of grab box
    loc : 4, // location of grab box
    autoDestroy : 1, // destroy self on take down
    autoTakeDown : 1, // take down if pen is outside grab box
    spare : 8; // unused (reserved)
} GRAB_BOX_STYLE, *P_GRAB_BOX_STYLE;

**Comments**
Note that changing style.loc or style.type while the grab box is being shown is not supported.

msgGrabBoxGetMetrics
Passes back current metrics.
Takes P_GRAB_BOX_METRICS, returns STATUS.

```c
#define msgGrabBoxGetMetrics MakeMsg(clsGrabBox, 3)
```

msgGrabBoxSetMetrics
Sets metrics.
Takes P_GRAB_BOX_METRICS, returns STATUS.

```c
#define msgGrabBoxSetMetrics MakeMsg(clsGrabBox, 4)
```

**Comments**
Sets the style as in msgGrabBoxSetStyle.

msgGrabBoxShow
Puts up or takes down the grab box.
Takes P_GRAB_BOX_INFO, returns STATUS.

```c
#define msgGrabBoxShow MakeMsg(clsGrabBox, 5)
```

**Message**

typedef struct GRAB_BOX_INFO {
    WIN win; // window over which grab box will be drawn
    U16 thickness; // thickness of visible grab area, in twips
    U16 length; // length of visible grab area, in twips
    RECT32 outerMargin; // thickness of invisible grab area, in twips
    BOOLEAN includeOuter; // true to include invisible area
    BOOLEAN penIsDown; // true if pen is down (for msgGrabBoxShow)
    XY32 downXY; // xy on pen down in win space (for msgGrabBoxShow)
    U16 visualInset; // amount to inset length for visual, in twips
    U16 cornerRadius; // radius for round corners (zero for square), in twips
    U32 spare1; // unused (reserved)
    U32 spare2; // unused (reserved)
} GRAB_BOX_INFO, *P_GRAB_BOX_INFO;

**Comments**
If pArgs is not pNull, clsGrabBox will grab input using InputSetGrab() and paint the grab box. If style.autoTakeDown is true, the grab box will be taken down when the pen leaves proximity or moves out of the grab box with the pen up.

If pArgs is pNull, clsGrabBox will take down the grab box and self-send msgDestroy(pNull) if style.autoDestroy is true.
The area on which the grab box was drawn will be damaged with `msgWinDirtyRect` when the grab box is taken down.

The grab box is drawn in the rectangle computed by `GrabBoxLocToRect()`.

### Public Functions

#### GrabBoxIntersect
Determines where `pRect` is in `win`. Returns a grab box location, e.g. `gbLocLRCorner`.

Returns U16.

**Function Prototype**

```c
U16 EXPORTED GrabBoxIntersect(  
    P_GRAB_BOX_INFO pInfo, // info about grab box locations  
    P_RECT32    pRect     // Rect to intersect  
);
```

**Comments**

- `pRect->origin` is commonly the coordinate of an event in `pInfo->win`'s space, in device units.
- `pInfo->thickness` is the thickness (in twips) of the visible grab-sensitive area within `pInfo->win`.
- `pInfo->outerMargin` is the thickness (in twips) of the invisible grab-sensitive area within `pInfo->win`.
- `pInfo->outerMargin.{origin.x, size.w}` are margins for the left and right, respectively.
- `pInfo->outerMargin.{origin.y, size.h}` are margins for the bottom and top, respectively.
- `pInfo->length` is the length of each grab-sensitive area, in twips.
- If `pInfo->includeOuter` is true, the outer margin area is included in the rect for each grab box.

This is used by `clsBorder` to place a grab box over the resize handles.

#### GrabBoxLocToRect
Computes the rectangle of the grabBox at the given location.

Returns void.

**Function Prototype**

```c
void EXPORTED GrabBoxLocToRect(  
    P_GRAB_BOX_INFO pInfo, // info about grab box locations  
    U16        location,  // e.g. gbLocBottom  
    P_RECT32   pRect      // Rect to locate  
);
```

**Comments**

- `pInfo` is as described in `GrabBoxIntersect()`.

The corresponding rect for location is returned in `pRect`, in device units.

#### GrabBoxPaint
Paints the grab box at the specified location.

Returns `STATUS`.

**Function Prototype**

```c
STATUS EXPORTED GrabBoxPaint(  
    P_GRAB_BOX_INFO pInfo,  
    U16        loc,  
    SYSDC      dc,  
    P_RECT32   pRect,  
    BOOLEAN    clearOuter,  
    BOOLEAN    on  
);
```
Comments

 pInfo is as described in GrabBoxIntersect().
 If dc is not objNull, it will be used for the painting.
 If pRect is pNull, the corresponding rect for location will be used; otherwise pRect will be used.
 If clearOuter is true, all of pRect will be cleared before painting.
 If on is true, the grab box will be painted in black, otherwise gray66.
 This is used by clsBorder to paint the resize handles.

Messages from other classes

msgInputEvent
Notification of an input event.
Takes P_INPUT_EVENT, returns STATUS.

clsGrabBox will respond to input events that trigger resizing.
If pArgs->devCode is msgPenUp, msgPenOutProxUp, msgPenOutProxDown, or msgPenMoveUp and pArgs->xy is not in the rectangle of the grab box and style.autoTakeDown is true or msgPenDown has been received, the grab box is taken down as in msgGrabBoxShow(false).
If pArgs->devCode is msgPenDown the following is done:
msgTrackProvideMetrics is sent to metrics.client with the following METRICS parameters:
msgNewDefaults is sent to clsTrack to initialize a TRACK_METRICS struct and then:

```
style.track = tsTrackResize;
style.anchor = computed from self's style.loc;
win = parent of metrics.client;
client = self;
clientData = window to be resized;
initRect = bounds of metrics.client;
minWH = small rectangle;
maxWH = limited to stay within parent of metrics.client
tag = tagBorderResize;
```
If style.loc is gbLocLeftEdge or gbLocRightEdge, maxWH is altered to horizontal resize.
If style.loc is gbLocBottomEdge or gbLocTopEdge, maxWH is altered to vertical resize.
An instance of clsTrack is created and started via msgTrackStart.

msgTrackDone
Sent by a tracker when it's done.
Takes P_TRACK_METRICS, returns STATUS. Category: client notification.

clsGrabBox responds by resizing metrics.client to pArgs->rect.size.
If the width/height is changed, wsMaskWrapWidth/Height will be turned on in WIN_METRICS.flags.style for metrics.client.
The client window is resized by sending msgWinLayout with the following WIN_METRICS parameters:
options = 0;
bounds = pArgs->rect;
If style.autoDestroy is true, msgDestroy(pNull) is self-posted.
This file contains the API for clsIconChoice.

clsIconChoice inherits from clsChoice.

IconChoices are exclusive choices with icon buttons and boxed-style previewing/on feedback.

See the documentation for msgTkTableChildDefaults below.

```c
 ifndef ICHOICE_INCLUDED
 #define ICHOICE_INCLUDED
 #include <choice.h>
 #ifndef ICON_INCLUDED
 #include <icon.h>
 #endif
```

### Common #defines and typedefs

```c
typedef OBJECT ICON_CHOICE;
typedef struct ICON_CHOICE_STYLE {
    U16 spare : 16; // unused (reserved)
} ICON_CHOICE_STYLE, *P_ICON_CHOICE_STYLE;
```

**msgNew**

Creates an iconChoice (and its nested icon windows).

Takes P_ICON_CHOICE_NEW, returns STATUS. Category: class message.

```c
typedef struct ICON_CHOICE_NEW ONLY {
    ICON_CHOICE_STYLE style; // overall style
    ICON_CHOICE NEW iconNew; // storage for default child new struct
    U32 spare; // unused (reserved)
} ICON_CHOICE_NEW ONLY, *P_ICON_CHOICE_NEW ONLY;
#define iconChoiceNewFields \
    choiceNewFields \ 
    ICON_CHOICE NEW ONLY iconChoice;

typedef struct ICON_CHOICE_NEW {
    iconChoiceNewFields
    ICON_CHOICE NEW iconChoice;
} ICON_CHOICE_NEW, *P_ICON_CHOICE_NEW;
```

**msgNewDefaults**

Initializes the ICON_CHOICE_NEW structure to default values.

Takes P_ICON_CHOICE_NEW, returns STATUS. Category: class message.

```c
typedef struct ICON_CHOICE_NEW {
    iconChoiceNewFields
} ICON_CHOICE_NEW, *P_ICON_CHOICE_NEW;
```

Sets up pArgs->tkTable.pButtonNew to create instances of clsIcon with boxed-style previewing/on feedback by default as follows:

```c
pButtonNew->button.feedback = bsFeedbackBox;
Zeroes out pNew.iconChoice.
```
**msgTkTableChildDefaults**

Sets the defaults in P_ARGS for a common child.

Takes P_UNKNOWN, returns STATUS.

**Comments**

Here is how an iconChoice processes this message:

```c
if <pArgs->object.class inherits from clsButton> {
    pArgs->button.style.feedback = bsFeedbackBox;
}
```
ICON.H

This file contains the API definition for clsIcon.

clsIcon inherits from clsMenuButton.

Icons support drawing a picture as well as a label string. Several picture types are supported, including bitmap.

#ifndef ICON_INCLUDED
#define ICON_INCLUDED

#include <mbutton.h>

#ifndef MBUTTON_INCLUDED
#endif
#endif

Common #defines and typedefs

typedef OBJECT ICON;

Picture Styles

#define isPictureBitmap 0 // picture is a bitmap
#define isPictureNone 1 // no picture
#define isPicturePixelmap 2 // picture is a pixelmap
#define isPicturePixelmap 2 // picture is a pixelmap

Aspect Ratio Styles

#define isAspectWidthFromHeight 0 // compute width from height & sample size
#define isAspectHeightFromWidth 1 // compute height from width & sample size
#define isAspectAsIs 2 // use the width and height as-is
#define isAspectAsIs 2 // use the width and height as-is

Common Layout Units Picture Sizes

#define iconSizeNormal 21 // standard size, both width and height
#define iconSizeSmall 10 // standard small size

typedef struct ICON_STYLE {
    U16 transparent : 2, // make the background transparent
    picture : 2, // type of picture
    freeBitmap : 1, // true => msgDestroy to bitmap after provide
    open : 1, // modify picture to look open
    sizeUnits : 6, // units for pictureSize, e.g. bsUnitsPoints
    sampleBias : 1, // true => alter pictureSize for quality
    aspect : 2, // aspect ratio rule (e.g. isAspectWidthFromHeight)
    spare1 : 1; // unused (reserved)
    U16 spare2 : 16; // unused (reserved)
} ICON_STYLE, *P_ICON_STYLE;
**Messages**

**msgNew**  
Creates an icon window.  
Takes P_ICON_NEW, returns STATUS. Category: class message.

```c
typedef struct ICON_NEW_ONLY {
    ICON_STYLE style; // overall style
    SIZE16 pictureSize; // picture size, in device units
    U32 spare; // unused (reserved)
} ICON_NEWONLY, *P_ICON_NEWONLY;

#define iconNewFields
    menuButtonNewFields \n    ICON_NEWONLY icon;

typedef struct ICON_NEW {
    iconNewFields
} ICON_NEW, *P_ICON_NEW;
```

**Comments**  
If pArgs->icon.style.transparent is true, wsTransparent is turned on in pArgs->win.flags.style and bsInkExclusive will be or-ed into pArgs->border.style.backgroundlnk.

**msgNewDefaults**  
Initializes the ICON_NEW structure to default values.  
Takes P_ICON_NEW, returns STATUS. Category: class message.

```c
typedef struct ICON_NEW {
    iconNewFields
} ICON_NEW, *P_ICON_NEW;
```

**Comments**  
Zeroes out pArgs->icon and sets

```c
    pArgs->gWin.style.gestureEnable = true;

    pArgs->border.style.backgroundlnk = bsInkWhite | bsInkExclusive;
    pArgs->border.style.borderlnk = bsInkWhite | bsInkExclusive;

    pArgs->border.style.previewAlter = bsAlterBorders;
    pArgs->border.style.selectedAlter = bsAlterBorders;
    pArgs->border.style.edge = bsEdgeBottom;
    pArgs->border.style.shadow = bsShadowNone;

    pArgs->control.style.showDirty = true;

    pArgs->label.style.xAlignment = lsAlignCenter;

    pArgs->icon.style.freeBitmap = true;
    pArgs->icon.style.sampleBias = true;
    pArgs->icon.pictureSize.w = pArgs->icon.pictureSize.h = iconSizeNormal;
```

Default ICON_STYLE:

- transparent = false
- picture = isPictureBitmap
- freeBitmap = true
- open = false
- sizeUnits = bsUnitsLayout
- sampleBias = true
- aspect = isAspectWidthFromHeight
**msgIconGetStyle**

Passes back the current style values.

Takes P_ICON_STYLE, returns STATUS.

```c
#define msgIconGetStyle MakeMsg(clsIcon, 1)
```

```c
typedef struct ICON_STYLE {
    U16 transparent  : 2,  // make the background transparent
    picture           : 2,  // type of picture
    freeBitmap        : 1,  // true => msgDestroy to bitmap after provide
    open              : 1,  // modify picture to look open
    sizeUnits         : 6,  // units for pictureSize, e.g. bsUnitsPoints
    sampleBias        : 1,  // true => alter pictureSize for quality
    aspect            : 2,  // aspect ration rule (e.g. isAspectWidthFromHeight)
    spare1            : 1;  // unused (reserved)
    U16 spare2         : 16; // unused (reserved)
} ICON_STYLE, *P_ICON_STYLE;
```

**msgIconSetStyle**

Sets the style values.

Takes P_ICON_STYLE, returns STATUS.

```c
#define msgIconSetStyle MakeMsg(clsIcon, 2)
```

```c
typedef struct ICON_STYLE {
    U16 transparent  : 2,  // make the background transparent
    picture           : 2,  // type of picture
    freeBitmap        : 1,  // true => msgDestroy to bitmap after provide
    open              : 1,  // modify picture to look open
    sizeUnits         : 6,  // units for pictureSize, e.g. bsUnitsPoints
    sampleBias        : 1,  // true => alter pictureSize for quality
    aspect            : 2,  // aspect ration rule (e.g. isAspectWidthFromHeight)
    spare1            : 1;  // unused (reserved)
    U16 spare2         : 16; // unused (reserved)
} ICON_STYLE, *P_ICON_STYLE;
```

**msgIconGetPictureSize**

Passes back the picture size in style.sizeUnits.

Takes P_SIZE16, returns STATUS.

```c
#define msgIconGetPictureSize MakeMsg(clsIcon, 3)
```

**msgIconSetPictureSize**

Sets the picture size.

Takes P_SIZE16, returns STATUS.

```c
#define msgIconSetPictureSize MakeMsg(clsIcon, 4)
```

**Comments**

If style.open changes, the rect of the picture is dirtied by self-sending msgWinDirtyRect.

Note that changing style.transparent is not implemented.

The new picture size should be in style.sizeUnits (e.g. bsUnitsLayout). clsIcon will free the cached picture as in msgIconFreeCache.
msgIconGetActualPictureSize
Computes and passes back the actual picture size in device units.
Takes P_SIZE16, returns STATUS.
#define msgIconGetActualPictureSize MakeMsg(clsIcon, 10)

Comments
This is equivalent using msgIconGetPictureSize and converting to device units if style.sampleBias is
false or style.picture is not isPictureBitmap.
Otherwise, clsIcon will compute and pass back the actual picture size used based on the sample size of
the bitmap, the specified picture size and style.sizeUnits, style.aspect, and the device resolution of the
icon's window device.

msgIconFreeCache
Frees the cached picture, if any.
Takes pNull, returns STATUS.
#define msgIconFreeCache MakeMsg(clsIcon, 8)

Comments
If style.picture isPictureBitmap, is clsIcon will send msgIconProvideBitmap on the next
msgWinRepaint.

Note that clsIcon does not self-send msgWinDirtyRect here. You should send msgWinDirtyRect after
msgIconFreeCache if you want the icon to repaint before it is otherwise damaged.

msgIconGetRects
Passes back the bounds for the picture in pArgs[0] and the label in pArgs[1].
Takes P_RECT32, returns STATUS.
#define msgIconGetRects MakeMsg(clsIcon, 6)

Comments
Note that pArgs is an array of two RECT32 structs. Bounds are in device units, relative to the origin of
the icon.

msgIconProvideBitmap
Sent to control client when icon needs the picture bitmap.
Takes P_ICON_PROVIDE_BITMAP, returns STATUS. Category: self-sent and client notification.
#define msgIconProvideBitmap MakeMsg(clsIcon, 7)

typedef struct ICON_PROVIDE_BITMAP {
    WIN icon;          // in: icon asking for the bitmap
    TAG tag;           // in: window tag of icon
    OBJECT device;     // in: device on which bitmap will be rendered
    SIZE16 pictureSize; // in: size of picture, device units
    OBJECT bitmap;     // out: bitmap to render
    U32 spare1;        // unused (reserved)
    U32 spare2;        // unused (reserved)
} ICON_PROVIDE_BITMAP, *P_ICON_PROVIDE_BITMAP;

Comments
clsIcon will self-send this message when it needs the picture bitmap. Subclasses can catch this message
and provide the appropriate bitmap.

If clsIcon receives this message, the message will be forwarded on to the icon's control client.
After the subclass or client provides the bitmap, clsIcon will copy the bitmap to a cached data structure for use when painting. If style.freeBitmap is true, clsIcon will send msgDestroy to the bitmap after creating the cache.

**msgIconCopyPixels**

Causes the icon to copy pixels from pArgs->srcWin to a pixelmap.

Takes P_ICON_COPY_PIXELS, returns STATUS.

```
#define msgIconCopyPixels MakeMsg(clsIcon, 9)
```

**Arguments**

```
typedef struct ICON_COPY_PIXELS {
    WIN srcWin;  // in: source window
    XY32 srcXY;  // in: origin of area to copy, srcWin space
    U32 spare1;  // unused (reserved)
} ICON_COPY_PIXELS, *P_ICON_COPY_PIXELS;
```

**Comments**

If style.picture is not isPicturePixmap or pArgs->srcWin is objNull, clsIcon will return stsBadParam.

The area copied has size of pictureSize and origin pArgs->srcXY in pArgs->srcWin space. The pixelmap will be used during msgWinRepaint.

**msgIconSampleBias**

Computes the sample-biased size for a given picture size.

Takes P_ICON_SAMPLE_BIAS, returns STATUS. Category: class message.

```
#define msgIconSampleBias MakeMsg(clsIcon, 11)
```

**Arguments**

```
typedef struct ICON_SAMPLE_BIAS {
    WIN win;  // in: device window
    U32 tolerance;  // in: snap-to tolerance, in layout units
    SIZE32 sampleSize;  // in: sample size, in device units
    SIZE32 size;  // in/out: picture size, in device units
    UI6 sizeUnits : 6;  // in: units for size
    aspect : 2;  // in: aspect ratio style
    spare1 : 8;  // unused (reserved)
    U32 spare2;  // unused (reserved)
} ICON_SAMPLE_BIAS, *P_ICON_SAMPLE_BIAS;
```

**Comments**

clsIcon will alter pArgs->size.w/h to be a multiple of pArgs->sampleSize.w/h. If the new pArgs->size.w/h is within pArgs->tolerance units from pArgs->sampleSize.w/h, the size is rounded up or down to the sample size.

pArgs->sampleSize should be in device units. pArgs->size should be in the units described by pArgs->sizeUnits (e.g. bsUnitsLayout). pArgs->tolerance should be in layout units. pArgs->win is any window on the related device.

If pArgs->aspect is is AspectWidthFromHeight, the width will be computed from the final height as size.w = size.h * (sampleSize.w / sampleSize.h);

If pArgs->aspect is is AspectHeightFromWidth, the height will be computed from the final width as size.h = size.w * (sampleSize.h / sampleSize.w);

This message can be sent to clsIcon or any instance of clsIcon.
Here is the current "size bias" code. In this fragment `sampleSize` is the sample's width or height, `size` is the proposed picture with or height, `tolerance` is the "snap-to" tolerance. All values are in device units. The computed value is the sample-biased picture width or height.

```c
if (size > sampleSize) {
    S32 mult;
    S32 lowerValue, lowerDelta;
    S32 upperValue, upperDelta;
    mult = size / sampleSize;
    lowerValue = mult * sampleSize;
    lowerDelta = size - lowerValue;
    upperValue = (mult + 1) * sampleSize;
    upperDelta = upperValue - size;
    if (lowerDelta < upperDelta) {
        value = lowerValue;
        delta = lowerDelta;
    } else {
        value = upperValue;
        delta = upperDelta;
    }
} else {  
    delta = sampleSize - size;
    value = sampleSize;
}
if (delta <= tolerance)
    size = value;
return size;
```

**Messages from other classes**

**msgWinSetTag**

Sets the window tag.

Takes `P_WIN_METRICS`, returns `STATUS`.

**Comments**

If `pArgs->tag` is the same as the current window tag, nothing is done and `stsOK` is returned.

If `style.picture` is `isPictureBitmap`, `clsIcon` will self-send `msgIconFreeCache` followed by `msgWinDirtyRect` to force a redraw of the icon picture.
ITABLE.H

This file contains the API for clsIconChoice.

clsIconTable inherits from clsToggleTable.

IconTables are non-exclusive toggle tables with icon buttons and boxed-style previewing/on feedback.

See the documentation for msgTkTableChildDefaults below.

#ifndef ITABLE_INCLUDED
#define ITABLE_INCLUDED

#include <tttable.h>

#ifndef TTABLE_INCLUDED
#define TTABLE_INCLUDED
#endif

#ifndef ICON_INCLUDED
#include <icon.h>
#endif

Common #defines and typedefs

typedef OBJECT ICON_TABLE;

typedef struct ICON_TABLE_STYLE {
    U16 spare : 16;  // unused (reserved)
} ICON_TABLE_STYLE, *P_ICON_TABLE_STYLE;

msgNew

Creates an iconTable (and its nested icon windows).

Takes P_ICON_TABLE_NEW, returns STATUS. Category: class message.

Arguments

typedef struct ICON_TABLE_NEW_ONLY {
    ICON_TABLE_STYLE style;  // overall style
    ICON_NEW iconNew;        // storage for default child new struct
    U32 spare;                // unused (reserved)
} ICON_TABLE_NEW_ONLY, *P_ICON_TABLE_NEW_ONLY;

#define iconTableNewFields \
    toggleTableNewFields \
    ICON_TABLE_NEW_ONLY iconTable;

typedef struct ICON_TABLE_NEW {
    iconTableNewFields
    ICON_TABLE_NEW, *P_ICON_TABLE_NEW;
}

msgNewDefaults

Initializes the ICON_TABLE_NEW structure to default values.

Takes P_ICON_TABLE_NEW, returns STATUS. Category: class message.

Message Arguments

typedef struct ICON_TABLE_NEW {
    iconTableNewFields
} ICON_TABLE_NEW, *P_ICON_TABLE_NEW;
Sets up pArgs->tkTable.pButtonNew to create instances of clsIcon with boxed-style previewing/on feedback by default as follows:

```c
pButtonNew->button.style.feedback = bsFeedbackBox;
pButtonNew->button.style.contact = bsContactToggle;
```

Zeroes out pNew.iconTable.

---

**Messages from Other Classes**

**msgTkTableChildDefaults**

Sets the defaults in P_ARGS for a common child.

Takes P_UNKNOWN, returns STATUS.

Comments

Here is how an iconTable processes this message:

```c
if <pArgs->object.class inherits from clsButton> {
   pArgs->button.style.feedback = bsFeedbackBox;
}
```
This file contains the API definition for `clsIconToggle`.

`clsIconToggle` inherits from `clsIcon`.

Icon toggles are toggle buttons with pictures for on and off states. These can be used to display an on/off mode switch.

```c
#ifndef ITOGGLE_INCLUDED
#define ITOGGLE_INCLUDED

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

### Common #defines and typedefs

```c
typedef OBJECT ICON_TOGGLE;
typedef struct ICON_TOGGLE_STYLE {
    U16 spare : 16; // unused (reserved)
} ICON_TOGGLE_STYLE, *P_ICON_TOGGLE_STYLE;
```

Default off/on picture tags. These are the resids for bitmaps in the system resource file. The default bitmaps represent "ink mode" for off (a picture of a pencil) and "gesture mode" for on (a picture of a check mark).

```c
#define tagIconToggleOff MakeTag(clsIconToggle, 1)
#define tagIconToggleOn MakeTag(clsIconToggle, 2)
```

### Messages

**msgNew**

Creates an icon toggle window.

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

**Arguments**

```c
typedef struct ICON_TOGGLE_NEWONLY {
    ICON_TOGGLE_STYLE style; // overall style
    TAG offTag; // picture tag to use when off
    TAG onTag; // picture tag to use when on
    U32 spare1; // unused (reserved)
    U32 spare2; // unused (reserved)
} ICON_TOGGLE_NEWONLY, *P_ICON_TOGGLE_NEWONLY;
```

```c
#define iconToggleNewFields  
iconNewFields  |
ICON_TOGGLE_NEWONLY iconToggle;
```

**Comments**

The fields you commonly set are:

- `pArgs->iconToggle.offTag` picture tag to use when button is off
- `pArgs->iconToggle.onTag` picture tag to use when button is on
msgNewDefaults

Initializes the ICON_TOGGLE_NEW structure to default values.

Takes P_ICON_TOGGLE_NEW, returns STATUS. Category: class message.

typedef struct ICON_TOGGLE_NEW {
    iconToggleNewFields
} ICON_TOGGLE_NEW, *P_ICON_TOGGLE_NEW;

Zeroes out pArgs->iconToggle and sets:

pArgs->gWin.style.gestureEnable = false;
pArgs->button.style.feedback = bsFeedbackNone;
pArgs->button.style.contact = bsContactToggle;
pArgs->icon.pictureSize.w = 8
pArgs->icon.pictureSize.h = 8
pArgs->iconToggle.offTag = tagIconToggleOff;
pArgs->iconToggle.onTag = tagIconToggleOn;

Note that the default picture size is set to 8x8 layout units, which is the width and height of the system font.
The default off and on tags represent bitmaps stored in the system resource file. These are the bitmaps for "ink mode" (off) and "gesture mode" (on).

msgIconToggleGetStyle

Passes back the current style values.

Takes P_ICON_TOGGLE_STYLE, returns STATUS.

#define msgIconToggleGetStyle MakeMsg(clsIconToggle, 1)

typedef struct ICON_TOGGLE_STYLE {
    U16 spare : 16; // unused (reserved)
} ICON_TOGGLE_STYLE, *P_ICON_TOGGLE_STYLE;

msgIconToggleSetStyle

Sets the style values.

Takes P_ICON_TOGGLE_STYLE, returns STATUS.

#define msgIconToggleSetStyle MakeMsg(clsIconToggle, 2)

typedef struct ICON_TOGGLE_STYLE {
    U16 spare : 16; // unused (reserved)
} ICON_TOGGLE_STYLE, *P_ICON_TOGGLE_STYLE;

msgIconToggleGetOnTag

Passes back the onTag.

Takes P_TAG, returns STATUS.

#define msgIconToggleGetOnTag MakeMsg(clsIconToggle, 3)
**msgIconToggleGetOffTag**

Passes back the offTag.

Takes P_TAG, returns STATUS.

```c
#define msgIconToggleGetOffTag MakeMsg(clsIconToggle, 4)
```

**msgIconToggleSetOnTag**

Sets the onTag.

Takes TAG, returns STATUS.

```c
#define msgIconToggleSetOnTag MakeMsg(clsIconToggle, 5)
```

**Comments**

`clsIconToggle` will remember the onTag for use when the button is on. If the button is currently on, `msgIconFreeCache` will be self-sent to free the current picture bitmap and use the new one.

---

**msgIconToggleSetOffTag**

Sets the offTag.

Takes TAG, returns STATUS.

```c
#define msgIconToggleSetOffTag MakeMsg(clsIconToggle, 6)
```

**Comments**

`clsIconToggle` will remember the offTag for use when the button is off. If the button is currently off, `msgIconFreeCache` will be self-sent to free the current picture bitmap and use the new one.

---

### Messages from Other Classes

**msgButtonShowFeedback**

Shows the feedback for an on/off button if pArgs is true/false.

Takes BOOLEAN, returns STATUS. Category: self-sent.

**Comments**

`clsIconToggle` will free the old bitmap via `msgIconFreeCache` and cause the new one to be displayed by damaging the picture rectangle. The current feedback state will be remembered for use in `msgIconProvideBitmap`, at which time the picture tag will be set to either the onTag (pArgs == true) or the offTag (pArgs == false).

**See Also**

`msgIconProvideBitmap`

---

**msgIconProvideBitmap**

Sent to control client when icon needs the picture bitmap.

Takes P_ICON_PROVIDE_BITMAP, returns STATUS. Category: client notification.

**Comments**

`clsIconToggle` will alter pArgs->tag to be onTag if the current feedback state is on, or the offTag otherwise. This results in the client of the icon receiving this message and providing the on or off bitmap.
This file contains the API definition for clsLabel.

clsLabel inherits from clsControl.

Implements much of the appearance of many toolkit components inside the border: font, decoration, scale, orientation, etc.

**Debugging Flags**

The clsLabel debugging flag is '0'. Defined values are:

- flag4 (0x0010)  msgSave/msgRestore info
- flag5 (0x0020)  boxed string/paint dc

```c
#ifndef LABEL_INCLUDED
#define LABEL_INCLUDED
#include <control.h>
#include <sysgraf.h>
#ifndef CONTROL_INCLUDED
#endif
#ifndef SYSGRAF_INCLUDED
#endif
#endif
```

**Common #defines and typedefs**

```c
typedef OBJECT LABEL;
```

**Info style**

- `lsInfoString` 0 // info is pString
- `lsInfoWindow` 1 // info is WIN
- `lsInfoStringId` 2 // info is a resource file string id
- `lsInfoUnused` 3 // unused (reserved)

**X and Y alignment styles**

- `lsAlignLeft` 0 // left-justified
- `lsAlignCenter` 1 // centered
- `lsAlignRight` 2 // right-justified
- `lsAlignBottom` 3 // bottom-justified
- `lsAlignTop` 4 // top-justified
- `lsAlignCustom` 5 // send msgLabelAlign to self

**Decoration style**

- `lsDecorationNone` 0 // no decoration
- `lsDecorationBlank` 1 // blank space on left
- `lsDecorationNonExclusiveOn` 2 // left check and double bar
- `lsDecorationExclusiveOff` 3 // left blank and single bar
- `lsDecorationExclusiveOn` 4 // left check and single bar
- `lsDecorationPullRight` 5 // pull-right arrow on right
- `lsDecorationNonExclusiveOff` 6 // left blank and double bar
#define lsDecorationCheck 7 // left checkmark
#define lsDecorationCircle 8 // left empty circle
#define lsDecorationBox 9 // left empty box
#define lsDecorationCheckedBox 10 // left checked box
#define lsDecorationCheckedCircle 11 // left checked circle
#define lsDecorationHollowLeft 12 // left hollow left delta
#define lsDecorationHollowRight 13 // left hollow right delta
#define lsDecorationSolidLeft 14 // left solid left delta
#define lsDecorationSolidRight 15 // left solid right delta
#define lsDecorationPopup 16 // left solid right delta w/space
#define lsDecorationButtonOff 17 // left off button glyph
#define lsDecorationButtonOn 18 // left on button glyph
#define lsDecorationCustomLeft 19 // left custom decoration
#define lsDecorationCustomRight 20 // right custom decoration
// 21 // unused (reserved)
// .. // unused (reserved)
// 31 // unused (reserved)

Font Type

#define lsFontSystem 0 // use the system font
#define lsFontCustom 1 // use the specified font
#define lsFontUser 2 // use the system user font
//

Common Scale Values, in layout units

#define lsScaleTiny 2 // 2/8 x normal
#define lsScaleSmall 4 // 4/8 x normal
#define lsScaleMedium 6 // 6/8 x normal
#define lsScaleNormal 8 // 8/8 x normal
#define lsScaleLarge 10 // 10/8 x normal
#define lsScaleJumbo 12 // 12/8 x normal
#define lsScaleHuge 14 // 14/8 x normal

Rotation styles

#define lsRotateNone 0 // 0 degrees (horizontal, left to right)
#define lsRotate90 1 // 90 degrees (vertical, bottom to top)
#define lsRotate180 2 // 180 degrees (horizontal, right to left)
#define lsRotate270 3 // 270 degrees (vertical, top to bottom)

Underline styles

#define lsUnderlineNone 0 // no underline
#define lsUnderlineSingle 1 // single underline
#define lsUnderlineDouble 2 // double underline
// 3 // unused (reserved)

Box styles

#define lsBoxNone 0 // no boxes
#define lsBoxSquare 1 // square box around each character
#define lsBoxTicks 2 // tick mark between characters
#define lsBoxInvisible 3 // don’t draw the box lines
# Number of rows/columns

```c
#define lsNumAsNeeded 0 // as many rows/columns as needed
#define lsNumAbsolute 1 // fixed number: rows/cols
// 2 // unused (reserved)
// 3 // unused (reserved)
```

typedef struct LABEL_STYLE {
    U16 infoType : 2, // type of pString field
    xAlignment : 2,  // x alignment style
    yAlignment : 2,  // y alignment style
    rotation : 2,    // text rotation
    underline : 2,   // underline style
    strikeout : 1,   // strikeout during msgDcDrawText
    decoration : 5;  // decoration style
    U16 numCols : 2, // style for number of columns
    numRows : 2,    // style for number of rows
    box : 2,        // boxing style
    wordWrap : 1,   // word wrap to next row
    fontType : 2,   // system or custom font
    scaleUnits : 6, // scale units style, e.g. bsUnitsLayout
    stringSelected : 1; // whether content string shows sel'd visual
    U16 spare2 : 16; // unused (reserved)
} LABEL_STYLE, *P_LABEL_STYLE;
```

Default: LABEL_STYLE:

- infoType = IsInfoString
- xAlignment = IsLeft
- yAlignment = IsBottom
- decoration = IsDecorationNone
- scaleUnits = bsUnitsLayout
- rotation = IsRotateNone
- underline = IsUnderlineNone
- strikeout = false
- box = IsBoxNone
- numCols = lsNumAsNeeded
- numRows = lsNumAsNeeded
- wordWrap = false
- fontType = IsFontSystem
- scaleUnits = bsUnitsLayout
- stringSelected = false

# Messages

**msgNew**

Creates a label window.

Takes P_LABEL_NEW, returns STATUS. Category: class message.

```c
typedef struct LABEL_NEW_ONLY {
    LABEL_STYLE style; // overall style
    P_CHAR pString;    // string to display or child window
    SYSDC_FONT_SPEC font; // spec to open if style.fontType == IsFontCustom
    P_CHAR fontName; // font name from which to derive font.id
    U8 scale;        // scale in scaleUnits
    U8 rows;         // number of rows
    U8 cols;         // number of columns (or zero for infinite)
    U16 customGlyph; // custom decoration glyph
    U32 spare1;      // unused (reserved)
    U32 spare2;      // unused (reserved)
} LABEL_NEW_ONLY, *P_LABEL_NEW_ONLY;
```
The fields you commonly set are:

- `pArgs->label.style` appropriate style values
- `pArgs->label.pString` string or child window uid

In response to msgNew, the label initializes all of its state. This is the only time `pArgs->fontName` would be used.

Since clsLabel copies the bytes pointed to by `pArgs->pString` (when style.infoType is IsInfoString), the client may free the string after msgNew if the string was allocated.

If style.infoType is IsInfoStringld, clsLabel self-sends msgLabelBindStringld to bind the resid to a string.

### msgNewDefaults

Initializes the LABEL_NEW structure to default values.

Takes P_LABEL_NEW, returns STATUS. Category: class message.

```c
#define msgNewLabel Defaults
typedef struct LABEL_NEW {
    labelNewFields
} LABEL_NEW, *P_LABEL_NEW;
```

Comments

Zeroes out `pArgs->label` and sets:

- `pArgs->win.flags.style` |= wsShrinkWrapWidth | wsShrinkWrapHeight;
- `pArgs->border.style.leftMargin` = bsMarginSmall;
- `pArgs->border.style.rightMargin` = bsMarginSmall;
- `pArgs->border.style.bottomMargin` = bsMarginSmall;
- `pArgs->border.style.topMargin` = bsMarginSmall;
- `pArgs->label.style.scaleUnits` = bsUnitsLayout;
- `pArgs->label.scale` = lsScaleNormal;

Also sets `pArgs->label.font` to the default system font.

### msgLabelGetStyle

Passes back the current style values.

Takes P_LABEL_STYLE, returns STATUS.

```c
#define msgLabelGetStyle MakeMsg(clsLabel, 1)
typedef struct LABEL_STYLE {
    U16 infoType 2,
    II type of pString field
    xAlignment 2,
    II x alignment style
    yAlignment 2,
    II y alignment style
    rotation 2,
    II text rotation
    underline 2,
    II underline style
    strikeout 1,
    II strikeout during msgDoDrawText
    decoration 5;
    II decoration style
    numCols 2,
    II style for number of columns
    numRows 2,
    II style for number of rows
    box 2,
    II boxing style
```
typedef struct LABEL_STYLE {
    U16 infoType  : 2,    // type of pString field
    xAlignment    : 2,    // x alignment style
    yAlignment    : 2,    // y alignment style
    rotation      : 2,    // text rotation
    underline     : 2,    // underline style
    strikeout     : 1,    // strikeout during msgDcDrawText
    decoration    : 5;    // decoration style
    U16 numCols   : 2,    // style for number of columns
    numRows       : 2,    // style for number of rows
    box           : 2;    // boxing style
    wordWrap      : 1,    // word wrap to next row
    fontType      : 2,    // system or custom font
    scaleUnits    : 6,    // scale units style, e.g. bsUnitsLayout
    stringSelected: 1;    // whether content string shows sel’d visual
    U16 spare2    : 16;   // unused (reserved)
} LABEL_STYLE, *P_LABEL_STYLE;

msgLabelSetStyle

Sets the style fields.

Takes P_LABEL_STYLE, returns STATUS.

#define msgLabelSetStyle MakeMsg(clsLabel, 2)

Comments

If the decoration style changes, the label uses msgWinDirtyRect to dirty the appropriate portion of itself.

If the new style.box is not IsBoxNone, then the label self-sends msgLabelProvideBoxSize to obtain the width and height the boxes should be. If either of these differ from the old values, then the label self-sends msgWinSetLayoutDirty(true).

If the style.numCols or style.numRows change, or any of the other style values that might require relayout change, label self-sends msgWinSetLayoutDirty(true).

It is the caller's responsibility to re-layout the label if the caller has changed any style that affects the layout of the label.

msgLabelGetString

Fills P_ARGS->pString with the current string.

Takes P_CONTROL_STRING, returns STATUS.

#define msgLabelGetString MakeMsg(clsLabel, 3)

Comments

Will fill the passed buffer up to len bytes worth of the string. The copied string is not null-terminated if the passed len wasn’t large enough.

If the passed len is zero, clsLabel sets len to the number of bytes the buffer would have to be in order to hold the entire label's string (including the terminating null).
**msgLabelSetString**

Sets the label string.

Takes P_CHAR, returns STATUS.

```c
#define msgLabelSetString MakeMsg(clsLabel, 4)
```

Comments

Allocates storage and copies P_ARGS. Note that clsLabel allocates within the context of the current process.

---

**msgLabelGetUnicode**

Fills P_ARGS->pString with the current string.

Takes P_CONTROL_STRING, returns STATUS.

```c
#define msgLabelGetUnicode MakeMsg(clsLabel, 21)
```

Comments

Like msgLabelGetString, except that the client is requesting the string in unicode format (where a character is represented in 16 bits).

Will fill the passed buffer up to len characters worth of the string. The copied string is not null-terminated if the passed len wasn’t large enough.

If the passed len is zero, clsLabel sets len to the number of characters the buffer would have to be in order to hold the entire label’s string (including the terminating null). Note that the number of bytes would be twice this number.

---

**msgLabelSetUnicode**

Sets the label string.

Takes P_U16 (P_CHAR after its 16-bit), returns STATUS.

```c
#define msgLabelSetUnicode MakeMsg(clsLabel, 22)
```

Comments

Like msgLabelSetString, except that the client is specifying the string in unicode format (where a character is represented in 16 bits).

Allocates storage and copies P_ARGS. Note that clsLabel allocates within the context of the current process.

---

**msgLabelGetStringId**

Passes back the string resource id; zero if none.

Takes P_RESID, returns STATUS.

```c
#define msgLabelGetStringId MakeMsg(clsLabel, 25)
```

Comments

clsLabel returns stsFailed if style.infoType is not IsInfoStringId.

---

**msgLabelSetStringId**

Sets the string resource id.

Takes RESID, returns STATUS.

```c
#define msgLabelSetStringId MakeMsg(clsLabel, 26)
```

Comments

clsLabel immediately binds the specified string id to a string by self-sending msgLabelBindStringId.

The string id is remembered and saved during msgSave.
msgLabelBindStringId
Binds the string resource id to a string.
Takes VOID, returns STATUS.

```c
#define msgLabelBindStringId MakeMsg(clsLabel, 27)
```

clsLabel returns stsFailed if style.infoType is not IsInfoStringId.
clsLabel binds the current string id to a string by loading the string from the ProcessResList.

msgLabelGetWin
Passes back the child window.
Takes P_WIN, returns STATUS.

```c
#define msgLabelGetWin MakeMsg(clsLabel, 5)
```

clsLabel returns stsFailed if style.infoType is not IsInfoWin.

msgLabelSetWin
Sets the child window.
Takes WIN, returns STATUS.

```c
#define msgLabelSetWin MakeMsg(clsLabel, 6)
```

clsLabel returns stsFailed if style.infoType is not IsInfoWin.

Since changing the window within the label sets the label's wsLayoutDirty bit, the caller should re-layout the label.

msgLabelGetFontSpec
Passes back the font spec.
Takes P_SYSDC_FONT_SPEC, returns STATUS.

```c
#define msgLabelGetFontSpec MakeMsg(clsLabel, 8)
```

Note that this font spec is not used unless style.fontType is IsFontCustom.

msgLabelSetFontSpec
Sets the font spec.
Takes P_SYSDC_FONT_SPEC, returns STATUS.

```c
#define msgLabelSetFontSpec MakeMsg(clsLabel, 9)
```

Note that this font spec is not used unless style.fontType is IsFontCustom.

As with msgLabelSetStyle, it is the caller's responsibility to re-layout the label if the caller has changed any style that affects the layout of the label (such as certain fields in the font spec). Note that the label does not currently explicitly set its wsLayoutDirty bit in response to msgLabelSetFontSpec, but that this may change in the future.


**msgLabelGetScale**
Passes back the font scale.
Takes P_U8, returns STATUS.

```c
#define msgLabelGetScale MakeMsg(clsLabel, 10)
```

Comments
Note that the units of this scale are determined by style.scaleUnits.

**msgLabelSetScale**
Sets the font scale.
Takes U8, returns STATUS.

```c
#define msgLabelSetScale MakeMsg(clsLabel, 11)
```

Comments
Note that the units of this scale are determined by style.scaleUnits.

As with msgLabelSetStyle, it is the caller's responsibility to re-layout the label if the caller has changed any style that affects the layout of the label (such as the font scale). Note that the label does not currently explicitly set its wsLayoutDirty bit in response to msgLabelSetScale, but that this may change in the future.

**msgLabelGetRows**
Passes back the number of rows the label will size itself to.
Takes P_U8, returns STATUS.

```c
#define msgLabelGetRows MakeMsg(clsLabel, 12)
```

Comments
Note that this is not used unless style.numRows is IsNumAbsolute.

**msgLabelSetRows**
Sets the number of rows the label will size itself to.
Takes U8, returns STATUS.

```c
#define msgLabelSetRows MakeMsg(clsLabel, 13)
```

Comments
Note that this is not used unless style.numRows is IsNumAbsolute.

As with msgLabelSetStyle, it is the caller's responsibility to re-layout the label if the caller has changed any style that affects the layout of the label (such as the number of rows). Note that the label does not currently explicitly set its wsLayoutDirty bit in response to msgLabelSetRows, but that this may change in the future.

**msgLabelGetCols**
Passes back the number of columns the label will size itself to.
Takes P_U8, returns STATUS.

```c
#define msgLabelGetCols MakeMsg(clsLabel, 14)
```

Comments
Note that this is not used unless style.numCols is IsNumAbsolute.
**msgLabelSetCols**
Sets the number of columns the label will size itself to.
Takes U8, returns STATUS.

```c
#define msgLabelSetCols MakeMsg(clsLabel, 15)
```

**Comments**
Note that this is not used unless style.numCols is IsNumAbsolute.
As with **msgLabelSetStyle**, it is the caller's responsibility to re-layout the label if the caller has changed any style that affects the layout of the label (such as the number of columns). Note that the label does not currently explicitly set its wsLayoutDirty bit in response to **msgLabelSetCols**, but that this may change in the future.

**msgLabelGetCustomGlyph**
Passes back the custom decoration glyph, zero if none.
Takes P_U16, returns STATUS.

```c
#define msgLabelGetCustomGlyph MakeMsg(clsLabel, 23)
```

**Comments**
Note that this is not used unless style.decoration is IsDecorationCustomLeft or IsDecorationCustomRight.

**msgLabelSetCustomGlyph**
Sets the custom decoration glyph.
Takes U16, returns STATUS.

```c
#define msgLabelSetCustomGlyph MakeMsg(clsLabel, 24)
```

**Comments**
Note that this is not used unless style.decoration is IsDecorationCustomLeft or IsDecorationCustomRight.

**msgLabelGetBoxMetrics**
Passes back the current box metrics.
Takes P_LABEL_BOX_METRICS, returns STATUS.

```c
#define msgLabelGetBoxMetrics MakeMsg(clsLabel, 16)
```

**typedef struct LABEL_BOX_METRICS {**

- RECT32 boxRect; // origin and size of boxed area
- SIZE32 singleBoxSize; // size of a single box
- U16 rows, cols; // current # of rows and columns
- U16 baseline; // positive baseline offset from bottom of box
- U32 spare1; // unused (reserved)
- U32 spare2; // unused (reserved)
- U32 spare3; // unused (reserved)

`} LABEL_BOX_METRICS, *P_LABEL_BOX_METRICS;`

**Comments**
The box metrics describe the arrangement and size of the box cells imaged by the label. These metrics are valid only if style.box is not lsBoxNone.
All origins and sizes are in device units.
msgLabelResolveXY
Resolves a point to a character in the string.
Takes P_LABEL_RESOLVE, returns STATUS.

#define msgLabelResolveXY MakeMsg(clsLabel, 17)

Arguments
typedef struct LABEL_RESOLVE {
    XY32 xy;       // in: point to resolve
    S32  index;    // out: index of char at point
    U32  spare1;   // unused (reserved)
    U32  spare2;   // unused (reserved)
} LABEL_RESOLVE, *P_LABEL_RESOLVE;

Comments
An index of -1 indicates point is not over any character. The xy should be relative to the label and expressed in device units.

msgLabelAlign
Self-sent if style.xAlignment or style.yAlignment is lsAlignCustom.
Takes P_LABEL_ALIGN, returns STATUS. Category: self-sent.

#define msgLabelAlign MakeMsg(clsLabel, 7)

Arguments
typedef struct LABEL_ALIGN {
    BOOLEAN alignX;  // in: true if x alignment
    SIZE16 outerSize;  // in: size of label outer rect (in twips)
    SIZE16 innerSize;  // in: size of label inner rect (in twips)
    SIZE16 contentsSize;  // in: size of label contents (in twips)
    COORD16 offset;    // out: computed x or y offset from origin
    U32  spare;       // unused (reserved)
} LABEL_ALIGN, *P_LABEL_ALIGN;

Comments
Allows subclasses to compute alignment. The subclass should fill in pArgs->offset.

msgLabelProvideInsPt
Self-sent message to obtain where to render insertion point.
Takes P_S16, returns STATUS. Category: self-sent.

#define msgLabelProvideInsPt MakeMsg(clsLabel, 18)

Comments
Receiver should return the zero-based index of the character at which the insertion point should be drawn. Non-boxed styles draw the insertion point before this character, boxed styles highlight the box around this character.
If the returned index is -1, no insertion point is drawn. clsLabel responds by default with -1.

msgLabelGetRects
Computes the rectangle for each given character index.
Takes P_LABEL_RECT, returns STATUS.

#define msgLabelGetRects MakeMsg(clsLabel, 19)
#define lgrInsPtRect flag0

Arguments
typedef struct {
    S16  index;
    RECT32 rect;
    U16  flags;
    U16  spare;
} LABEL_RECT, *P_LABEL_RECT;
pArgs points to an array of LABEL_RECTs. The receiver computes the rectangle for the character at the index for each index until it encounters one whose value is -1. The rects are relative to the label, and are expressed in device units.

The indices should be sorted in increasing order.

**msgLabelProvideBoxSize**

Self-sent message to obtain the char box size.

Takes P_SIZE16, returns STATUS. Category: self-sent.

```
#define msgLabelProvideBoxSize MakeMsg(clsLabel, 20)
```

Receiver should fill in *pArgs with the size of a character box, in points. This message is self-sent when a boxed label is processing the following messages: msgInit, msgRestore, msgLabelSetString, and msgLabelSetStyle.

c1sLabel responds by filling in *pArgs from the user preferences (using prCharBoxWidth and prCharBoxHeight from prefs.h).

### Messages from Other Classes

**msgWinLayoutSelf**

Tell a window to layout its children.

Takes P_WIN_METRICS, returns STATUS.

```
clsLabel responds by recomputing its layout parameters and by using msgWinDelta on its child window (if style.infoType is IsInfoWindow).
```

**msgSave**

Causes an object to file itself in an object file.

Takes P_OBJ_SAVE, returns STATUS.

```
c1sLabel responds by filing away all its state, including its string (if style.infoType is IsInfoString) or child window (if style.infoType is IsInfoWindow).

Note that the child window must have wsSendFile set to be filed. If wsSendFile is not set, then msgSave does not save it, and a subsequent msgRestore sets the label's pString field to objNull. (wsSendFile is the default for c1sBorder and its descendents).
```

**msgRestore**

Creates and restores an object from an object file.

Takes P_OBJ_RESTORE, returns STATUS.

```
c1sLabel responds by restoring all of its state, including its string (if style.infoType is IsInfoString) or child window (if style.infoType is IsInfoWindow).

If the child window was not filed during the msgSave, then after msgRestore the label's pString value is objNull.
```
msgFree
Sent as the last of three msgs to destroy an object.
Takes OBJ_KEY, returns STATUS.

Comments clsLabel responds by freeing its string if style.infoType is IsInfoString and the string pointer is not null. clsLabel uses OSHheapBlockFree.

msgWinRepaint
Tells a window to repaint itself.
Takes nothing, returns STATUS. Category: descendant responsibility.

Comments clsLabel responds by painting its decoration and string as appropriate.

msgWinGetBaseline
Gets the desired x,y alignment of a window.
Takes P_WIN_METRICS, returns STATUS.

Comments clsLabel responds by setting pArgs->bounds.origin.
If the label is displaying a decoration, the x coordinate is set to the x offset of the rightmost decoration position (there's a small gap between this position and the start of the string/window). If the label has no decoration, then the x coordinate is set to the offset of the left side of the string/window.
The y coordinate is set to a value derived from the label's innerRect origin and the baseline information from the label's font. This value is accurate in those cases where the label's bottom fits snugly around the string/window, but is incorrect in cases where this doesn't hold (e.g., a non-wsShrinkWrapHeight label that is taller than it needs to be).

See Also msgBorderGetInnerRect baseline coordinates are derived from this

msgEmbeddedWinGetMark
Get an embedded window mark.
Takes P_EMBEDDED_WIN_MARK, returns STATUS.

Comments clsLabel responds by copying into pArgs->label, then ensures that the buffer is terminated with a null character.
If style.infoType is not IsInfoString, or the label's string is null or empty, then clsLabel does nothing.

msgBorderPaintForeground
category: subclass window responsibility Receiver must paint the foreground, if any.
Takes VOID, returns STATUS.

Comments clsLabel responds by using msgWinBeginPaint, painting its decoration and string as appropriate, and then sending msgWinEndPaint.
**msgControlSetDirty**
Clears the dirty bit.
Takes BOOLEAN, returns STATUS.

**Comments**
clsLabel responds by calling ancestor, then checking the CONTROL_STYLE.showDirty value. If this is false, clsLabel just returns. Otherwise, if the old CONTROL_STYLE.dirty value is different from the new value, then clsLabel uses msgWinDirtyRect to dirty its decoration (if it has one).

**See Also**
msgControlSetStyle sets the CONTROL_STYLE values
msgControlSetMetrics sets the CONTROL_METRICS values
msgWinDirtyRect dirties a portion of a window

**msgControlSetStyle**
Sets the style values.
Takes P_CONTROL_STYLE, returns STATUS.

**Comments**
clsLabel responds by calling ancestor, then checking the CONTROL_STYLE.showDirty value. If this is false, clsLabel just returns. Otherwise, if the old CONTROL_STYLE.dirty value is different from the new value, then clsLabel uses msgWinDirtyRect to dirty its decoration (if it has one).

**See Also**
msgControlSetDirty sets the CONTROL_STYLE.dirty bit
msgControlSetMetrics sets the CONTROL_METRICS values
msgWinDirtyRect dirties a portion of a window

**msgControlSetMetrics**
Sets the metrics.
Takes P_CONTROL_METRICS, returns STATUS.

**Comments**
clsLabel responds by calling ancestor, then checking the CONTROL_STYLE.showDirty value. If this is false, clsLabel just returns. Otherwise, if the old CONTROL_STYLE.dirty value is different from the new value, then clsLabel uses msgWinDirtyRect to dirty its decoration (if it has one).

**See Also**
msgControlSetStyle sets the CONTROL_STYLE values
msgControlSetDirty sets the CONTROL_STYLE.dirty bit
msgWinDirtyRect dirties a portion of a window
This file contains the API for clsListBox.
clsListBox inherits from clsScrollWin.

Implements a scrolling list of windows (of arbitrary length).

The windows that the listBox manages may be of any class, and they may be of different classes within a listBox. The windows may have different heights as well. The listBox will constrain their widths as per various style settings.

A listBox is useful when the number of windows that could be displayed is unknown, variable, or large (say 30 or more). The listBox will, by default, destroy those windows that have scrolled out of view, thus keeping the number of windows in existence to a reasonable quantity.

By using a listBox, you trade performance for generality. If the number of windows is likely to be small and not particularly variable, you may choose to put a tableLayout window in as the clientWin of a scrollWin instead. The visual effect would be the same as for a listBox, but each of the tableLayout's child windows would, by default, be around for the lifetime of the parent (and as more windows are put on the screen, the overall performance of the UI degrades).

As with most UI Toolkit classes, you may use clsListBox as-is, or create your own subclass for special purposes. Since a common use of a listBox is to present a simple list of strings to the user, you may use clsStringListBox instead (see strlbox.h). That class presents a somewhat simpler API for this common usage. A subclass of clsStringListBox is clsFontListBox, which gets its strings from the list of currently installed fonts on the system. clsFontListBox proves useful in situations such as option sheets (see fontlbox.h).

### Debugging Flags

The clsListBox debugging flag is 'K'. Defined values are:

flag12 (0x1000)  general
ifndef LISTBOX_INCLUDED
#define LISTBOX_INCLUDED

 ifndef SWIN_INCLUDED
 #include <swin.h>
 endif

### Common #defines and typedefs

typedef OBJECT LIST_BOX;

### ListBox Filing Styles

#define lbFileMin 0       // file minimum data necessary
#define lbFileEntryInfo 1 // lbFileMin + entry info except windows
#define lbFileAll 2       // lbFileEntryInfo + windows

typedef struct {
    U16    filing : 2,
    spare : 14;
} LIST_BOX_STYLE, *P_LIST_BOX_STYLE;
Default style:

    filing = lbFileAll

typedef struct {
    LIST_BOX_STYLE style;
    OBJECT client; // client to send list box messages to.
    U16 nEntries; // total number of entries in list box.
    U16 nEntriesToView; // show this many entries at a time.
    U32 spare;
} LIST_BOX_METRICS, *P_LIST_BOX_METRICS;

Enum16(LIST_BOX_DATA_FREE_MODE) {
    lbFreeDataNotVisible = flag0,
    lbFreeDataWhenDestroyed = flag1,
    lbFreeDataByMessage = flag2,
    lbFreeDataDefault = lbFreeDataNotVisible | lbFreeDataWhenDestroyed
};

Enum16(LIST_BOX_ENTRY_STATE) {
    lbSelected = flag0,
    lbOpen = flag1,
    lbBusy = flag2,
    lbStateDefault = 0 // Not selected, not open
};

typedef struct LIST_BOX_ENTRY {
    WIN listBox; // in/out: requestor
    U16 position; // in: entry position
    WIN win; // in/out: entry window to display
    U16 freeEntry; // in/out: LIST_BOX_DATA_FREE_MODE
    U16 state; // in/out: LIST_BOX_ENTRY_STATE
    P_UNKNOWN data; // in/out: client data
    P_UNKNOWN arg; // message specific argument
    U32 spare; // reserved
} LIST_BOX_ENTRY, *P_LIST_BOX_ENTRY;

typedef struct LIST_BOX_POSITION_XY {
    XY32 place; // in
    U16 position; // in/out
    U32 spare; // unused (reserved)
} LIST_BOX_POSITION_XY, *P_LIST_BOX_POSITION_XY;

typedef struct LIST_BOX_ENTRY_ENUM {
    U16 max; // in = size of pEntry[] array.
    U16 count; // in = # of entries to return in array.
    // If count > max then memory may be allocated.
    // out = # of valid entries in array.
    U16 next; // in = 0 to start at beginning
    // OR previous out value to pick up
    // where we left off.
    P_LIST_BOX_ENTRY pEntry; // in = Ptr to array of entries.
    // out = If memory was allocated client should free the memory.
    U16 flags; // in = state flags to filter on.
    U32 spare; // unused (reserved)
} LIST_BOX_ENTRY_ENUM, *P_LIST_BOX_ENTRY_ENUM;

#define stsListBoxEmpty MakeStatus(clsListBox, 1)

msgNew

Creates a list box (initially empty).

Takes P_LIST_BOX_NEW, returns STATUS. Category: class message.

typedef LIST_BOX_METRICS LIST_BOX_NEW_ONLY, *P_LIST_BOX_NEW_ONLY;
#define listBoxNewFields MakeNew
#define scrollWinNewFields
LIST_BOX_NEW_ONLY listBox;
typedef struct {
    listBoxNewFields
} LIST_BOX_NEW, *P_LIST_BOX_NEW;

clsListBox sets the following values before calling its ancestor:

pArgs->scrollWin.style.getDelta = false;
pArgs->scrollWin.style.vertClient = swClientWin;
pArgs->scrollWin.style.horizClient = swClientScrollWin;
pArgs->scrollWin.style.getFieSize = true;
pArgs->scrollWin.style.forward = swForwardGesture;

msgNewDefaults

Initializes the LIST_BOX_NEW structure to default values.

Takes P_LIST_BOX_NEW, returns STATUS. Category: class message.

typedef struct {
    listBoxNewFields
} LIST_BOX_NEW, *P_LIST_BOX_NEW;

clsListBox sets the following values:

pArgs->win.flags.style &= wsShrinkWrapHeight;
pArgs->win.flags.style |= wsShrinkWrapWidth;
pArgs->border.style.edge = bsEdgeAll;
pArgs->scrollWin.style.expandChildWidth = true;
pArgs->listBox.style.filing = lbFileAll;
pArgs->listBox.client = objNull;
pArgs->listBox.nEntries = 0;
pArgs->listBox.nEntriesToView = 6;
pArgs->listBox.spare = 0;

msgListBoxGetMetrics

Passes back the metrics for a listBox.

Takes P_LIST_BOX_METRICS, returns STATUS.

#define msgListBoxGetMetrics MakeMsg(clsListBox, 1)

typedef struct {
    LIST_BOX_STYLE style;
    OBJECT client;   // client to send list box messages to.
    U16 nEntries;    // total number of entries in list box.
    U16 nEntriesToView; // show this many entries at a time.
    U32 spare;
} LIST_BOX_METRICS, *P_LIST_BOX_METRICS;

msgListBoxSetMetrics

Sets the metrics for a listBox.

Takes P_LIST_BOX_METRICS, returns STATUS.

#define msgListBoxSetMetrics MakeMsg(clsListBox, 2)

typedef struct {
    LIST_BOX_STYLE style;
    OBJECT client;   // client to send list box messages to.
    U16 nEntries;    // total number of entries in list box.
    U16 nEntriesToView; // show this many entries at a time.
    U32 spare;
} LIST_BOX_METRICS, *P_LIST_BOX_METRICS;
You should send `msgWinLayout` to the `listBox` if the value of `nEntriesToView` has changed.

The `listBox` might ask for new entries after the `SetMetrics` call returns if the value of `nEntries` has changed.

**msgListBoxAppendEntry**

Appends an entry to the list box after the specified position.

Takes `P_LIST_BOX_ENTRY`, returns `STATUS`.

```
#define msgListBoxAppendEntry MakeMsg(clsListBox, 3)
```

```
typedef struct LIST_BOX_ENTRY {
  WIN listBox; // in/out: requestor
  U16 position; // in: entry position
  WIN win; // in/out: entry window to display
  U16 freeEntry; // in/out: LIST_BOX_DATA_FREE_MODE
  U16 state; // in/out: LIST_BOX_ENTRY_STATE
  P_UNKNOWN data; // in/out: client data
  P_UNKNOWN arg; // message specific argument
  U32 spare; // reserved
} LIST_BOX_ENTRY, *P_LIST_BOX_ENTRY;
```

Comments

If `win` is `objNull`, the client will receive `msgListBoxProvideEntry` when the entry needs to be displayed.

This is computationally expensive when the `listBox` has a parent. In other words, all work necessary to fix up the `listBox` is performed immediately. If you want to batch several calls, consider extracting the `listBox` first.

See Also

- `msgListBoxAppendEntry` similar, but appends after

**msgListBoxInsertEntry**

Insert an entry to the list box before the specified position.

Takes `P_LIST_BOX_ENTRY`, returns `STATUS`.

```
#define msgListBoxInsertEntry MakeMsg(clsListBox, 4)
```

```
typedef struct LIST_BOX_ENTRY {
  WIN listBox; // in/out: requestor
  U16 position; // in: entry position
  WIN win; // in/out: entry window to display
  U16 freeEntry; // in/out: LIST_BOX_DATA_FREE_MODE
  U16 state; // in/out: LIST_BOX_ENTRY_STATE
  P_UNKNOWN data; // in/out: client data
  P_UNKNOWN arg; // message specific argument
  U32 spare; // reserved
} LIST_BOX_ENTRY, *P_LIST_BOX_ENTRY;
```

Comments

If `win` is `objNull`, the client will receive `msgListBoxProvideEntry` when the entry needs to be displayed.

This is computationally expensive when the `listBox` has a parent. In other words, all work necessary to fix up the `listBox` is performed immediately. If you want to batch several calls, consider extracting the `listBox` first.

See Also

- `msgListBoxAppendEntry` similar, but inserts before
msgListBoxRemoveEntry
Removes an entry from the list box.
Takes U16, returns STATUS.
#define msgListBoxRemoveEntry MakeMsg(clsListBox, 5)

Comments
If the item was added with freeEntry != 0, then the item is freed automatically by the list box.
This is computationally expensive when the listBox has a parent. In other words, all work necessary to
fix up the listBox is performed immediately. If you want to batch several calls, consider extracting the
listBox first.
Return Value
stsBadParam the specified position is >= number of entries

msgListBoxGetEntry
Gets an entry in a listBox by position.
Takes P_LIST_BOX_ENTRY, returns STATUS.
#define msgListBoxGetEntry MakeMsg(clsListBox, 6)

typedef struct LIST_BOX_ENTRY {
WIN listBox; // in/out: requestor
U16 position; // in: entry position
WIN win; // in/out: entry window to display
U16 freeEntry; // in/out: LIST_BOX_DATA_FREE_MODE
U16 state; // in/out: LIST_BOX_ENTRY_STATE
P_UNKNOWN data; // in/out: client data
P_UNKNOWN arg; // message specific argument
U32 spare; // reserved
} LIST_BOX_ENTRY, *P_LIST_BOX_ENTRY;

Comments
Will pass back the last one if the passed position is maxU16.
Return Value
stsListBoxEmpty the list box has no entries
stsNoMatch the list box has no entry at that position

msgListBoxSetEntry
Sets an entry's information.
Takes P_LIST_BOX_ENTRY, returns STATUS.
#define msgListBoxSetEntry MakeMsg(clsListBox, 7)

typedef struct LIST_BOX_ENTRY {
WIN listBox; // in/out: requestor
U16 position; // in: entry position
WIN win; // in/out: entry window to display
U16 freeEntry; // in/out: LIST_BOX_DATA_FREE_MODE
U16 state; // in/out: LIST_BOX_ENTRY_STATE
P_UNKNOWN data; // in/out: client data
P_UNKNOWN arg; // message specific argument
U32 spare; // reserved
} LIST_BOX_ENTRY, *P_LIST_BOX_ENTRY;

Comments
Typically this message is used to set an entry's data or flag values.
This message prohibits the caller from changing whether the entry has a window (by specifying an
objNull pArgs->win when the entry has a window or vice versa). Clients should use
Append/Insert/Remove for this purpose. msgListBoxSetEntry does support replacing a window with a different one.

Replacing an entry window is computationally expensive when the listBox has a parent.

Return Value

stsListBoxEmpty  the list box has no entries
stsBadParam   attempt to add or remove an entry

msgListBoxFindEntry

Finds the position of the given entry window/data.
Takes P_LIST_BOX_ENTRY, returns STATUS.

#define msgListBoxFindEntry MakeMsg(clsListBox, 8)

typedef struct LIST_BOX_ENTRY {
    WIN listBox; // in/out: requestor
    U16 position; // in: entry position
    WIN win; // in/out: entry window to display
    U16 freeEntry; // in/out: LIST_BOX_DATA_FREE_MODE
    U16 state; // in/out: LIST_BOX_ENTRY_STATE
    P_UNKNOWN data; // in/out: client data
    P_UNKNOWN arg; // message specific argument
    U32 spare; // reserved
} LIST_BOX_ENTRY, *P_LIST_BOX_ENTRY;

If pArgs->win is non-null, clsListBox searches for an entry whose window matches pArgs->win. If pArgs->win is null, then clsListBox searches for an entry whose data fields matches pArgs->data.

Return Value

stsListBoxEmpty  the list box has no entries
stsNoMatch   the list box had no matching entry

msgListBoxEnum

Enumerates the entries of a listBox according to the given flags.
Takes P_LIST_BOX_ENTRY_ENUM, returns STATUS.

#define msgListBoxEnum MakeMsg(clsListBox, 9)

typedef struct LIST_BOX_ENTRY_ENUM {
    U16 max; // in = size of pEntry[] array.
    U16 count; // in = # of entries to return in array.  // If count > max then memory may be allocated.
    U16 next; // in = 0 to start at beginning  // OR previous out value to pick up where we left off.
    P_LIST_BOX_ENTRY pEntry; // in = Ptr to array of entries.  // out = If memory was allocated client should free the memory.
    U16 flags; // in = state flags to filter on.
    U32 spare; // unused (reserved)
} LIST_BOX_ENTRY_ENUM, *P_LIST_BOX_ENTRY_ENUM;
msgListBoxEntryIsVisible

Passes back the visibility of an entry in a listBox.

Takes P_LIST_BOX_ENTRY, returns STATUS.

```c
#define msgListBoxEntryIsVisible MakeMsg(clsListBox, 10)
```

**Message**

typedef struct LIST_BOX_ENTRY {
  WIN listBox; // in/out: requestor
  U16 position; // in: entry position
  WIN win; // in/out: entry window to display
  U16 freeEntry; // in/out: LIST_BOX_DATA_FREE_MODE
  U16 state; // in/out: LIST_BOX_ENTRY_STATE
  P_UNKNOWN data; // in/out: client data
  P_UNKNOWN arg; // message specific argument
  U32 spare; // reserved
} LIST_BOX_ENTRY, *P_LIST_BOX_ENTRY;

**Arguments**

Sets the 'arg' field to zero if not visible, non-zero if visible. If the position is maxU16, then uses pArgs->win instead.

msgListBoxXYToPosition

Gets the position for a given listBox window coordinate.

Takes P_LIST_BOX_POSITION_XY, returns STATUS.

```c
#define msgListBoxXYToPosition MakeMsg(clsListBox, 11)
```

**Message**

typedef struct LIST_BOX_POSITION_XY {
  XY32 place; // in
  U16 position; // in/out
  U32 spare; // unused (reserved)
} LIST_BOX_POSITION_XY, *P_LIST_BOX_POSITION_XY;

**Arguments**

This message resolves positions only to currently visible entry windows. It does not attempt to interpolate arbitrary coordinates to positions.

pArgs->place should be in the listBox's clientWin space.

**Return Value**

stsNoMatch the place did not intersect any currently visible entry windows

msgListBoxMakeEntryVisible

Makes the specified entry visible.

Takes P_LIST_BOX_ENTRY, returns STATUS.

```c
#define msgListBoxMakeEntryVisible MakeMsg(clsListBox, 12)
```

**Message**

typedef struct LIST_BOX_ENTRY {
  WIN listBox; // in/out: requestor
  U16 position; // in: entry position
  WIN win; // in/out: entry window to display
  U16 freeEntry; // in/out: LIST_BOX_DATA_FREE_MODE
  U16 state; // in/out: LIST_BOX_ENTRY_STATE
  P_UNKNOWN data; // in/out: client data
  P_UNKNOWN arg; // message specific argument
  U32 spare; // reserved
} LIST_BOX_ENTRY, *P_LIST_BOX_ENTRY;

**Arguments**

If the specified position is maxU16, msgListBoxFindEntry is first used to find the given window. If the position is not visible, it will be scrolled so that its top is at the center of the view. Otherwise, the minimum amount is scrolled. No msgWinUpdate is required.
Self-Sent/Client Messages

All of the messages in this section are first sent to the listBox itself. This is so that subclasses of clsListBox may intercept these messages and process them as desired. If these messages reach the clsListBox message handler, they will be forwarded on to the listBox client.

**msgListBoxProvideEntry**

Self-sent when a listBox needs a window for display.

Takes P_LIST_BOX_ENTRY, returns STATUS. Category: self-sent/client responsibility.

```c
#define msgListBoxProvideEntry MakeMsg(clsListBox, 13)
```

```c
ttypedef struct LIST_BOX_ENTRY {
    WIN listBox; // in/out: requestor
    U16 position; // in: entry position
    WIN win; // in/out: entry window to display
    U16 freeEntry; // in/out: LIST_BOX_DATA_FREE_MODE
    U16 state; // in/out: LIST_BOX_ENTRY_STATE
    P_UNKNOWN data; // in/out: client data
    P_UNKNOWN arg; // message specific argument
    U32 spare; // reserved
} LIST_BOX_ENTRY, *P_LIST_BOX_ENTRY;
```

**Comments**

The client should pass back a window UID in the win field. The client should also set the freeEntry, state, and data fields as desired. Note that the state and data fields have no meaning to clsListBox; they're uninterpreted fields for the client to use for any purpose.

A listBox will send this message even when the position it's asking for is >= the number of entries specified for the listBox. In this case, the client is free to return a non-zero status value, indicating to the listBox that no entry should be created for that position. Providing another entry window in this case allows the user to create new entries by merely scrolling past the end of the list.

If the message reaches the standard listBox message procedure, the listBox will forward the message to the client. This scheme allows subclasses of clsListBox to handle the message in a different way.

**msgListBoxDestroyEntry**

Sent to the client for an entry that has lbFreeDataByMessage enabled.

Takes P_LIST_BOX_ENTRY, returns STATUS. Category: self-sent/client responsibility.

```c
#define msgListBoxDestroyEntry MakeMsg(clsListBox, 14)
```

```c
ttypedef struct LIST_BOX_ENTRY {
    WIN listBox; // in/out: requestor
    U16 position; // in: entry position
    WIN win; // in/out: entry window to display
    U16 freeEntry; // in/out: LIST_BOX_DATA_FREE_MODE
    U16 state; // in/out: LIST_BOX_ENTRY_STATE
    P_UNKNOWN data; // in/out: client data
    P_UNKNOWN arg; // message specific argument
    U32 spare; // reserved
} LIST_BOX_ENTRY, *P_LIST_BOX_ENTRY;
```

**Comments**

The client should destroy the entry win and free any storage pointed to by the entry's 'data' field.
**msgListBoxEntryGesture**

Notifies the subclass / client that a gesture occurred over an entry.

Takes P_LIST_BOX_ENTRY, returns STATUS. Category: self-sent/client responsibility.

```c
#define msgListBoxEntryGesture MakeMsg(clsListBox, 15)
```

```c
typedef struct LIST_BOX_ENTRY { 
  WIN listBox;    // in/out: requestor
  U16 position;   // in: entry position
  WIN win;        // in/out: entry window to display
  U16 freeEntry;  // in/out: LIST_BOX_DATA_FREE_MODE
  U16 state;      // in/out: LIST_BOX_ENTRY_STATE
  P_UNKNOWN data; // in/out: client data
  PUNKNOWN arg;   // message specific argument
  U32 spare;      // reserved
} LIST_BOX_ENTRY, *P_LIST_BOX_ENTRY;
```

**Comments**

The 'arg' field contains a P_GWIN_GESTURE pointer.

If the position is maxU16, this means that the listbox currently has no entry windows. Any other value indicates the position of the entry window to which the gesture is directed. The listbox will use `msgGWinTransformGesture` to translate the coordinates in the GWIN_GESTURE to be relative to the entry window.

The listbox returns (from its `msgGWinGesture/msgGWinForwardedGesture` handler) the status resulting from self-sending `msgListBoxEntryGesture`. This means that the client should return stsOK, stsRequestDenied, or stsRequestForward as appropriate. See gwin.h.

---

**Messages from Other Classes**

**msgWinStartPage**

Advises window that it is on a printer, and printing is about to start.

Takes pNull, returns STATUS. Category: advisory message.

**Comments**

clsListBox responds by ensuring that its `clientWin` is appropriately populated with entry windows.

**msgWinGetBaseline**

Gets the desired x,y alignment of a window.

Takes P_WIN_METRICS, returns STATUS.

**Comments**

clsListBox will set pArgs->bounds.origin.x to 0. If there is an entry window visible, pArgs->bounds.origin.y is set to:

- `<top of scrollWin's inner window> - <row height>`
- `<y baseline of first visible entry window>`

If no entry window is visible, the y is set to 0.
This file contains the API for clsManager.

clsManager inherits from clsObject.

Provides an abstract manager class and associated protocol.

Managers are used to implement group behavior for collections of components. For example, each instance of clsChoice uses one to change the state of child buttons when the user is tapping on the choice's children. Also, the menu button holding onto a menu uid acts as a manager for that menu. Manager uids are held by instances of clsTkTable.

 ifndef MANAGER_INCLUDED
 define MANAGER_INCLUDED
 define managerNewFields
 objectNewFields
 endif

 A manager returns stsManagerContinue if it wishes msgWinSend propagation to continue. Any other return value causes propagation to stop and the return value to be passed back to the original msgWinSend sender.

 define stsManagerContinue MakeMsg(clsManager, 1)
This file contains the API definition for clsMenuButton.

clsMenuButton inherits from clsButton.

Menu buttons support an optional pull-down or pull-right pop-up menu.

```c
#ifndef MBUTTON_INCLUDED
#define MBUTTON_INCLUDED

#ifndef BUTTON_INCLUDED
#include <button.h>
#endif

Common #defines and typedefs
typedef OBJECT MENU_BUTTON;

Submenu Types
#define mbMenuNone 0 // no sub-menu defined
#define mbMenuPullDown 1 // sub-menu is pull-down
#define mbMenuPullRight 2 // sub-menu is pull-right
#define mbMenuPopup 3 // sub-menu is popup
#define mbMenuSibling 4 // sub-menu is a window sibling
//
//
//
//
//
//
//
//
//
//
//
//
```

Menu Actions
```c
#define mbAction1Tap 0 // menu up/down on xgs1Tap or msgPenUp
#define mbAction2Tap 1 // menu up/down on xgs2Tap
typedef struct MENU_BUTTON_STYLE { U16 subMenuType, // sub-menu type getWidth, // self-send msgMenuButtonProvideWidth getMenu, // send msgMenuButtonProvideMenu to client enableMenu, // send msgControlEnable to menu menuAction, // action to bring up/down menu menuIsUp, // read-only: true => menu is up spare; // unused (reserved) } MENU_BUTTON_STYLE, *P_MENU_BUTTON_STYLE;
```

Default MENU_BUTTON_STYLE:
```c
subMenuType = mbMenuNone
getWidth = false
getMenu = false
enableMenu = false
menuAction = mbAction1Tap
menuIsUp = false
```

```c
typedef struct MENU_BUTTON_PROVIDE_MENU { MENU_BUTTON menuButton; // In: requestor WIN menu; // In/Out: uid of menu U32 spare; // reserved (unused) } MENU_BUTTON_PROVIDE_MENU, *P_MENU_BUTTON_PROVIDE_MENU;
```
Messages

msgNew

Creates a menu button window.

Takes P_MENU_BUTTON_NEW, returns STATUS. Category: class message.

Arguments
typedef struct MENU_BUTTON_NEW_ONLY {
   MENU_BUTTON_STYLE style; // overall style
   WIN menu; // sub-menu or objNull
   U32 spare1; // unused (reserved)
   U32 spare2; // unused (reserved)
} MENU_BUTTON_NEW_ONLY, *P_MENU_BUTTON_NEW_ONLY;
define menuButtonNewFields \
   buttonNewFields \
   MENU_BUTTON_NEW_ONLY menuButton;
typedef struct MENU_BUTTON_NEW {
   menuButtonNewFields
} MENU_BUTTON_NEW, *P_MENU_BUTTON_NEW;

Comments

The fields you commonly set are:

pArgs->menuButton.style.subMenuType kind of subMenu
pArgs->menuButton.menu uid of menu window

If pArgs->menuButton.style.subMenuType is mbMenuPullRight, sets pArgs->label.style.decoration to lsDecorationPullRight.

If pArgs->menuButton.style.subMenuType is not mbMenuNone, sets pArgs->button.style.contact to bsContactToggle.

If pArgs->menuButton.menu is not objNull, it self-sends msgWinSetPopup with WIN_METRICS parameters of child = menu;

msgNewDefaults

Initializes the MENU_BUTTON_NEW structure to default values.

Takes P_MENU_BUTTON_NEW, returns STATUS. Category: class message.

Arguments
typedef struct MENU_BUTTON_NEW {
   menuButtonNewFields
} MENU_BUTTON_NEW, *P_MENU_BUTTON_NEW;

Comments

Zeroes our pArgs->menuItemButton and sets:

pArgs->win.flags.style |= wsFileNoBounds;
pArgs->border.style.edge = bsEdgeNone;
pArgs->border.style.join = bsJoinSquare;
pArgs->border.style.shadow = bsShadowNone;
pArgs->gWin.style.gestureEnable = false;
pArgs->control.style.showDirty = false;
pArgs->label.style.xAlignment = lsAlignLeft;
pArgs->label.style.yAlignment = lsAlignBottom;
**msgSave**
Causes an object to file itself in an object file.
Takes P_OBJ_SAVE, returns STATUS.
If the menu button has a menu, and the menu has wsSendFile on, msgSave be sent to the menu passing along pArgs.

**msgRestore**
Creates and restores an object from an object file.
Takes P_OBJ_RESTORE, returns STATUS.
clsMenuButton restores the instance from the file. If the menu buttona menu when filed, the menu is restored and the following is done:Sends msgTkTableSetManager, with pArgs of self to the menu. Self-sends msgWinSetPopup with WIN_METRICS parameters of child = menu;

**msgFree**
Sent as the last of three msgs to destroy an object.
Takes OBJ_KEY, returns STATUS.
If the menu button has a menu, msgDestroy is sent to the menu.

**msgMenuButtonGetStyle**
Passes back the current style values.
Takes P_MENU_BUTTON_STYLE, returns STATUS.
#define msgMenuButtonGetStyle MakeMsg(clsMenuButton, 1)

typedef struct MENU_BUTTON_STYLE {
  U16 subMenuType : 3, // sub-menu type
  getWidth : 1, // self-send msgMenuButtonProvideWidth
  getMenu : 1, // send msgMenuButtonProvideMenu to client
  enableMenu : 1, // send msgControlEnable to menu
  menuAction : 2, // action to bring up/down menu
  menusUp : 1, // read-only: true => menu is up
  spare : 7; // unused (reserved)
} MENU_BUTTON_STYLE, *P_MENU BUTTON_STYLE;

**msgMenuButtonSetStyle**
Sets the style values.
Takes P_MENU_BUTTON_STYLE, returns STATUS.
#define msgMenuButtonSetStyle MakeMsg(clsMenuButton, 2)

typedef struct MENU_BUTTON_STYLE {
  U16 subMenuType : 3, // sub-menu type
  getWidth : 1, // self-send msgMenuButtonProvideWidth
  getMenu : 1, // send msgMenuButtonProvideMenu to client
  enableMenu : 1, // send msgControlEnable to menu
  menuAction : 2, // action to bring up/down menu
  menusUp : 1, // read-only: true => menu is up
  spare : 7; // unused (reserved)
} MENU_BUTTON_STYLE, *P_MENU BUTTON_STYLE;

Note that style.menusUp is read-only -- pArgs->menusUp will be ignored.
If style.subMenuType changes the following is done:

- if style.subMenuType is mbMenuPullRight, the label.style.decoration is set to IsDecoratePullRight, otherwise it is set to IsDecorateNone.
- if the menu button has a menu, button.style.contact is set to bsContactToggle, otherwise bsContactMomentary.
- if the menu button has a menu, self-sends msgWinSetPopup with WIN_METRICS parameters of child = menu;

### msgMenuButtonGetMenu

Passes back the menu, objNull if none.

Takes P_MENU, returns STATUS.

```c
#define msgMenuButtonGetMenu MakeMsg(clsMenuButton, 3)
```

**Comments**

Note that this message does not result in msgMenuButtonProvideMenu to the menu button’s client, even if style.getMenu is true. To retrieve the menu that will be shown, send msgMenuButtonProvideMenu to the menu button.

**See Also**

msgMenuButtonProvideMenu

### msgMenuButtonSetMenu

Sets the menu.

Takes MENU, returns STATUS.

```c
#define msgMenuButtonSetMenu MakeMsg(clsMenuButton, 4)
```

**Comments**

The submenu is only used if style.subMenuType is not mbMenuNone. Note that the old menu, if any, is not freed. If the new menu is not objNull, self-sends msgWinSetPopup with WIN_METRICS parameters of

child = menu;

### msgMenuButtonProvideWidth

Self-sent when style.getWidth is true.

Takes P_S32, returns STATUS. Category: self-sent.

```c
#define msgMenuButtonProvideWidth MakeMsg(clsMenuButton, 7)
```

**Comments**

Subclasses should respond with the desired width of the menu button. clsMenuButton responds with self’s current window width.

### msgMenuButtonInsertMenu

Self-sent when style.menuAction is detected.

Takes WIN, returns STATUS. Category: self-sent.

```c
#define msgMenuButtonInsertMenu MakeMsg(clsMenuButton, 10)
```

**Comments**

Subclasses should respond by inserting pArgs into the window tree. If style.subMenuType is mbMenuSibling, clsMenuButton responds by inserting pArgs as a window sibling to self. Otherwise, msgMenuShow(true), is sent to pArgs.
msgMenuButtonExtractMenu
Self-sent when style.menuAction is detected.

Takes WIN, returns STATUS. Category: self-sent.

#define msgMenuButtonExtractMenu MakeMsg(clsMenuButton, 11)

Comments
Subclasses should respond by extracting pArgs from the window tree. clsMenuButton responds by sending msgMenuShow(false) to pArgs. If style.subMenuType is mbMenuSibling, clsMenuButton responds by sending msgWinExtract to pArgs. Otherwise, msgMenuShow(false), is sent to pArgs.

msgMenuButtonShowMenu Arguments
Enum16 (MENU_BUTTON_SHOW_MENU) {
    mbShowToggle = 0, // toggle the state of the menu
    mbShowExtract = 1, // take down the menu
    mbShowInsert = 2  // put up the menu
};

msgMenuButtonShowMenu
Puts up or takes down the menu.

Takes MENU_BUTTON_SHOW_MENU, returns STATUS.

#define msgMenuButtonShowMenu MakeMsg(clsMenuButton, 5)

Message
Enum16 (MENU_BUTTON_SHOW_MENU) {
    mbShowToggle = 0, // toggle the state of the menu
    mbShowExtract = 1, // take down the menu
    mbShowInsert = 2  // put up the menu
};

Comments
If the menu is currently up, and pArgs is mbShowToggle or mbShowExtract, does the following:

• self-sends msgMenuButtonExtractMenu(menu).
• if style.getMenu is true, sends msgMenuButtonMenuDone with the following MENU_BUTTON_PROVIDE_MENU parameters to the menu button’s client:

    menuButton = self;
    menu = menu;

If the menu is currently down, and pArgs is mbShowToggle or mbShowInsert, does the following:

• if style.subMenuType is not mbMenuSibling and the menu has wsLayoutDirty set in its WIN_METRICS.flags.style, sends msgWinLayout with the following WIN_METRICS parameters to the menu:

    options = wsLayoutResize;

• if style.getMenu is true, sends msgMenuButtonProvideMenu with the following MENU_BUTTON_PROVIDE_MENU parameters to the menu button’s client (and then the resulting MENU_BUTTON_PROVIDE_MENU.menu will be used):

    menuButton = self;
    menu = menu;

• if style.enableMenu is true, the process of the selection owner is compared against the process of OSThisApp(). The menu is sent msgControlEnable with the following CONTROL_ENABLE parameters:


**msgMenuButtonPlaceMenu**

Self-sent whenever a menu button needs to position its associated menu.

Takes P_WIN_METRICS, returns STATUS. Category: self-sent.

```c
#define msgMenuButtonPlaceMenu MakeMsg(clsMenuButton, 6)
```

**Comments**

If this message reaches clsMenuButton, that class will do some default positioning. In the message arguments:

- `bounds.origin`: origin of menu *button* wrt theRootWindow
- `bounds.size`: size of menu

Since clsMenuButton uses msgMenuShow to display the menu, and that message always ensures that the menu lies within theRootWindow, there's no need in the method for msgMenuButtonPlaceMenu to check the bounds of the menu against theRootWindow.

**msgMenuButtonProvideMenu**

Sent to the client if style.getMenu is true.

Takes P_MENU_BUTTON_PROVIDE_MENU, returns STATUS. Category: client responsibility.

```c
#define msgMenuButtonProvideMenu MakeMsg(clsMenuButton, 8)
```

**Message**

`typedef struct MENU_BUTTON_PROVIDE_MENU {
    MENU_BUTTON menuButton;  // In: requestor
    WIN menu;  // In/Out: uid of menu
    U32 spare;  // reserved (unused)
} MENU_BUTTON_PROVIDE_MENU, *P_MENU_BUTTON_PROVIDE_MENU;`

**Arguments**

clsMenuButton will send this message to the client of the menu button just before showing the menu. The MENU_BUTTON_PROVIDE_MENU parameters will be set as follows:

- `menuButton`: uid of menu button needing the menu
- `menu`: uid of last provided or set (via msgMenuButtonSetMenu) menu

The client may modify the passed menu or supply a different menu uid. If the client makes changes to the menu that invalidate its layout or supplies a different uid, the client should lay out the menu before returning. If the client changes the uid of the menu, clsMenuButton will self-send msgMenuButtonSetMenu(pArgs->menu) (i.e. the menu button will remember the provided menu for future use). The client will be sent msgMenuButtonMenuDone when the menu button is finished with the menu.
Note that this message can also be sent to a menu button to retrieve the actual menu that would be shown by the menu button. If style.getMenu is true, clsMenuButton will send msgMenuButtonProvideMenu to the menu button's client. In this case, the caller must send msgMenuButtonMenuDone to the menu button when finished with the menu.

msgMenuButtonMenuDone

Sent to the client if style.getMenu is true.

Takes P_MENU_BUTTON_PROVIDE_MENU, returns STATUS. Category: client responsibility.

```c
#define msgMenuButtonMenuDone MsgNoError(MakeMsg(clsMenuButton, 9))
```

- **Message Arguments**
  - typedef struct MENU_BUTTON_PROVIDE_MENU {
    - MENU_BUTTON menuButton; // In: requestor
    - WIN menu; // In/Out: uid of menu
    - U32 spare; // reserved (unused)
  } MENU_BUTTON_PROVIDE_MENU, *P_MENU_BUTTON_PROVIDE_MENU;

- **Comments**
  - clsMenuButton will send this message to the menu button's client when the menu button has taken down the menu and style.getMenu is true. Note that clsMenuButton does remember the uid of the menu even after sending this message. If the client destroys the menu, msgMenuButtonSetMenu(objNull) should be sent to the menu button.
  - If clsMenuButton receives this message, and style.getMenu is true, this message will be forwarded to the menu button's client.

Messages from Other Classes

msgWinLayoutSelf

Tell a window to layout its children.

Takes P_WIN_METRICS, returns STATUS.

- **Comments**
  - If the menu button has a menu, and style.getWidth is true and pArgs->options has wsLayoutResize set and the menu button has wsShrinkWrapWidth on, clsButton self-sends msgMenuButtonProvideWidth to compute pArgs->bounds.size.w.

msgWinSend

Sends a message up a window ancestry chain.

Takes WIN_SEND, returns STATUS.

- **Comments**
  - clsMenuButton acts as the manager for its menu, and looks for msgMenuDone to be sent via msgWinSend.
  - If style.subMenuType is not mbMenuNone and pArgs->msg is msgMenuDone, and the menu is currently up, and pArgs->data[0] is not self, clsMenuButton will do the following:
    - take down the menu as in msgMenuButtonShowMenu.
    - self-send msgButtonSetNoNotify, false to turn off the menu button.
    - ObjectCallAncestor() to all the msgWinSend to continue up the window tree.
If pArgs->data[0] is self, nothing is done and stsOK is returned without calling ancestor. This allows, for example, prevents a menu button with a pull-right menu from taking down the menu containing the menu btnuton.

**msgGWinGesture:**
Called to process the gesture.
Takes P_GWINGesture, returns STATUS.

Comments
If pArgs->msg is xgs2Tap and style.menuAction is mbAction2Tap and style.subMenuType is not mbMenuNone, the menu will be inserted/removed as in msgMenuButtonShow.

clsMenuButton will notify its manager after any gesture.

clsMenuButton self-sends msgButtonNotifyManager with the following BUTTON_NOTIFY parameters:

```plaintext
msg = msgMenuDone;
data = self;
button = self;
```

This is followed by ObjectCallAncestor(), to allow the gesture to be processed normally.

**msgControlSetClient**
Sets the control metrics.client.
Takes UID, returns STATUS.

Comments
clsMenuButton will send msgTkTableSetClient(pArgs) to the menu.
This file contains the API definition for clsMoveCopyIcon.

clsMoveCopyIcon inherits from clsIcon.

Move-copy icons support the move/copy UI. Move-copy icon with drag style mcDragMove will appear with a single marquee. Move-copy icon with drag style mcDragCopy will appear with a double marquee.

```cpp
#ifndef MCICON_INCLUDED
#define MCICON_INCLUDED

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

## Common #defines and typedefs

typedef OBJECT MOVE_COPY_ICON;

drag styles

```cpp
#define mcDragNone 0 // disabled
#define mcDragMove 1 // drag means move
#define mcDragCopy 2 // drag means copy
// 3 // unused (reserved)
typedef struct MOVE_COPY_ICON_STYLE {
  U16 move : 2, // private
copy : 2, // private
drag : 2, // drag behavior
destroyOnSelChange : 1, // destroy self on msgSelChangedOwners
spare : 9; // unused (reserved)
} MOVE_COPY_ICON_STYLE, *P_MOVE_COPY_ICON_STYLE;
tag for clsTrack instances created by clsMoveCopyIcon
#define tagMoveCopyIconTrack MakeTag(clsMoveCopyIcon, 1)
```

## Messages

### msgNew

Creates a move-copy icon window.

Takes P_MOVE_COPY_ICON_NEW, returns STATUS. Category: class message.

```cpp
typedef struct MOVE_COPY_ICON_NEW_ONLY {
  MOVE_COPY_ICON_STYLE style; // overall style
  U32 spare; // unused (reserved)
} MOVE_COPY_ICON_NEW_ONLY, *P_MOVE_COPY_ICON_NEW_ONLY;
#define moveCopyIconNewFields
iconNewFields
MOVE_COPY_ICON_NEW_ONLY moveCopyIcon;
```
typedef struct MOVE_COPY_ICON_NEW {
    moveCopyIconNewFields
} MOVE_COPY_ICON_NEW, *P_MOVE_COPY_ICON_NEW;

If style.drag is not mcDragNone, sets the following:

    pArgs->win.flags.input |= inputMoveDown | inputMoveDelta;
pArgs->win.flags.input &= ~inputLRContinue;
pArgs->gWin.style.gestureEnable = false;
pArgs->border.style.edge = bsEdgeAll;
pArgs->border.style.leftMargin = bsMarginSmall;
pArgs->border.style.rightMargin = bsMarginSmall;
pArgs->border.style.bottomMargin = bsMarginSmall;
pArgs->border.style.topMargin = bsMarginSmall;
pArgs->border.style.borderlnk = bslnkBlack;
pArgs->border.style.LINEStyle =
    (pArgs->moveCopyIcon.style.drag == mcDragMove) ?
        bsLineMarquee : bsLineDoubleMarquee;
pArgs->border.style.selected = true;

Note that if you set style.destroyOnSelChanged to true, you must add the move copy icon as an observer of theSelectionManager to have the move copy icon notified when the selection changes.

msgNewDefaults

Initializes the MOVE_COPY_ICON_NEW structure to default values.

Takes P_MOVE_COPY_ICON_NEW, returns STATUS. Category: class message.

typedef struct MOVE_COPY_ICON_NEW {
    moveCopyIconNewFields
} MOVE_COPY_ICON_NEW, *P_MOVE_COPY_ICON_NEW;

Zeroes out pArgs->moveCopyIcon and sets

    pArgs->moveCopyIcon.style.move = mcMoveCopyEnable;
pArgs->moveCopyIcon.style.copy = mcMoveCopyEnable;

Default MOVE_COPY_ICON_STYLE:

    drag = mcDragNone
    destroyOnSelChange = false

msgMoveCopyIconGetStyle

Passes back the current style values.

Takes P_MOVE_COPY_ICON_STYLE, returns STATUS.

#define msgMoveCopyIconGetStyle MakeMsg(clsMoveCopyIcon, 1)

typedef struct MOVE_COPY_ICON_STYLE {
    U16 move : 2, // private
    copy : 2,     // private
    drag : 2,     // drag behavior
    destroyOnSelChange : 1, // destroy self on msgSelChangedOwners
    spare : 9,    // unused (reserved)
} MOVE_COPY_ICON_STYLE, *P_MOVE_COPY_ICON_STYLE;
**Message from other classes**

### msgMoveCopyIconSetStyle

Sets the style values.

Takes P_MOVE_COPY_ICON_STYLE, returns STATUS.

```c
#define msgMoveCopyIconSetStyle MakeMsg(clsMoveCopyIcon, 2)
```

**Arguments**

- `move` (U16): // private
- `copy` (U16): // private
- `drag` (U16): // drag behavior
- `destroyOnSelChange` (U16): // destroy self on msgSelChangedOwners
- `spare` (U16): // unused (reserved)

**Comments**

Note that changing style.drag is not implemented.

### msgMoveCopyIconDone

Sent to the control.client when the icon completes move or copy mode.

Takes P_MOVE_COPY_ICON_DONE, returns STATUS. Category: client notification.

```c
#define msgMoveCopyIconDone MakeMsg(clsMoveCopyIcon, 6)
```

**Arguments**

- `icon` (WIN): // icon sending msg
- `move` (BOOLEAN): // true for Move, false for Copy
- `dest` (WIN): // destination window to move/copy to
- `destXY` (XY32): // point to move/copy to in dest space
- `penOffset` (XY32): // offset of pen from icon origin (grab point)
- `iconSize` (SIZE32): // unused (reserved)
- `spare1` (U32): // unused (reserved)
- `spare2` (U32): // unused (reserved)
- `spare3` (U32): // unused (reserved)

**Comments**

If the client responds with stsRequestDenied, stsMessageIgnored, or a status < stsOK, the `moveCopyIcon` will be jumped to pArgs->rect.origin and the single or double marquee will be restarted. Otherwise, msgDestroy will be self-sent.

### msgMoveCopyIconCancel

Sent to the control.client when the icon cancels move or copy mode.

Takes OBJECT, returns STATUS. Category: client notification.

```c
#define msgMoveCopyIconCancel MakeMsg(clsMoveCopyIcon, 5)
```

**Comments**

clsMoveCopyIcon will send self as pArgs. This is sent when style.destroyOnSelChange is true, and msgSelChangedOwners is received.

### msgInputEvent

Notification of an input event.

Takes P_INPUT_EVENT, returns STATUS.

**Comments**

If style.drag is not mcDragNone, clsMoveCopyIcon responds as follows:
If `pArgs->devCode` is `msgPenMoveDown` and the pen has moved beyond defined threshold, or
`pArgs->devCode` is `msgPenExitDown`, an instance of `msgTrackProvideMetrics` will be created to indicate the move/copy.

If `pArgs->devCode` is `msgPenUp`, and `msgPenDown` has already been seen, it is sent to the client.

**msgTrackProvideMetrics**

Sent to a track client before track is created.


Comments

If `pArgs->tag` is `tagMoveCopyIconTrack`, `msgTrackProvideMetrics(pArgs)` is sent to the `moveCopyIcon`'s client.

**msgTrackDone**

Sent by a tracker when it's done.

Takes `P_TRACK_METRICS`, returns `STATUS`. Category: client notification.

```c
#define msgTrackDone MakeMsg(clsTrack, 6)
```

Comments

`clsMoveCopyIcon` will hit-detect `pArgs->curXY` to locate the window over which the track was dropped. The client will be sent `msgMoveCopyIconDone` with the following `MOVE_COPY_ICON_DONE` parameters:

- `icon = self;`
- `move = true for move, false for copy;`
- `dest = destination window;`
- `destXY = pArgs->curXY in dest window's space;`
- `penOffset = pArgs->curXY in pArgs->rect-relative space;`

**msgSelChangedOwners**

Notify the observers when either of the selection owners have changed.

Takes `P_SEL_OWNERS`, returns `STATUS`.

Comments

If `style.destroyOnSelChange` is true, `clsMoveCopyIcon` will send `msgMoveCopyIconCancel(self)` to its client followed by `msgDestroy` to self.
This file contains the API definition for clsMenu.

clsMenu inherits from clsTkTable.

Menus are collections of menu buttons (each of the latter may have a submenu associated with it, which in turn is a collection of menu buttons...).

```c
#ifndef MENU_INCLUDED
#define MENU_INCLUDED

#include <tktable.h>

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

**Common #defines and typedefs**

```c
typedef OBJECT MENU, *P_MENU;
```

**Menu Type Styles**

```c
#define msTypeMenuBar 0 // horizontal menu bar
#define msTypeMenu 1 // pull-down or pull-right
// 2 // unused (reserved)
// 3 // unused (reserved)

typedef struct MENU_STYLE {
  U16 type : 2; // menu type
  spare : 14; // unused (reserved)
} MENU_STYLE, *P_MENU_STYLE;
```

**Messages**

**msgNew**

Creates a menu window, together with the child windows specified in pEntries.

Takes P_MENU_NEW, returns STATUS. Category: class message.

Arguments

```c
typedef struct MENU_NEW_ONLY {
  MENU_STYLE style; // overall style
  MENU_BUTTON_NEW menuButtonNew; // storage for default child new struct
  U32 spare; // unused (reserved)
} MENU_NEW_ONLY, *P_MENU_NEW_ONLY;
```

```c
#define menuNewFields
  tkTableNewFields
  MENU_NEW_ONLY menu;
```

```c
typedef struct MENU_NEW {
  menuNewFields
} MENU_NEW, *P_MENU_NEW;
```

Comments

If pArgs->menu.style.type is msTypeMenu, the following is done before ObjectCallAncestor():
pArgs->win.flags.style |= wsSaveUnder;
pArgs->win.flags.style |= wsClipSiblings;
pArgs->win.flags.style &= ~(U32)wsParentClip;

pArgs->border.style.shadow = bsShadowThinBlack;
pArgs->border.style.shadowGap = bsGapTransparent;

pArgs->border.style.leftMargin = bsMarginMedium;
pArgs->border.style.rightMargin = bsMarginMedium;
pArgs->border.style.bottomMargin = bsMarginMedium;
pArgs->border.style.topMargin = bsMarginMedium;

pArgs->tableLayout.style.growChildWidth = true;
pArgs->tableLayout.style.wrap = false;

pArgs->tableLayout.numRows.constraint = tlInfinite;
pArgs->tableLayout.numCols.constraint = tlAbsolute;
pArgs->tableLayout.numCols.value = 1;
pArgs->tableLayout.rowHeight.constraint = tlChildrenMax;

---

msgNewDefaults

Initializes the MENU_NEW structure to default values.

Takes P_MENU_NEW, returns STATUS. Category: class message.

typedef struct MENU_NEW {
  menuNewFields
} MENU_NEW, *P_MENU_NEW;

Zeroes out pArgs->menu and sets

pArgs->win.flags.style |= wsFileNoBounds;
pArgs->embeddedWin.style.selection = ewSelectPreserve;
pArgs->gWin.style.gestureEnable = false;
pArgs->border.style.edge = bsEdgeAll;
pArgs->border.style.leftMargin = bsMarginMedium;
pArgs->border.style.rightMargin = bsMarginMedium;
pArgs->border.style.bottomMargin = bsMarginSmall;
pArgs->border.style.topMargin = bsMarginMedium;

// layout for MSTypeMenuBar
pArgs->tableLayout.style.growChildWidth = false;
pArgs->tableLayout.style.growChildHeight = false;
pArgs->tableLayout.style.wrap = true;

pArgs->tableLayout.colWidth.gap = defaultColGap;
pArgs->tableLayout.rowHeight.constraint = tlGroupMax;
pArgs->tableLayout.rowHeight.gap = defaultRowGap;
pArgs->menu.style.type = msTypeMenuBar;

The menu is a table of clsMenuButton buttons, so pArgs->tkTable.pButtonNew is set to the address of pArgs->menu.menuButtonNew. This menuButtonNew is initialized using msgNewDefaults to clsMenuButton, then altered as in msgTkTableChildDefaults. See msgTkTableChildDefaults for more info.

Default Menu style:

type = msTypeMenuBar
### msgMenuGetStyle

Passes back the current style values.

Takes P_MENU_STYLE, returns STATUS.

```c
#define msgMenuGetStyle MakeMsg(clsMenu, 4)
```

**Message**

typedef struct MENU_STYLE {
    U16 type : 2, // menu type
    spare : 14; // unused (reserved)
} MENU_STYLE, *P_MENU_STYLE;

**Comments**

Note: setting style.type is not implemented.

### msgMenuSetStyle

Sets the style values.

Takes P_MENU_STYLE, returns STATUS.

```c
#define msgMenuSetStyle MakeMsg(clsMenu, 5)
```

**Message**

typedef struct MENU_STYLE {
    U16 type : 2, // menu type
    spare : 14; // unused (reserved)
} MENU_STYLE, *P_MENU_STYLE;

**Comments**

### msgMenuShow

Puts up or takes down the menu by inserting or extracting it as a child of the RootWindow.

Takes BOOLEAN, returns STATUS.

```c
#define msgMenuShow MakeMsg(clsMenu, 1)
```

**Comments**

To show the menu, first delta the menu to the desired position, in root window space and use pArgs of true. To hide the menu, use pArgs of false.

Before showing the menu, the menu's origin is altered as follows (in this order):

- If the menu is wider or taller than the RootWindow, the menu will be placed in an instance of clsScrollWin to allow the user to scroll through the menu contents.
- If the menu falls off the right edge of the root window, the menu is right-justified.
- If the menu falls off the left edge of the root window, the menu is left-justified.
- If the menu falls below the bottom edge of the root window, the menu is bottom-justified.
- If the menu falls above the top edge of the root window, the menu is top-justified.

The menu will insert itself as an input filter when shown, and remove itself when hidden. The menu will be extracted from the root window when hidden.

### msgMenuDone

Sent via msgWinSend to the manager when the menu is "done".

Takes WIN, returns STATUS. Category: manager notification.

```c
#define msgMenuDone MakeMsg(clsMenu, 2)
```

**Comments**

The manager should use msgMenuShow to take down the menu. See msgWinSend for clsMenu's response to msgMenuDone via msgWinSend.
msgMenuAdjustSections
Adjusts the border edges and margins of children to correctly reflect a sectioned menu.
Takes BOOLEAN, returns STATUS.
#define msgMenuAdjustSections MakeMsg(clsMenu, 3)
Comments
This message is provided for compatibility and results in a self-send of msgTblLayoutAdjustSections. New clients should use msgTblLayoutAdjustSections directly.
See Also
msgTblLayoutAdjustSections

Messages from other classes

msgTkTableChildDefaults
Sets the defaults in pArgs for a common child.
Takes P_UNKNOWN, returns STATUS.
clsMenu sets up defaults for each child as follows:

pArgs->win.flags.style |= wsParentClip;
pArgs->win.flags.style &= - (U32)(wsClipSiblings|wsClipChildren);
If the child is a descendant of clsBorder, then
pArgs->border.style.backgroundlnk |= bsInkExclusive;
If the child is a descendant of clsButton, then
pArgs->button.style.manager = bsManagerParent;

msgInputEvent
Notification of an input event.
Takes P_INPUT_EVENT, returns STATUS.
clsMenu receives input events as a result of the InputFilterAdd() done during msgMenuShow. The events are handled as follows:

• If pArgs->destination is self, stsInputSkip returned.
• If pArgs->destination is a descendant of self (i.e. in the menu's window tree), the event is passed through to the destination by returning stsInputSkip.
• If the pArgs->devCode is msgPenDown, clsMenu will ObjectCallAncestor() msgWinSend with the following WIN_SEND parameters:

msg = msgMenuDone;
data[0] = pArgs->destination;
flags = wsSendDefault;
lenSend = SizeOf(WIN_SEND);

This is intended as a notification to the menu's manager that the menu is ready to be taken down. If pArgs->destination is a descendant of clsMenuButton, stsInputContinue is returned to allow the input event to continue; otherwise, the event is terminated by returning stsInputTerminateRemoveStroke.
• All other input events result in a return status of stsInputContinue.
msgWinSend

Sends a message up a window ancestry chain.

Takes WIN_SEND, returns STATUS.

Comments

clsMenu looks for manager notifications of msgMenuDone or msgButtonDone via msgWinSend.

If pArgs->msg is msgMenuDone and pArgs->data[0] is a descendant of self, clsMenu will return stsOK.
This prevents self's manager from receiving the msgMenuDone and taking down the menu. This prevents, for example, a pull-right menu coming down from taking down its main menu.

If pArgs->msg is msgButtonDown, pArgs->msg is replaced with msgMenuDone before calling ObjectCallAncestor(). This results, for example, in the menu coming down when a button in the menu is hit.

All other values of pArgs->msg result in ObjectCallAncestor().
This file contains the API for clsModalFilter. clsModalFilter inherits from clsObject.

Modal filters implement window-relative input modality. Modal filters are useful for making a window tree behave in a modal fashion: the user must interact with the windows in the tree (and make it go away) before they can use other windows in the application (or system).

Here is an example of how to set up a modal filter object:

```c
M O D A L _ F I L T E R _ N E W  m f n ;

// Create it.
ObjCallWarn(msgNewDefaults, clsModalFilter, &mfn);

mfn.modalFilter.flags = <appropriate flags>;

mfn.modalFilter.process = OSThisProcess();

mfn.modalFilter.subTreeRoot = <root of window tree to make modal>;
ObjCallRet(msgNew, clsModalFilter, &mfn, s);

// Activate it.
ObjCallRet(msgModalFilterActivate, mfn.object.uid, pNull, s);

// Tell input system about it.
StsRet(InputFilterAdd( \
    mfn.object.uid, inputAllRealEventsFlags, 0, <priority>), s);

```

See input.h for a discussion of filter priorities and tips on choosing a priority.

### Debugging Flags

The clsModalFilter debugging flag is 'K'. Defined values are:

flag10 (0x0400) general

```c
#ifndef MFILTER_INCLUDED
#define MFILTER_INCLUDED
#include <clsmgr.h>
#include <ostypes.h>
#include <win.h>
#endif
```

```c
#define mfSystemModal
#define mfAutoDismiss
#define mfDefaultFlags
```

### Common #defines and typedefs

```c
typedef OBJECT MODAL_FILTER, *P_MODAL_FILTER;

// Flags
#define mfSystemModal flag0
#define mfAutoDismiss flag1
#define mfDefaultFlags mfAutoDismiss
```
typedef struct MODAL_FILTER_METRICS {
    U16 flags;
    OS_TASK_ID process; // app process for filter; ignored for mfSystemModal
    WIN subTreeRoot; // window tree to which to give events
    U32 spare; // reserved
} MODAL_FILTER_METRICS, *P_MODAL_FILTER_METRICS;

**msgNew**

Creates a modal filter.

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

```c
typedef MODAL_FILTER_METRICS MODAL_FILTER_NEW, *P_MODAL_FILTER_NEW;
#define modalFilterNewFields
    objectNewFields
    MODAL_FILTER_NEW_ONLY modalFilter;
```

**Arguments**

typedef struct MODAL_FILTER_NEW {
    modalFilterNewFields
} MODAL_FILTER_NEW, *P_MODAL_FILTER_NEW;

**Comments**

The fields you commonly set are:

- `pArgs->modalFilter.flags` appropriate flags
- `pArgs->modalFilter.process` process owning the window tree
- `pArgs->modalFilter.subTreeRoot` root of window tree for which to filter

A filter is active after `msgNew`, and becomes deactivated only after it has dismissed its window.

**msgNewDefaults**

Initializes the `MODAL_FILTER_NEW` structure to default values.

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

```c
typedef struct MODAL_FILTER_NEW {
    modalFilterNewFields
} MODAL_FILTER_NEW, *P_MODAL_FILTER_NEW;
```

**Arguments**

Zeroes out `pArgs->modalFilter` and sets:

- `pArgs->modalFilter.flags = mfDefaultFlags`;

**msgModalFilterGetFlags**

Passes back the receiver's flags.

Takes `P_U16`, returns `STATUS`.

```c
#define msgModalFilterGetFlags MakeMsg(clsModalFilter, 1)
```

**msgModalFilterSetFlags**

Sets the receiver's flags.

Takes `P_U16`, returns `STATUS`.

```c
#define msgModalFilterSetFlags MakeMsg(clsModalFilter, 2)
```
msgModalFilterActivate
Activates the filter.
Takes nothing, returns STATUS.
#define msgModalFilterActivate MakeMsg(clsModalFilter, 3)
Comments
A filter is active after msgNew, and becomes deactivated only after it has dismissed its window.

msgModalFilterDeactivate
Deactivates the filter.
Takes nothing, returns STATUS.
#define msgModalFilterDeactivate MakeMsg(clsModalFilter, 4)
Comments
A filter is active after msgNew, and becomes deactivated only after it has dismissed its window.

msgModalFilterDismissWin
Sent to the subTreeRoot if the win should be dismissed.
Takes nothing, returns STATUS. Category: third-party notification.
#define msgModalFilterDismissWin MakeMsg(clsModalFilter, 5)

Messages from Other Classes

msgInputEvent
Notification of an input event.
Takes P_INPUT_EVENT, returns STATUS.
Comments
If the filter is inactive, or the input event’s devCode is not of clsPen, or the evfGrabTracker flag is set in pArgs->flags, or there’s a grabber object present (InputGetGrab), then the filter just returns stsInputContinue.

Next, if the pArgs->destination is not a valid object, the filter returns stsInputTerminate.

If, at this point, the mfSystemModal flag is clear and the process of the pArgs->destination doesn’t match MODAL_FILTER_METRICS.process, the filter does the following:

    if mfAutoDismiss is on
        if the pArgs->devCode is msgPenDown
            self-send msgModalFilterDeactivate
            send msgModalFilterDismissWin to MODAL_FILTER_METRICS.subTreeRoot
            (and if that returns an error status, top and flash subTreeRoot)
            return stsInputTerminate.
            otherwise return stsInputContinue.
        otherwise return stsInputContinue.

Now, if pArgs->destination is within subTreeRoot, return stsInputSkipTo4. (See input.h)

Next, if the subTreeRoot is not a valid object, return stsFailed.

Next, if mfAutoDismiss is on and pArgs->devCode is msgPenDown:

    self-send msgModalFilterDeactivate
    send msgModalFilterDismissWin to MODAL_FILTER_METRICS.subTreeRoot
    (and if that returns an error status, top and flash subTreeRoot)
    return stsInputTerminate.
Finally, if pArgs->devCode is msgPenDown, the filter tops the subTreeRoot, flashes it, and returns
stsInputTerminate.

Return Value
stsInputContinue

See Also
msgWinInsert used by a filter to top the subTreeRoot
msgBorderFlash used by a filter to flash the subTreeRoot
NOTE.H

This file contains the API for clsNote.
clsNote inherits from clsFrame.
Provides the UI for system- and app-modal messages to the user.

⚠️ Debugging Flags

The clsNote debugging flag is 'K'. Defined values are:
flag15 (0x8000) general
ifndef NOTE_INCLUDED
#define NOTE_INCLUDED
#include <clsmgr.h>
ifndef CLSMGR_INCLUDED
#endif
ifndef WIN_INCLUDED
#include <win.h>
#endif
ifndef FRAME_INCLUDED
#include <frame.h>
#endif
ifndef TKTABLE_INCLUDED
#include <tktable.h>
#endif

ifndef NOTE_INCLUDED
#define NOTE_INCLUDED
#endif
ifndef CLSMGR_INCLUDED
#endif
ifndef WIN_INCLUDED
#endif
ifndef FRAME_INCLUDED
#endif
ifndef TKTABLE_INCLUDED
#endif
#endif

Common #defines and typedefs

typedef WIN NOTE, *P_NOTE;
// Tags of component windows within a note.
define tagNoteTitle MakeTag(clsNote, 1)
define tagNoteTkTable MakeTag(clsNote, 2)
define tagNoteCmdBar MakeTag(clsNote, 3)
// Note flags
define nfSystemModal flag0
define nfAutoDestroy flag1
define nfSystemTitle flag2 // use system title; ignore pTitle
define nfAppTitle flag3 // use app title; ignore pTitle
define nfUnformattedTitle flag9 // use pTitle as is (not "Note from <pTitle>")
define nftimeout flag4 // dismiss on timeout or input
#define nfNoWordWrap flag5 // don't word wrap content labels
#define nfResContent flag6 // pContentEntries is P_NOTE_RES_ID
#define nfNoBeep flag7 // disable prefs-controlled beeping
#define nfExplicitCancel flag8 // note will ignore cmdBar buttons
#define nfDefaultSysFlags
(nfSystemModal | nfAutoDestroy | nfSystemTitle | nfNoBeep)
define nfDefaultAppFlags (nfAppTitle | nfNoBeep)
define nfDefaultFlags nfDefaultSysFlags

typedef struct NOTE_METRICS {
U16 flags; // looks and filter flags
MESSAGE autoDismissMsg; // returned iff win dismissed
OBJECT modalFilter; // filter or objNull for default
OS_MILLISECONDS timeout; // timeout or 0 for user pref
OBJECT client; // client for msgNoteDone
U32 spare; // reserved
} NOTE_METRICS, *P_NOTE_METRICS;
**msgNew**

Creates a note.

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

**Arguments**

typedef struct {
    NOTE_METRICS metrics;
    P_CHAR pTitle;
    P_UNKNOWN pContentEntries; // used to create the content
    P_TK_TABLE_ENTRY pCmdBarEntries; // used to create the command bar
    U32 spare; // reserved
} NOTE_NEW ONLY, *P_NOTE_NEW ONLY;

If `nfSystemModal` is on, then the client is ignored. If `nfSystemModal` is off, then `msgNoteShow` returns immediately, and the client will be sent `msgNoteDone` when the note is dismissed.

If `pTitle` will be used (`nfSystemTitle` and `nfAppTitle` are off), the title will appear as follows:

"Note from `<pTitle>` ..." if `nfUnformattedTitle` is off

"<pTitle>" if `nfUnformattedTitle` is on

```c
#define noteNewFields
#define frameNewFields
#define NOTE_NEW ONLY note;
```

typedef struct NOTE_NEW {
    noteNewFields
} NOTE_NEW, *P_NOTE_NEW;

typedef struct {
    RES_ID resId; // resId for a string table resource
    U32 index; // index within that table of a string
    U32 spare; // reserved (unused)
} NOTE_RES_IO, *P_NOTE_RES_IO;

clsNote will use `msgResReadData` to read the string from either `OSThisAppO's APP_METRICS.resList`, or `theSystemResFile` if `OSThisAppO` returns `objNull`.

Since `clsNote` will make a label from the string and `clsLabel` will break word-wrapped labels at newlines (`'
'`), you may embed newlines in the string to force line breaks.

The fields you commonly set are:

- `pArgs->note.flags` appropriate flags
- `pArgs->note.autoDismissMsg` arg for `msgNoteCancel`
- `pArgs->note.timeout` timeout if desired
- `pArgs->note.client` client if app-modal

`clsNote` will create all the appropriate interior windows, then self-send `msgWinLayout` to size and place all the windows. After that, if either the x or y of the note's origin is 0, `clsNote` will delta the new instance so that when it is inserted as a child of `theRootWindow` the note will appear in a reasonable location.

To display and activate the note, use `msgNoteShow`. 
msgNewDefaults
Initializes the NOTE_NEW structure to default values.
Takes P_NOTE_NEW, returns STATUS. Category: class message.

typedef struct NOTE_NEW {
    noteNewFields
} NOTE_NEW, *P_NOTE_NEW;

Zeroes out pArgs->note and sets:

pArgs->win.flags.style
   |= wsSaveUnder | wsShrinkWrapWidth | wsShrinkWrapHeight;
pArgs->border.style.resize = false;
pArgs->border.style.drag = bsDragNone;
pArgs->customLayout.style.limitToRootWin = true;
pArgs->frame.style.closeBox = false;
pArgs->frame.style.zoomable = false;
pArgs->note.metrics.flags = nfDefaultFlags;

msgNoteGetMetrics
Get the metrics of a note.
Takes P_NOTE_METRICS, returns STATUS.

#define msgNoteGetMetrics MakeMsg(clsNote, 1)

typedef struct NOTE_METRICS {
    U16 flags;       // looks and filter flags
    MESSAGE autoDismissMsg; // returned iff win dismissed
    OBJECT modalFilter; // filter or objNull for default
    OS_MILLISECONDS timeout; // timeout or 0 for user pref
    OBJECT client;   // client for msgNoteDone
    U32 spare;       // reserved
} NOTE_METRICS, *P_NOTE_METRICS;

msgNoteSetMetrics
Set the metrics of a note.
Takes P_NOTE_METRICS, returns STATUS.

#define msgNoteSetMetrics MakeMsg(clsNote, 2)

typedef struct NOTE_METRICS {
    U16 flags;       // looks and filter flags
    MESSAGE autoDismissMsg; // returned iff win dismissed
    OBJECT modalFilter; // filter or objNull for default
    OS_MILLISECONDS timeout; // timeout or 0 for user pref
    OBJECT client;   // client for msgNoteDone
    U32 spare;       // reserved
} NOTE_METRICS, *P_NOTE_METRICS;

clsNote will destroy any previous filter object if the filter is changed.

msgNoteShow
Displays a note.
Takes P_MESSAGE, returns STATUS.

#define msgNoteShow MakeMsg(clsNote, 3)
PENPOINT API REFERENCE
Part 4 / UI Toolkit

Comments
If $nfSystemModal$ is on, then the send of this message will block until the note is dismissed. At that
time, $msgNoteShow$ will set $pArgs$ to the message sent by the button that was hit (or $autoDismissMsg$
if the win was dismissed by its modal filter). Be aware that the entire input system (and therefore the
window system) will be blocked while $msgNoteShow$ is waiting for completion.

If $nfSystemModal$ is off, then $msgNoteShow$ returns immediately. It is the app's responsibility to
implement whatever notion of "modality" is appropriate. Usually this means remembering that the app
should be "modal" and waiting for $msgNoteDone$ to be sent to the note's client (which should usually
be the app object). Although the note will filter all the input to the app and discard that input not
directed at the note, the app must still respond to messages from the app framework. When
$nfSystemModal$ is off, the $pArgs$ to $msgNoteShow$ is not set.

$msgNoteDone$
This is the message sent to clients when a note is dismissed.
Takes MESSAGE, returns STATUS.
#define $msgNoteDone$ MakeMsg(clsNote, 4)

Comments
$msgNoteDone$ is only sent if $nfSystemModal$ is off.
The parameter message is the message sent by the button that was hit (or $autoDismissMsg$ if the win
was dismissed by its modal filter).

$msgNoteCancel$
Informs a note that it should take itself down.
Takes P.MESSAGE, returns STATUS.
#define $msgNoteCancel$ MakeMsg(clsNote, 5)

Comments
This will be posted to a note when:
* it receives $msgButtonNotify$ from its command bar, or
* it receives $msgModalFilterDismissWin$ from its filter.
The method code will do all the final cleanup, including extracting the note window (and destroying it
if $nfAutoDestroy$ was on). The $pArgs$ message will either be returned to the original code that called
$msgNoteShow$ (if $nfSystemModal$ is on), or passed to $msgNoteDone$ (if $nfSystemModal$ is off).
This message is only interesting to subclasses of clsNote. It should not be used by normal clients.

Messages from Other Classes

$msgFree$
Sent as the last of three msgs to destroy an object.
Takes OBJ_KEY, returns STATUS.

Comments
clsNote will use InputFilterRemoveO to take its filter out of the input system's list of filters if the filter is
active. clsNote will then send $msgDestroy$ to its filter if the note had created it (as opposed to the client
passing in a filter).
**msgRestore**
Creates and restores an object from an object file.
Takes P_OBJ_RESTORE, returns STATUS.
Comments clsNote will restore its flags, autoDismissMsg, and timeout.

**msgSave**
Causes an object to file itself in an object file.
Takes P_OBJ_SAVE, returns STATUS.
Comments clsNote will file its flags, autoDismissMsg, and timeout. It will not file its modalFilter or client.

**msgWinSend**
Sends a message up a window ancestry chain.
Takes P_WIN_SEND, returns STATUS.
Comments The note may respond by posting itself msgNoteCancel (passing a pointer to its autoDismissMsg), depending on the pArgs->msg and the nfExplicitCancel flag.

**msgWinLayoutSelf**
Tells a window to layout its children (sent during layout).
Takes P_WIN_METRICS, returns STATUS.
Comments If wsLayoutResize is on and nfNoWordWrap is off and the note is shrinkwrapping in width, the note might further adjust the results of the default layout (obtained by just calling ancestor). The note's width will be forced wider if the height of the initial layout is taller than dictated by the 'golden section' ratio of h/w = 0.618.

**msgGWinGesture**
Self-sent to process the gesture.
Takes P_GWIN_GESTURE, returns STATUS.
Comments clsNote will just return the result of calling its ancestor if the note has buttons (i.e., NOTE_NEW_ONLY had a non-null pCmdBarEntries).
Otherwise, the note will post itself msgNoteCancel, passing a pointer to its autoDismissMsg. Although clsNote should check the nfExplicitCancel flag, it does not yet do so for msgGWinGesture (although this may change in the future).
See Also msgNoteCancel tells a note to take itself down.

**msgModalFilterDismissWin**
Sent to the subTreeRoot if the win should be dismissed.
Takes nothing, returns STATUS. Category: third-party notification.
Comments The note will respond by posting itself msgNoteCancel, passing a pointer to its autoDismissMsg.
**msgTimerNotify**

Notifies the client that the timer request has elapsed.

Takes P_TIMER_NOTIFY, returns nothing. Category: advisory message.

**Comments**

A note may receive this when a non-zero NOTE_METRICS.timeout was specified and the note was displayed via msgNoteShow. If this msgTimerNotify does indeed signify that the note should take itself down, the note will do so by posting itself msgNoteCancel (passing a pointer to its autoDismissMsg).
This file contains the API for clsOption.

clsOption inherits from clsFrame.

Provides the standard looks, behavior, and protocol of option sheets.

An option sheet is a special kind of frame that you can use to display the properties of a selected object. If the selected object has several different sets of properties, then the option sheet will have several windows stacked in it like a deck of cards. Each of these windows is called an option card. For more information on option cards, please see clsOptionTable (in opttable.h).

The user navigates between the option cards with a popup choice, which is available on the title line of the option sheet. The popup choice contains a clsTabButton for each option card. The typical PenPoint developer does not need to know about how option sheets use clsTabButton, but feel free to take a look at it (in tbutton.h).

Although clsOption provides a rich API, most PenPoint developers need to understand only the following:

Messages sent by a client to an option sheet:
- msgOptionAddCard
- msgOptionAddLastCard

Messages sent to a sheet's client by an option sheet:
- msgOptionClosed
- msgOptionProvideTopCard

Messages sent to a card's client by an option sheet:
- msgOptionProvideCardWin
- msgOptionApplyCard
- msgOptionRefreshCard
- msgOptionApplicableCard

Messages self-sent by a client to create an option sheet:
- msgOptionCreateSheet
- msgOptionAddCards
Debugging Flags

The clsOption debugging flag is '%'. Defined values are:

flag8 (0x0100) general

```c
#ifndef OPTION_INCLUDED
#define OPTION_INCLUDED
#include <frame.h>
#endif
#include <tktable.h>
#endif
```

Common #defines and typedefs

```c
#define tagOptionApplyButton
#define tagOptionApplyAndCloseButton
#define tagOptionCloseButton
#define hlpOptionApplyButton
#define hlpOptionApplyAndCloseButton
#define hlpOptionCloseButton
typedef OBJECT OPTION;
```

Sheet Modality Style

The sheet modality style specifies whether the card is modal, and if so, whether system-modal or application-modal.

```c
#define osModalNone 0
#define osModalApp 1
#define osModalSystem 2
```

Card Navigation Style

The card navigation style specifies how the user can move between option cards. GO recommends that you use a popup choice.

```c
#define osNavPopup 0 // popup choice in the title bar
#define osNavTabBar 1 // tab buttons in the tab bar
// 2 // unused (reserved)
// 3 // unused (reserved)
typedef struct OPTION_STYLE {
  U16 senseSelection : 1, // observe theSelectionManager
  modality : 2, // whether modal, and what type
  cardNav : 2, // card navigation style
  getCards : 1, // true => enable msgOptionAddCards protocol
  needCards : 1, // true => current list of cards is invalid
  needTopCard : 1, // true => current top card is invalid
  hideNav : 1, // true => hide card navigation
  spare1 : 7; // unused (reserved)
  U16 spare2 : 16; // unused (reserved)
} OPTION_STYLE, *P_OPTION_STYLE;
```

Default OPTION_STYLE:

```c
senseSelection = true
modality = osModalNone
cardNav = osNavPopup
getCards = false
needCards = true
needTopCard = true
hideNav = false
```
typedef struct OPTION_CARD {
   OPTION option;    // out: option sheet sending the msg.
   U32 tag;          // in: tag for tab
   WIN win;          // in: card window or objNull
   P_CHAR pName;     // in: card name
   U16 nameLen;      // in: max. len for pName (for msgOptionGetCardAndName)
   OBJECT client;    // in: for msgOptionRefreshCard, etc.
   U32 clientData[2]; // in: arbitrary client data
   U32 spare1;       // unused (reserved)
   U32 spare2;       // unused (reserved)
} OPTION_CARD, *P_OPTION_CARD;

typedef struct OPTION_TAG {
   OPTION option;    // out: option sheet sending the msg.
   TAG tag;          // in: tag for tab
   OPTION_TAG, *P_OPTION_TAG;
}

Messages

msgNew

Creates an option sheet.

Takes P_OPTION_NEW, returns STATUS. Category: class message.

Arguments
typedef struct OPTION_NEW_ONLY {
   OPTION_STYLE style;    // overall style
   PTK_TABLE_ENTRY pCmdBarEntries; // optional override
   U32 spare1;       // unused (reserved)
   U32 spare2;       // unused (reserved)
} OPTION_NEW_ONLY, *P_OPTION_NEW_ONLY;

If pCmdBarEntries is not null, then it should be the address of a null-terminated array of entries. It is
used to create a custom command bar rather than the usual Apply and Apply&Close buttons. The client
of this custom command bar is set to the frame's client.

#define optionNewFields
   frameNewFields
   OPTION_NEW_ONLY option;

typedef struct OPTION_NEW {
   optionNewFields
} OPTION_NEW, *P_OPTION_NEW;

Comments

If pArgs->option.style.cardNav is osNavPopup, clsOption will create an instance of clsTkTable with a
label and a popupChoice in it as the frame's title bar. The label string will be set to the frame's title
string. The popup choice will contain a choice for each card in the option sheet.

msgNewDefaults

Initializes the OPTION_NEW structure to default values.

Takes P_OPTION_NEW, returns STATUS. Category: class message.

Arguments
typedef struct OPTION_NEW {
   optionNewFields
} OPTION_NEW, *P_OPTION_NEW;

Zeroes out pArgs->option and sets

   pArgs->win.flags.style |= wsSendGeometry | wsSaveUnder;
   pArgs->embeddedWin.style.selection = ewSelectPreserve;
**msgSave**
Causes an object to file itself in an object file.

Takes P_OBJ_SAVE, returns STATUS.

**Comments**
The option sheet saves its style and the tag of the current top card. This tag is used as the default value for the top card when msgOptionProvideTopCard is next sent (e.g., after the option sheet is restored and inserted in the window tree).

Saving an option sheet causes msgSave to be sent to each of the option card’s tab buttons. If a card’s client is OSThisApp(), its tab button records and saves this fact. Otherwise, the client is not saved.

**msgRestore**
Creates and restores an object from an object file.

Takes P_OBJ_RESTORE, returns STATUS.

**Comments**
The option sheet restores its instance data and sets the following:
```
style.needTopCard = true; needCards = true;
```

If the restored frame has a command bar, msgTkTableSetClient is sent to it to force its client to be the option sheet.

If style.getCards and style.senseSelection are true, the option sheet is set up to observe theSelectionManager.

Restoring an option sheet causes msgRestore to be sent to each of the option card’s tab buttons. If a card’s client was OSThisApp(), its tab button sets the client to the new value for OSThisApp(). Other cards have their client set to objNull.

**msgOptionGetStyle**
Passes back the style of the option sheet.

Takes P_OPTION_STYLE, returns STATUS.

```c
#define msgOptionGetStyle MakeMsg(clsOption, 1)
typedef struct OPTION_STYLE {
    U16 senseSelection : 1,  // observe theSelectionManager
    modality : 2,            // whether modal, and what type
    cardNav : 2,             // card navigation style
    getCards : 1,            // true => enable msgOptionAddCards protocol
    needCards : 1,           // true => current list of cards is invalid
    needTopCard : 1,         // true => current top card is invalid
    hideNav : 1,             // true => hide card navigation
    spare1 : 7;              // unused (reserved)
    spare2 : 16;             // unused (reserved)
} OPTION_STYLE, *P_OPTION_STYLE;
```
Most clients do not need to deal with this message.

**msgOptionSetStyle**
Sets the style of the option sheet.
Takes `P_OPTION_STYLE`, returns `STATUS`.

```c
#define msgOptionSetStyle MakeMsg(clsOption, 2)
```

**Message**
```c
typedef struct OPTION_STYLE {  
  U16 senseSelection : 1; // observe theSelectionManager  
  modality : 2; // whether modal, and what type  
  cardNav : 2; // card navigation style  
  getCards : 1; // true => enable msgOptionAddCards protocol  
  needCards : 1; // true => current list of cards is invalid  
  needTopCard : 1; // true => current top card is invalid  
  hideNav : 1; // true => hide card navigation  
  spare1 : 7; // unused (reserved)  
  U16 spare2 : 16; // unused (reserved)  
} OPTION_STYLE, *P_OPTION_STYLE;
```

**Comments**
Note that changing style.cardNav is not supported.
Most clients do not need to deal with this message.

**msgOptionGetNeedCards**
Passes back the value of style.needCards.
Takes `P_BOOLEAN`, returns `STATUS`.

```c
#define msgOptionGetNeedCards MakeMsg(clsOption, 34)
```

**msgOptionSetNeedCards**
Sets style.needCards.
Takes `BOOLEAN`, returns `STATUS`.

```c
#define msgOptionSetNeedCards MakeMsg(clsOption, 35)
```

**Comments**
If style.needCards and style.getCards are true, the option sheet self-sends `msgOptionGetCards` when the current cards are needed.
Most clients do not need to deal with this message.

**msgOptionGetCard**
Passes back some information about a card in the option sheet.
Takes `P_OPTION_CARD`, returns `STATUS`.

```c
#define msgOptionGetCard MakeMsg(clsOption, 3)
```

**Message**
```c
typedef struct OPTION_CARD {  
  OPTION option; // out: option sheet sending the msg.  
  U32 tag; // in: tag for tab  
  WIN win; // in: card window or objNull  
  P_CHAR pName; // in: card name  
  U16 nameLen; // in: max. len for pName (for msgOptionGetCardAndName)  
  OBJECT client; // in: for msgOptionRefreshCard, etc.  
  U32 clientData[2]; // in: arbitrary client data  
  U32 spare1; // unused (reserved)  
  U32 spare2; // unused (reserved)  
} OPTION_CARD, *P_OPTION_CARD;
```
Comments

msgOptionGetTopCard

Passes back some information about the top card in the option sheet.

Takes _P_OPTION_CARD, returns STATUS.

```
#define msgOptionGetTopCard MakeMsg(clsOption, 25)
```

```
typedef struct OPTION_CARD {
    OPTION option; // out: option sheet sending the msg.
    U32 tag; // in: tag for tab
    WIN win; // in: card window or objNull
    P_CHAR pName; // in: card name
    U16 nameLen; // in: max. len for pName (for msgOptionGetCardAndName)
    OBJECT client; // in: for msgOptionRefreshCard, etc.
    U32 clientData[2]; // in: arbitrary client data
    U32 spare1; // unused (reserved)
    U32 spare2; // unused (reserved)
} OPTION_CARD, *P_OPTION_CARD;
```

Out parameters:

tag tag of the top card.

win uid of the card.

client client of the card.

If there is no top card, the option sheet sets all of the out parameters to null.

Most clients do not need to deal with this message.

msgOptionGetCardAndName

Passes back some information about a card in the option sheet.

Takes _P_OPTION_CARD, returns STATUS.

```
#define msgOptionGetCardAndName MakeMsg(clsOption, 20)
```

```
typedef struct OPTION_CARD {
    OPTION option; // out: option sheet sending the msg.
    U32 tag; // in: tag for tab
    WIN win; // in: card window or objNull
    P_CHAR pName; // in: card name
    U16 nameLen; // in: max. len for pName (for msgOptionGetCardAndName)
    OBJECT client; // in: for msgOptionRefreshCard, etc.
    U32 clientData[2]; // in: arbitrary client data
    U32 spare1; // unused (reserved)
    U32 spare2; // unused (reserved)
} OPTION_CARD, *P_OPTION_CARD;
```
Comments
In parameters:
tag  tag of the card to get.
pName  pointer to a buffer in which to put the card’s name.
nameLen  size of pName buffer in bytes (if 0, pName is ignored).
Out parameters:
win  uid of the card.
client  client of the card.
pName  buffer is filled in with the first nameLen bytes of the name of the card (if 0 was not passed for nameLen).
Will return stsBadParam if a card matching the passed tag was not found in the option sheet.
Most clients do not need to deal with this message.

msgOptionEnumCards
Enumerates the tags of the cards in the option sheet.
Takes P_OPTION_ENUM, returns STATUS.
#define msgOptionEnumCards MakeMsg(clsOption, 33)
typedef struct OPTION_ENUM {
    U16 max,  // in = size of pTags[] array
         count;  // in = # to return in array
                   // if count > max then memory may be allocated
         // out = # of valid entries in array
    P_TAG pTag;  // in = ptr to array of card tags
                   // out = if memory was allocated
                   // client should free the memory using OSHeapBlockFree()
    U16 next;  // in = 0 to start at beginning
                   // OR previous out value to pick up
                   // out = where we left off
    U32 flags;  // in = various flags (must be 0 for now)
    U32 spare;  // unused (reserved)
} OPTION_ENUM, *P_OPTION_ENUM;
This message is sent to enumerate all of the cards that have been added to the option sheet. Typical usage is shown below.

TAG cards[10];
OPTION_ENUM oe;
    oe.max  = 10;  // we have space for 10 card tags
    oe.count  = oe.max;  // we want all the card tags
    oe.pTag  = cards;  // our tag buffer
    oe.next  = 0;  // first call to msgOptionEnumCards
    oe.flags  = 0;  // unused for now
ObjCallRet(msgOptionEnumCards, sheet, &oe, s);
    // oe.pTag[0 .. oe.count] is the array of card tags
    // ...
    // free any allocated memory when finished with the tags
if (oe.pTag != cards)
    StsWarn(OSHeapBlockFree(oe.pTag));
Most clients do not need to deal with this message.
**msgOptionSetCard**

Changes some of the information of a card in the option sheet.

Takes `P_OPTION_CARD`, returns STATUS.

```c
#define msgOptionSetCard
MakeMsg(clsOption, 4)
```

**Message**

```c
typedef struct OPTION_CARD {
    OPTION option;        // out: option sheet sending the msg.
    U32 tag;              // in: tag for tab
    WIN win;              // in: card window or objNull
    P_CHAR pName;         // in: card name
    U16 nameLen;          // in: max. len for pName (for msgOptionGetCardAndName)
    OBJECT client;        // in: for msgOptionRefreshCard, etc.
    U32 clientData[2];   // in: arbitrary client data
    U32 spare1;           // unused (reserved)
    U32 spare2;           // unused (reserved)
} OPTION_CARD, *P_OPTION_CARD;
```

**Arguments**

- **tag**  tag of the card to set.
- **client**  client for the card.
- **win**  window for the card.
- **pName**  pointer to a buffer holding a new name, or `pNull` to keep the old name.

The option sheet changes the various parameters of the specified card. To avoid changing the name of the card, set `pArgs->pName` to `pNull`.

Most clients do not need to deal with this message.

**msgOptionAddCard**

Adds a card to the option sheet.

Takes `P_OPTION_CARD`, returns STATUS.

```c
#define msgOptionAddCard
MakeMsg(clsOption, 5)
```

**Message**

```c
typedef struct OPTION_CARD {
    OPTION option;        // out: option sheet sending the msg.
    U32 tag;              // in: tag for tab
    WIN win;              // in: card window or objNull
    P_CHAR pName;         // in: card name
    U16 nameLen;          // in: max. len for pName (for msgOptionGetCardAndName)
    OBJECT client;        // in: for msgOptionRefreshCard, etc.
    U32 clientData[2];   // in: arbitrary client data
    U32 spare1;           // unused (reserved)
    U32 spare2;           // unused (reserved)
} OPTION_CARD, *P_OPTION_CARD;
```

**Arguments**

- **tag**  tag of the card to set.
- **pName**  pointer to a buffer holding the card's name.
- **win**  window for the card.
- **client**  client for the card.
- **clientData**  any client data you want stored with the card.
If the card specified by `pArgs->tag` has already been added to the option sheet, the following is done:

- if `pArgs->win` is `objNull`, the window for the card is unchanged.
- otherwise, the current window for the card is destroyed and replaced by `pArgs->win`.
- if `pArgs->pName` is not `pNull`, the new name is used.
- the card client is replaced by `pArgs->client`.

Note that the card's tag is also used as the helpId of the tab button representing the card (in the popup choice card navigation menu or the tab bar). The caller should insure that quick help exists for the card with the card's tag as the helpId.

Most clients send this message to add a card to an option sheet (if there is more than one card).

See Also

`msgOptionAddLastCard`

---

**msgOptionAddLastCard**

Adds the last card of a group to the option sheet.

Takes `P_OPTION_CARD`, returns `STATUS`.

```c
#define msgOptionAddLastCard MakeMsg(clsOption, 29)
```

```c
typedef struct OPTION_LAST_CARD {
    OPTION option;         // out: option sheet sending the msg.
    U32 tag;               // in: tag for tab
    WIN win;               // in: card window or objNull
    P_CHAR pName;          // in: card name
    U16 nameLen;           // in: max. len for pName (for msgOptionGetCardAndName)
    OBJECT client;         // in: for msgOptionRefreshCard, etc.
    U32 clientData[2];     // in: arbitrary client data
    U32 spare1;            // unused (reserved)
    U32 spare2;            // unused (reserved)
} OPTION_LAST_CARD, *P_OPTION_LAST_CARD;
```

**Comments**

This is the same as `msgOptionAddCard`, except that the menu button for this card has a line break after it.

Most clients send this message to add the last card to an option sheet.

See Also

`msgOptionAddCard`

---

**msgOptionAddFirstCard**

Adds the first card of a group to the option sheet.

Takes `P_OPTION_CARD`, returns `STATUS`.

```c
#define msgOptionAddFirstCard MakeMsg(clsOption, 42)
```

```c
typedef struct OPTION_CARD {
    OPTION option;         // out: option sheet sending the msg.
    U32 tag;               // in: tag for tab
    WIN win;               // in: card window or objNull
    P_CHAR pName;          // in: card name
    U16 nameLen;           // in: max. len for pName (for msgOptionGetCardAndName)
    OBJECT client;         // in: for msgOptionRefreshCard, etc.
    U32 clientData[2];     // in: arbitrary client data
    U32 spare1;            // unused (reserved)
    U32 spare2;            // unused (reserved)
} OPTION_CARD, *P_OPTION_CARD;
```
This is the same as msgOptionAddCard, except that the menu button for this card has a line break before it.

Most clients don’t need to send this message.

See Also

msgOptionAddCard

msgOptionAddAndInsertCard

Adds a card to the option sheet and inserts it into the sheet.

Takes P_OPTION_CARD, returns STATUS.

```c
#define msgOptionAddAndInsertCard MakeMsg(clsOption, 17)
```

```c
typedef struct OPTION_CARD {
    OPTION option;    // out: option sheet sending the msg.
    U32 tag;          // in: tag for tab
    WIN win;          // in: card window or objNull
    P_CHAR pName;     // in: card name
    U16 nameLen;      // in: max. len for pName (for msgOptionGetCardAndName)
    OBJECT client;    // in: for msgOptionRefreshCard, etc.
    U32 clientData[2]; // in: arbitrary client data
    U32 spare1;       // unused (reserved)
    U32 spare2;       // unused (reserved)
} OPTION_CARD, *P_OPTION_CARD;
```

This message is handled exactly as in msgOptionAddCard, including the case in which pArgs->tag has already been added to the sheet.

Normally, msgOptionAddCard does not actually insert the card’s window into the option sheet’s window tree. msgOptionAddAndInsertCard does insert the window.

Most clients do not need to deal with this message.

See Also

msgOptionAddCard

msgOptionRemoveCard

Removes a card from an option sheet and destroys that card.

Takes P_OPTION_CARD, returns STATUS.

```c
#define msgOptionRemoveCard MakeMsg(clsOption, 6)
```

```c
typedef struct OPTION_CARD {
    OPTION option;    // out: option sheet sending the msg.
    U32 tag;          // in: tag for tab
    WIN win;          // in: card window or objNull
    P_CHAR pName;     // in: card name
    U16 nameLen;      // in: max. len for pName (for msgOptionGetCardAndName)
    OBJECT client;    // in: for msgOptionRefreshCard, etc.
    U32 clientData[2]; // in: arbitrary client data
    U32 spare1;       // unused (reserved)
    U32 spare2;       // unused (reserved)
} OPTION_CARD, *P_OPTION_CARD;
```

The option sheet removes and destroys the specified card. It also removes the window for the card, but does not destroy the window.

In parameters:

- **tag** tag of card to remove.

Will return stsBadParam if a card matching the passed tag was not found in the option sheet.
Most clients do not need to deal with this message.

**msgOptionExtractCard**

Extracts a card's window from an option sheet.

Takes P_OPTION_CARD, returns STATUS.

```c
#define msgOptionExtractCard MakeMsg(clsOption, 19)
```

```c
typedef struct OPTION_CARD {
    OPTION option; // out: option sheet sending the msg.
    U32 tag; // in: tag for tab
    WIN win; // in: card window or objNull
    P_CHAR pName; // in: card name
    U16 nameLen; // in: max. len for pName (for msgOptionGetCardAndName)
    OBJECT client; // in: for msgOptionRefreshCard, etc.
    U32 clientData[2]; // in: arbitrary client data
    U32 spare1; // unused (reserved)
    U32 spare2; // unused (reserved)
} OPTION_CARD, *P_OPTION_CARD;
```

**Comments**

The option sheet extracts the card's window, but does not destroy it. Note that the tab button for the card remains, with its win set to objNull.

In parameters:

- **tag** tag of card to extract.

Out parameters:

- **win** win of extracted card.

Will return stsBadParam if a card matching the passed tag was not found in the option sheet.

Most clients do not need to deal with this message.

**msgOptionShowCard**

Causes the specified card to be displayed as the current card.

Takes P_OPTION_CARD, returns STATUS.

```c
#define msgOptionShowCard MakeMsg(clsOption, 14)
```

```c
typedef struct OPTION_CARD {
    OPTION option; // out: option sheet sending the msg.
    U32 tag; // in: tag for tab
    WIN win; // in: card window or objNull
    P_CHAR pName; // in: card name
    U16 nameLen; // in: max. len for pName (for msgOptionGetCardAndName)
    OBJECT client; // in: for msgOptionRefreshCard, etc.
    U32 clientData[2]; // in: arbitrary client data
    U32 spare1; // unused (reserved)
    U32 spare2; // unused (reserved)
} OPTION_CARD, *P_OPTION_CARD;
```

**Comments**

The option sheet sends msgOptionRefreshCard to the card.

In parameters:

- **tag** tag of card to show.
Out parameters:
win    uid of card.
client  client of card.
Will return \texttt{stsBadParam} if a card matching the passed tag was not found in the option sheet.
Most clients do not need to deal with this message.

\textbf{msgOptionShowCardAndSheet}
Causes the specified card to be displayed as the current card.
Takes \texttt{TAG}, returns \texttt{STATUS}.
\begin{verbatim}
#define msgOptionShowCardAndSheet MakeMsg(clsOption, 44)
\end{verbatim}

\textbf{Comments}
The sheet is shown if it is not currently shown.
The option sheet self-sends \texttt{msgOptionShowCard(OPTION\_CARD.tag = pArgs)}, followed by \texttt{msgOptionShowSheet}.
Most clients do not need to deal with this message.

\textbf{See Also}
msgOptionShowCard

\textbf{msgOptionShowTopCard}
Shows the client-defined top card.
Takes nothing, returns \texttt{STATUS}.
\begin{verbatim}
#define msgOptionShowTopCard MakeMsg(clsOption, 30)
\end{verbatim}

\textbf{Comments}
The option sheet sends \texttt{msgOptionProvideTopCard} to its client with the following \texttt{OPTION\_CARD} parameters:
\begin{itemize}
\item option = uid of the option sheet
\item tag = tag of the current top card
\item win = win of the current top card
\item pNull = 0
\item client = client of the current top card
\end{itemize}
The option sheet then shows the new top card specified by \texttt{OPTION\_CARD.tag} by self-sending \texttt{msgOptionShowCard}.
Most clients do not need to deal with this message.

\textbf{msgOptionGetCards}
Gets the cards from the option sheet's client
Takes nothing, returns \texttt{STATUS}.
\begin{verbatim}
#define msgOptionGetCards MakeMsg(clsOption, 32)
\end{verbatim}

\textbf{Comments}
If \texttt{style.getCards} is false, this message is ignored. Otherwise, the option sheet sends \texttt{msgOptionAddCards} to its client with the following \texttt{OPTION\_TAG} parameters:
\begin{itemize}
\item option = uid of the option sheet
\item tag = tag of the option sheet
\end{itemize}
Most clients do not need to deal with this message.
**msgOptionApply**
Tell the option sheet to initiate the Apply protocol.
Takes nothing, returns STATUS.
```c
#define msgOptionApply MakeMsg(clsOption, 8)
```
Comments
This message is sent by the sheet's Apply button. The option sheet sends `msgOptionApplyCard` to the top card.
Most clients do not need to deal with this message.

**msgOptionApplyAndClose**
Tell an option sheet to run the Apply protocol and then close itself.
Takes nothing, returns STATUS.
```c
#define msgOptionApplyAndClose MakeMsg(clsOption, 9)
```
Comments
This message is sent by the sheet's Apply&Close button. The option sheet:
sends `msgOptionApplyCard` to the top card in the sheet, and
sends `msgOptionClosed` to the sheet's client.
Most clients do not need to deal with this message.

**msgOptionRefresh**
Tells an option sheet to refresh its card settings.
Takes nothing, returns STATUS.
```c
#define msgOptionRefresh MakeMsg(clsOption, 21)
```
Comments
This is sent to an option sheet by the default application code when it receives a forwarded "check" gesture.
If the apply buttons in the command bar are grayed out (i.e., the top card is not applicable), nothing is done, and `stsOK` is returned.
Otherwise, the option sheet sends `msgOptionRefreshCard` to its top card. It then marks the other cards as needing to be refreshed when shown.
Most clients do not need to deal with this message.

**msgOptionApplicable**
Tells an option sheet to ask the top card if it is applicable.
Takes `P_BOOL`, returns STATUS.
```c
#define msgOptionApplicable MakeMsg(clsOption, 37)
```
Comments
The option sheet sends `msgOptionApplicableCard` to its top card. It then marks the other cards as needing to be sent `msgOptionApplicableCard` when shown.
If the top card is not applicable, the command bar buttons are grayed out.
If `pArgs` is not `pNull`, true is passed back if the top card is applicable; otherwise false is passed back.
Most clients do not need to deal with this message.
msgOptionDirty
Tells an option sheet to ask the top card to dirty its controls.
Takes nothing, returns STATUS.
\#define msgOptionDirty MakeMsg(clsOption, 38)

Comments
The option sheet sends msgOptionDirtyCard to its top card. It then marks the other cards as needing to be sent msgOptionDirtyCard when shown.
Most clients do not need to deal with this message.

msgOptionClean
Tells an option sheet to ask the top card to clean its controls.
Takes nothing, returns STATUS.
\#define msgOptionClean MakeMsg(clsOption, 39)

Comments
The option sheet sends msgOptionCleanCard to its top card. The other cards are NOT cleaned.
Most clients do not need to deal with this message.

msgOptionToggleDirty
Tells an option sheet to toggle the dirty/clean state of the cards.
Takes nothing, returns STATUS.
\#define msgOptionToggleDirty MakeMsg(clsOption, 40)

Comments
The option sheet sends msgOptionProvideCardDirty to the top card's client to determine the dirty/clean state of the top card. If the client responds with stsNotUnderstood, the option sheet sends msgBorderGetDirty to the top card's window to determine the dirty/clean state.
If the top card is clean, msgOptionDirty is then self-sent; otherwise msgOptionClean is self-sent.
Most clients do not need to deal with this message.

msgOptionClose
Tells an option sheet to close itself.
Takes nothing, returns STATUS.
\#define msgOptionClose MakeMsg(clsOption, 10)

Comments
When a sheet receives msgOptionClose, it sends msgOptionClosed to the sheet's client.
A sheet self-sends msgOptionClose when it receives msgFrameClose.
Most clients do not need to deal with this message.

msgOptionGetCardMenu
Passes back the card navigation menu.
Takes P_MENU, returns STATUS.
\#define msgOptionGetCardMenu MakeMsg(clsOption, 26)

Comments
A copy of the popup card navigation menu is passed back. The option sheet returns objNull if style.cardNav is not osNavPopup.
Menu buttons in the navigation menu have option sheet as their client, `msgOptionShowCardAndSheet` as their message, and the appropriate card tag as their data. This causes the sheet being displayed and the appropriate card being turned to when the user taps on a menu button.

The caller must send `msgOptionCardMenuDone` when finished with the menu.

Most clients do not need to deal with this message.

`msgOptionCardMenuDone`

Indicates the caller is finished with the card menu.

Takes MENU, returns STATUS.

```c
#define msgOptionCardMenuDone MakeMsg(clsOption, 27)
```

Comments

This message should be sent to an option sheet when the card menu retrieved via `msgOptionGetCardMenu` is no longer needed.

Most clients do not need to deal with this message.

### Messages Option Sheets send to each card's client

#### `msgOptionShowSheet`

Asks the client of the option sheet to show the option sheet.

Takes P OPTION_TAG, returns STATUS. Category: client responsibility.

```c
#define msgOptionShowSheet MakeMsg(clsOption, 28)
```

#### `msgOptionProvideCardWin`

Asks the client of the card to provide the window for the card.

Takes P OPTION_CARD, returns STATUS. Category: client responsibility.

```c
#define msgOptionProvideCardWin MakeMsg(clsOption, 18)
```
This message is sent by the option sheet when a card is about to be shown, and the window for the card is `objNull`.

The card client should set `pArgs->win` to the desired card window.

Most clients need to override and handle this message.

**msgOptionProvideTopCard**

Asks the client of the option sheet to provide the tag for the top card.


```c
#define msgOptionProvideTopCard MakeMsg(clsOption, 31)
```

```c
typedef struct OPTION_CARD {
    OPTION option; // out: option sheet sending the msg.
    U32 tag; // in: tag for tab
    WIN win; // in: card window or objNull
    P_CHAR pName; // in: card name
    U16 nameLen; // in: max. len for pName (for msgOptionGetCardAndName)
    OBJECT client; // in: for msgOptionRefreshCard, etc.
    U32 clientData[2]; // in: arbitrary client data
    U32 spare1; // unused (reserved)
    U32 spare2; // unused (reserved)
} OPTION_CARD, *P_OPTION_CARD;
```

Comments

This message is sent by the option sheet when the top card must be shown. This can be in response to `msgOptionShowTopCard` or when the option sheet is first inserted.

The option sheet sends `msgOptionProvideTopCard` to its client with the following `OPTION_CARD` parameters:

- `option`: uid of the option sheet
- `tag`: tag of the current top card
- `win`: win of the current top card
- `pNull`: 0 = client of the current top card
- `client`: 0 = `msgNull`

The option sheet's client should set `pArgs->tag` to the tag for the desired top card.

Note that only `pArgs->tag` is used as an out parameter; other changes to `pArgs` are ignored.

**msgOptionShowTopCard**

**msgOptionProvideCardDirty**

Asks the client of the card to provide the dirtiness of the card window.


```c
#define msgOptionProvideCardDirty MakeMsg(clsOption, 41)
```

```c
typedef struct OPTION_CARD {
    OPTION option; // out: option sheet sending the msg.
    U32 tag; // in: tag for tab
    WIN win; // in: card window or objNull
    P_CHAR pName; // in: card name
    U16 nameLen; // in: max. len for pName (for msgOptionGetCardAndName)
    OBJECT client; // in: for msgOptionRefreshCard, etc.
    U32 clientData[2]; // in: arbitrary client data
    U32 spare1; // unused (reserved)
    U32 spare2; // unused (reserved)
} OPTION_CARD, *P_OPTION_CARD;
```

Comments

This message is sent by the option sheet in response to `msgOptionToggleDirty`.

The card's client should return `stsOK` if the card is dirty, `stsRequestDenied` if clean.

Most clients do not need to deal with this message.
**msgOptionApplyCard**

This is sent to a card's client when the card should apply its settings.

Takes P_OPTION_CARD, returns STATUS. Category: client responsibility.

```c
#define msgOptionApplyCard   MakeMsg(clsOption, 12)
```

```c
typedef struct OPTION_CARD {
    OPTION option;       // out: option sheet sending the msg.
    U32    tag;          // in: tag for tab
    WIN    win;          // in: card window or objNull
    P_CHAR pName;        // in: card name
    U16    nameLabel;    // in: max. len for pName (for msgOptionGetCardAndName)
    OBJECT client;      // in: for msgOptionRefreshCard, etc.
    U32    clientData[2];  // in: arbitrary client data
    U32    spare1;       // unused (reserved)
    U32    spare2;       // unused (reserved)
} OPTION_CARD, *P_OPTION_CARD;
```

With this message, an option option sheet tells a card to apply its settings to the selection. This is sent whenever the user chooses Apply or Apply&Close on the option sheet.

Most clients need to override and handle this message.

Here is the typical sequence of steps a card client should take in response:

Run through every control in the card and for each one 1) check to see if it's dirty, and if it is 2) validate it if necessary. If any control has an invalid value, return stsFailed from the handler for msgOptionApplyCard. (This step can be omitted if there's no way any control could have an invalid value.)

Again make a pass through every control in the card. If a control is dirty, apply its value.

Finally, clean all the controls in the card. This can usually be done by sending msgControlSetDirty(false) to the card window. Note that most "command sheets" should have their control's CONTROL_STYLE.showDirty set false, and so this final step should be omitted.

**msgOptionRefreshCard**

Tells a card's client to refresh its settings from the current selection.

Takes P_OPTION_CARD, returns STATUS. Category: client responsibility.

```c
#define msgOptionRefreshCard   MsgNoError(MakeMsg(clsOption, 11))
```

```c
typedef struct OPTION_CARD {
    OPTION option;       // out: option sheet sending the msg.
    U32    tag;          // in: tag for tab
    WIN    win;          // in: card window or objNull
    P_CHAR pName;        // in: card name
    U16    nameLabel;    // in: max. len for pName (for msgOptionGetCardAndName)
    OBJECT client;      // in: for msgOptionRefreshCard, etc.
    U32    clientData[2];  // in: arbitrary client data
    U32    spare1;       // unused (reserved)
    U32    spare2;       // unused (reserved)
} OPTION_CARD, *P_OPTION_CARD;
```

This is sent to a card's client when the option sheet has received msgOptionRefresh. The client should refresh the card's settings from the current selection.

Most clients need to override and handle this message.
msgOptionApplicableCard

Finds out if a card is applicable to the current selection.

Takes P_OPTION_CARD, returns STATUS. Category: client responsibility.

```c
#define msgOptionApplicableCard MakeMsg(clsOption, 22)
```

typedef struct OPTION_CARD {
    /* out: option sheet sending the msg. */
    OPTION option;
    U32 tag; /* in: tag for tab */
    WIN win; /* in: card window or objNull */
    P_CHAR pName; /* in: card name */
    U16 nameLen; /* in: max. len for pName (for msgOptionGetCardAndName) */
    OBJECT client; /* in: for msgOptionRefreshCard, etc. */
    U32 clientData[2]; /* in: arbitrary client data */
    U32 spare1; /* unused (reserved) */
    U32 spare2; /* unused (reserved) */
} OPTION_CARD, *P_OPTION_CARD;  

Comments

The card's client should respond by returning stsOK if the card can be applied to the current selection, stsFailed if not.

Most clients need to override and handle this message.

msgOptionDirtyCard

Sent to a card's client when the card should dirty all its controls.

Takes P_OPTION_CARD, returns STATUS. Category: client responsibility.

```c
#define msgOptionDirtyCard MakeMsg(clsOption, 23)
```

typedef struct OPTION_CARD {
    /* out: option sheet sending the msg. */
    OPTION option;
    U32 tag; /* in: tag for tab */
    WIN win; /* in: card window or objNull */
    P_CHAR pName; /* in: card name */
    U16 nameLen; /* in: max. len for pName (for msgOptionGetCardAndName) */
    OBJECT client; /* in: for msgOptionRefreshCard, etc. */
    U32 clientData[2]; /* in: arbitrary client data */
    U32 spare1; /* unused (reserved) */
    U32 spare2; /* unused (reserved) */
} OPTION_CARD, *P_OPTION_CARD;  

Comments

This is sent when the user changes the selection while an option sheet is up. It is needed so that if the card is applied to the new selection, every property on the card is applied, not just those changed by the user since the last apply.

The usual scenario is for the card window to inherit from clsBorder, whose instances respond to msgBorderSetDirty by forwarding that message on to their immediate children. Card clients may elect NOT to respond to msgOptionDirtyCard—if the option sheet code gets back stsNotUnderstood, then it will send msgBorderSetDirty(true) to the card window.

Most clients do not need to deal with this message.

msgOptionCleanCard

Sent to a card's client when the card should clean all its controls.

Takes P_OPTION_CARD, returns STATUS. Category: client responsibility.

```c
#define msgOptionCleanCard MakeMsg(clsOption, 36)
```
typedef struct OPTION_CARD {
    OPTION option;      // out: option sheet sending the msg.
    U32  tag;           // in:  tag for tab
    WIN  win;           // in:  card window or objNull
    P_CHAR pName;       // in:  card name
    UI6  nameLen;       // in:  max. len for pName (for msgOptionGetCardAndName)
    OBJECT client;      // in:  for msgOptionRefreshCard, etc.
    U32  clientData[2]; // in:  arbitrary client data
    U32  spare1;        // unused (reserved)
    U32  spare2;        // unused (reserved)
} OPTION_CARD, *P_OPTION_CARD;

This is sent after msgOptionApplyCard is sent.

The usual scenario is for the card window to inherit from clsBorder, whose instances respond to msgBorderSetDirty by forwarding that message on to their immediate children. Card clients may elect to NOT respond to msgOptionCleanCard—if the option sheet code gets back thestsNotUnderstood, then it will send msgBorderSetDirty(false) to the card window.

Most clients do not need to deal with this message.

msgOptionUpdateCard

Sent to a card's client every time the card is about to be shown.

Takes P_OPTION_CARD, returns STATUS. Category: client responsibility.

#define msgOptionUpdateCard     MsgNoError(MakeMsg(clsOption, 24))

typedef struct OPTION_CARD {
    OPTION option;      // out: option sheet sending the msg.
    U32  tag;           // in:  tag for tab
    WIN  win;           // in:  card window or objNull
    P_CHAR pName;       // in:  card name
    UI6  nameLen;       // in:  max. len for pName (for msgOptionGetCardAndName)
    OBJECT client;      // in:  for msgOptionRefreshCard, etc.
    U32  clientData[2]; // in:  arbitrary client data
    U32  spare1;        // unused (reserved)
    U32  spare2;        // unused (reserved)
} OPTION_CARD, *P_OPTION_CARD;

Most clients do not need to respond to this message. It is intended for those circumstances where one card has dependencies on the state of another, and would need to look at that other card before being (re)displayed to the user.

See Also

msgOptionRetireCard

msgOptionRetireCard

Sent to a card's client every time the current shown card is hidden.

Takes P_OPTION_CARD, returns STATUS. Category: client responsibility.

#define msgOptionRetireCard     MsgNoError(MakeMsg(clsOption, 43))

typedef struct OPTION_CARD {
    OPTION option;      // out: option sheet sending the msg.
    U32  tag;           // in:  tag for tab
    WIN  win;           // in:  card window or objNull
    P_CHAR pName;       // in:  card name
    UI6  nameLen;       // in:  max. len for pName (for msgOptionGetCardAndName)
    OBJECT client;      // in:  for msgOptionRefreshCard, etc.
    U32  clientData[2]; // in:  arbitrary client data
    U32  spare1;        // unused (reserved)
    U32  spare2;        // unused (reserved)
} OPTION_CARD, *P_OPTION_CARD;
Most clients do not need to respond to this message. It is intended for those circumstances where one card builds a context (e.g., allocates resources) when shown, and needs to destroy the context when the card is no longer shown. This can happen when another card is turned to or when the option sheet is extracted or destroyed.

See Also  
msgOptionUpdateCard

**Messages Option Sheets send to their frame’s client**

**msgOptionClosed**

This is sent to an option sheet’s client when the sheet is closed.

Takes OPTION, returns STATUS. Category: client responsibility.

```c
#define msgOptionClosed MakeMsg(clsOption, 13)
```

Comments  
The client should respond by using `msgAppRemoveFloatingWin` to take down the option sheet, then optionally destroying the sheet with `msgDestroy`.

**Messages sheet clients should self-send**

**msgOptionCreateSheet**

A message sent by convention by clients creating option sheets.

Takes P_OPTION_TAG, returns STATUS. Category: descendant responsibility.

```c
#define msgOptionCreateSheet MakeMsg(clsOption, 16)
```

**Message Arguments**

typedef struct OPTION_TAG {
    OPTION option;
    TAG tag;
} OPTION_TAG, *P_OPTION_TAG;

Comments  
When you need to create an option sheet, you should self-send this message, rather than directly creating a sheet. By following this convention, subclasses can modify the sheet or supply a different one (which would have to behave the same as the original).

When self-sending this message, the client should fill in the 'tag' of the option sheet desired (if applicable) or some other identifying value (some clients may create different kinds of option sheets). The client should also zero out the ‘option’ field of the OPTION_TAG struct.

In `msgOptionCreateSheet`, a client creates an EMPTY option sheet and fills in the 'option' field with the uid of the sheet. Subclasses handle this message by calling the ancestor's handler and then either modifying the sheet or supplying a new one (and destroying any non- null sheet already in the 'option' field).

**msgOptionAddCards**

A message to be sent by convention by clients creating option sheets.

Takes P_OPTION_TAG, returns STATUS. Category: descendant responsibility.

```c
#define msgOptionAddCards MakeMsg(clsOption, 15)
```
typedef struct OPTION_TAG {
    OPTION option;
    TAG tag;
} OPTION_TAG, *P_OPTION_TAG;

This message embodies the second step of creating an option sheet. Just like msgOptionCreateSheet, msgOptionAddCards is self-sent by a client to fill in a sheet with some cards, and to allow subclasses of the client to modify cards or add different ones.

if style.getCards is true, the option sheet sends this message to the frame’s client as follows:
- when the sheet is first inserted into the window tree
- if style.cardNav is osNavPopup, when the card navigation menu is needed after the selection has changed.

Messages from other classes

**msgContentsButtonGoto**

Default message sent when the user taps on a menu button.

Takes TAG, returns STATUS. Category: client notification.

**Comments**

This is also sent to the client when the managed button is hit.

The option sheet responds by self-sending msgOptionShowCard with the following OPTION_CARD parameter:

tag = pArgs;

**msgOptionBookProvideContents**

Receiver passes back a window representing its contents.

Takes P_WIN, returns STATUS.

**Comments**

The option sheet responds by creating an instance of clsContentsTable with one clsContentsButton child for each card in the option sheet. Cards which themselves respond to msgOptionBookProvideContents are represented by cbSection style contents buttons.
This file contains the API definition for clsOptionTable.

clsOptionTable inherits from clsTkTable.

Option tables implement no new behavior; they only change ancestor defaults to lay out their child
windows in the standard two-column table format used by option sheets.

```c
#ifndef OPTTABLE_INCLUDED
#define OPTTABLE_INCLUDED

#ifndef TKTABLE_INCLUDED
#include <tktable.h>
#endif
#endif
```

**Common #defines and typedefs**

```c
typedef OBJECT OPTION_TABLE;
typedef struct OPTION_TABLE_STYLE {
  UI16 spare  : 16;  // unused (reserved)
} OPTION_TABLE_STYLE, *P_OPTION_TABLE_STYLE;
```

**Messages**

**msgNew**

Creates an option table window.

Takes P_OPTION_TABLE_NEW, returns STATUS. Category: class message.

```c
typedef struct OPTION_TABLE_NEW_ONLY {
  U32 spare1;  // unused (reserved)
  U32 spare2;  // unused (reserved)
} OPTION_TABLE_NEW_ONLY, *P_OPTION_TABLE_NEW_ONLY;
#define optionTableNewFields  \
  tkTableNewFields  \
  OPTION_TABLE_NEW_ONLY  optionTable;
typedef struct OPTION_TABLE_NEW {
  optionTableNewFields  \
  OPTION_TABLE_NEW  optionTable;
} OPTION_TABLE_NEW, *P_OPTION_TABLE_NEW;
```

**msgNewDefaults**

Initializes the OPTION_TABLE_NEW structure to default values.

Takes P_OPTION_TABLE_NEW, returns STATUS. Category: class message.

```c
typedef struct OPTION_TABLE_NEW {
  optionTableNewFields  \
} OPTION_TABLE_NEW, *P_OPTION_TABLE_NEW;
```
pArgs->win.flags.style &= ~(wsClipChildren | wsFilelnline);
pArgs->border.style.leftMargin = bsMarginLarge;
pArgs->border.style.rightMargin = bsMarginLarge;
pArgs->border.style.bottomMargin = bsMarginLarge;
pArgs->border.style.topMargin = bsMarginLarge;
pArgs->gWin.style.grabDown = false;
pArgs->tableLayout.style.childXAlignment = tlAlignBaseline;
pArgs->tableLayout.style.childYAlignment = tlAlignBaseline;
pArgs->tableLayout.style.growChildWidth = false;
pArgs->tableLayout.style.growChildHeight = false;
pArgs->tableLayout.numRows.constraint = tllrifinite;
pArgs->tableLayout.numRows.value = 0;
pArgs->tableLayout.numCols.constraint = tlAbsolute;
pArgs->tableLayout.numCols.value = 2;
pArgs->tableLayout.colWidth.constraint = tlGroupMax | tlBaselineBox;
pArgs->tableLayout.rowHeight.constraint = tlGroupMax | tlBaselineBox;
pArgs->tableLayout.rowHeight.gap = defaultRowGap;
pArgs->tableLayout.colWidth.gap = defaultColGap;

Sets

Sends msgNewDefaults to clsLabel to initialize pNew->tkTable.pButtonNew, then sets:

pArgs->tkTable.pButtonNew->win.flags.style |= wsParentClip;
pArgs->tkTable.pButtonNew->win.flags.style &= ~(wsClipSiblings | wsClipChildren);
pArgs->tkTable.pButtonNew->border.style.backgroundInk = bsInkTransparent;
pArgs->tkTable.pButtonNew->label.style.fontType = lsFontCustom;
pArgs->tkTable.pButtonNew->label.font.attr.weight = sysDcWeightBold;
This file contains the API definition for clsPageNum.

clsPageNum inherits from clsLabel.

Page numbers are the standard notebook frame decorations which display the current page number.

```c
#ifndef PAGENUM_INCLUDED
#define PAGENUM_INCLUDED

#include <label.h>

#endif
```

### Common #defines and typedefs

typedef OBJECT PAGE_NUM;
typedef struct PAGE_NUM_STYLE {
    U16 spare = 16; // unused (reserved)
} PAGE_NUM_STYLE, *P_PAGE_NUM_STYLE;

### Messages

#### msgNew

Creates a pagenum window.

Takes P_PAGE_NUM_NEW, returns STATUS. Category: class message.

**Arguments**

typedef struct PAGE_NUM_NEW_ONLY {
    PAGE_NUM_STYLE style;
    U32 pageNumber; // initial page number
    U32 spare; // unused (reserved)
} PAGE_NUM_NEW_ONLY, *P_PAGE_NUM_NEW_ONLY;

#define pageNumNewFields
#define labelNewFields
#define PAGE_NUM_NEW_ONLY pageNum;

typedef struct PAGE_NUM_NEW {
    pageNumNewFields
} PAGE_NUM_NEW, *P_PAGE_NUM_NEW;

#### msgNewDefaults

Initializes the PAGE_NUM_NEW structure to default values.

Takes P_PAGE_NUM_NEW, returns STATUS. Category: class message.

**Message**

typedef struct PAGE_NUM_NEW {
    pageNumNewFields
} PAGE_NUM_NEW, *P_PAGE_NUM_NEW;

**Comments**

Zeroes out pArgs->pageNum and sets

- pArgs->border.style.leftMargin = bsMarginMedium;
- pArgs->border.style.rightMargin = bsMarginMedium;
- pArgs->border.style.bottomMargin = bsMarginSmall;
- pArgs->border.style.topMargin = bsMarginMedium;
pArgs->label.style.xAlignment = lsAlignRight;
pArgs->label.style.yAlignment = lsAlignCenter;

---

**msgPageNumGetStyle**

Passes back the current style values.

Takes P_PAGE_NUM_STYLE, returns STATUS.

```c
#define msgPageNumGetStyle MakeMsg(clsPageNum, 1)
```

**Message Arguments**

```c
typedef struct PAGE_NUM_STYLE {
    U16 spare : 16; // unused (reserved)
} PAGE_NUM_STYLE, *P_PAGE_NUM_STYLE;
```

---

**msgPageNumSetStyle**

Sets the style values.

Takes P_PAGE_NUM_STYLE, returns STATUS.

```c
#define msgPageNumSetStyle MakeMsg(clsPageNum, 2)
```

**Message Arguments**

```c
typedef struct PAGE_NUM_STYLE {
    U16 spare : 16; // unused (reserved)
} PAGE_NUM_STYLE, *P_PAGE_NUM_STYLE;
```

---

**msgPageNumGet**

Passes back the current page number.

Takes P_U32, returns STATUS.

```c
#define msgPageNumGet MakeMsg(clsPageNum, 3)
```

---

**msgPageNumSet**

Sets the current page number.

Takes U32, returns STATUS.

```c
#define msgPageNumSet MakeMsg(clsPageNum, 4)
```

---

**msgPageNumIncr**

Increments the current page number.

Takes S32, returns STATUS.

```c
#define msgPageNumIncr MakeMsg(clsPageNum, 5)
```
This file contains the API for clsPopupChoice.

clsPopupChoice inherits from clsMenuButton.

Popup choices are buttons that pop up a menu of choices when tapped.

A popup choice assumes that the first (bottom) child of its menu inherits from clsChoice. When this choice changes value, the popup choice button will copy the string of the new 'on' button in the choice as the popup choice's own string. Popup choices also respond to flick gestures by cycling their value among the set of possible values in the choice.

### Debugging Flags

The clsPopupChoice debugging flag is 'K'. Defined values are:

```c
#define POPUP_CHOICE_INCLUDED

#define POPUP_CHOICE_INCLUDED

#ifndef POPUP_CHOICE_INCLUDED

#define POPUP_CHOICE_INCLUDED

#include <choice.h>

#include <mbutton.h>

COnlnlon

defines and typedefs

typedef struct POPUP_CHOICE_STYLE {
    U16 spare;
} POPUP_CHOICE_STYLE, *P_POPUP_CHOICE_STYLE;

typedef OBJECT POPUP_CHOICE, *P_POPUP_CHOICE;

#define popupChoiceNewFields

#define msgNew

Creates a popup choice button.

Takes P_POPUP_CHOICE_NEW, returns STATUS. Category: class message.

Arguments:

```c
typedef struct POPUP_CHOICE_NEW_ONLY {
    POPUP_CHOICE_STYLE style;
    U32 spare; // unused (reserved)
} POPUP_CHOICE_NEW_ONLY, *P_POPUP_CHOICE_NEW_ONLY;

#define popupChoiceNewFields

#define msgNew

Creates a popup choice button.

Takes P_POPUP_CHOICE_NEW, returns STATUS. Category: class message.

Arguments:

```c
typedef typedef struct POPUP_CHOICE_NEW { ...

Comments

The popup choice will set its pString from the 'on' button within the popup's choice, if any button there is 'on'.

```
The fields you commonly set are:

\[ \text{pArgs->menuButton.menu.uid of a menu whose first child is a choice} \]

---

**msgNewDefaults**

Initializes the POPUP_CHOICE_NEW structure to default values.

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

**Message**

typedef struct POPUP_CHOICE_NEW {
    popupChoiceNewFields
} POPUP_CHOICE_NEW, *P_POPUP_CHOICE_NEW;

**Comments**

Zeroes out `pArgs->popupChoice` and sets:

- `pArgs->gWin.style.gestureEnable = true`
- `pArgs->control.style.showDirty = true`
- `pArgs->label.style.decoration = lsDecorationPopup`
- `pArgs->button.style.feedback = bsFeedbackNone`
- `pArgs->menuButton.style.subMenuType = mbMenuPopup`
- `pArgs->menuButton.style.getWidth = true`

---

**msgPopupChoiceGetStyle**

Passes back the receiver's style. NOT IMPLEMENTED.

Takes `P.Popup_CHOICE_STYLE`, returns `STATUS`.

#define msgPopupChoiceGetStyle MakeMsg(clsPopupChoice, 1)

**Message**

typedef struct POPUP_CHOICE_STYLE {
    U16 spare;
} POPUP_CHOICE_STYLE, *P_POPUP_CHOICE_STYLE;

---

**msgPopupChoiceSetStyle**

Sets the receiver's style. NOT IMPLEMENTED.

Takes `P_POPUP_CHOICE_STYLE`, returns `STATUS`.

#define msgPopupChoiceSetStyle MakeMsg(clsPopupChoice, 2)

**Message**

typedef struct POPUP_CHOICE_STYLE {
    U16 spare;
} POPUP_CHOICE_STYLE, *P_POPUP_CHOICE_STYLE;

---

**msgPopupChoiceGetChoice**

Passes back the choice associated with this popup.

Takes `P_CHOICE`, returns `STATUS`.

#define msgPopupChoiceGetChoice MakeMsg(clsPopupChoice, 3)

**Comments**

The popup choice will self-send `msgMenuButtonGetMenu` to get the menu. If the menu is null, the popup choice will set `*pArgs` null and return `stsOK`. Otherwise, `*pArgs` will be set to the first child of the menu.
Messages from Other Classes

msgWinSend
Sends a message up a window ancestry chain.
Takes P_WIN_SEND, returns STATUS.

Comments
If pArgs->msg is not msgMenuDone, clsPopupChoice just calls its ancestor.
Otherwise, clsPopupChoice calls its ancestor (to allow clsMenuButton to take down the menu), then resets its visuals to reflect the new 'on' button within the choice.
For popup choices that display a string, this just means obtaining the string from the 'on' button (or, if the button has LABEL_STYLE.infoType of IsInfoWindow, from the first IsInfoString label found within using depth enumeration) and using msgLabelSetString on self.
For popup choices that display an icon, the visuals are changed by getting the icon within self (msgLabelGetWin), sending it msgIconFreeCache, setting its window tag to the tag of the 'on' icon, and finally using msgWinDirtyRect(pNull) to get the icon to repaint. Note that because of this strategy, the icon within self cannot change size when its picture changes. The picture size is not copied from the 'on' icon to the icon within self.

msgGWinGesture
Self-sent to process the gesture.
Takes P_GWIN_GESTURE, returns STATUS.

Comments
If the popup's CONTROL_STYLE.enable is false, the popup choice just returns stsOK.
If the class of pArgs->msg is not clsXGesture, the popup choice returns stsMessageIgnored.
If pArgs->msg is not one of xgsFlick* or xgsDbflick*, then the popup choice returns the result of calling its ancestor.
Otherwise, the popup choice obtains the 'on' button within its choice, and searches through the choice's list of children for the next, previous, first, or last child based on what type of flick gesture was received. The popup choice sets its value to be this new button and returns stsOK. Buttons that are not enabled (msgControlGetEnable) are skipped over.

Return Value
stsMessageIgnored pArgs->msg is not of clsXGesture.

msgControlGetValue
Passes back the receiver's value (tag of button that is on).
Takes P_TAG, returns STATUS.

Comments
clsPopupChoice overrides clsButton's response (of passing back BUTTON_STYLE.on) by instead forwarding msgControlGetValue on to its choice. This means popup choices behave like choices with respect to msgControlGetValue.
msgControlSetValue
Sets the receiver's value.
Takes TAG, returns STATUS.

Comments
clsPopupChoice overrides clsButton's response (of setting BUTTON_STYLE.on) by instead forwarding msgControlSetValue on to its choice. Changing the choice's value then results in an update of the popup's label string. This means popup choices behave like choices with respect to msgControlSetValue.

msgControlGetClient
Passes back the receiver's client.
Takes P_UID, returns STATUS.

Comments
clsPopupChoice intercepts this message and forwards it on to the popup's choice.

msgControlSetClient
clsPopupChoice forwards this message on to the popup's choice.
Takes UID, returns STATUS.

msgControlBeginPreview
clsPopupChoice responds by noting internally that its menu is now up, then calling ancestor.
Takes P_INPUT_EVENT, returns STATUS.

msgControlSetMetrics
Sets the metrics.
Takes P_CONTROL_METRICS, returns STATUS.

Comments
If the popup choice's menu is up, it prohibits the CONTROL_STYLE.dirty bit from changing.

msgControlSetStyle
Sets the style values.
Takes P_CONTROL_STYLE, returns STATUS.

Comments
If the popup choice's menu is up, it prohibits the CONTROL_STYLE.dirty bit from changing.

msgControlSetDirty
Sets style.dirty.
Takes BOOLEAN, returns STATUS.

Comments
If the popup choice's menu is up, it prohibits the CONTROL_STYLE.dirty bit from changing.

msgMenuButtonProvideWidth
Self-sent when MENU_BUTTON_STYLE.getWidth is true.
Takes P_S32, returns STATUS. Category: self-sent.

Comments
clsPopupChoice responds by computing a width based on its menu.
If the wsLayoutDirty bit of its menu is true, the popup choice will lay out its menu. clsPopupChoice then enumerates all the children of its choice and computes the maximum width of all the children that inherit from clsLabel and whose LABEL_STYLE.infoType is not IsInfoWindow (if an IsInfoWindow label child is encountered, clsPopupChoice will find the first string-type label within it and use the width of that).

**msgMenuButtonPlaceMenu**

Self-sent whenever a menu button needs to position its associated menu.

Takes P_WIN_METRICS, returns STATUS. Category: self-sent.

**Comments**

clsPopupChoice first gets the 'on' button from its choice. If there is a button on, clsPopupChoice will position its menu so that the 'on' button is adjacent to the popup. If there is no button on in the choice, clsPopupChoice just calls its ancestor.
This file contains the API for clsProgressBar.
clsProgressBar inherits from clsControl.
Implements a read-only or read/write progress indicator.

## Debugging Flags

The clsProgressBar debugging flag is 'K'. Defined values are:

```c
flag14 (0x4000)  general
#ifndef PROGRESS_INCLUDED
#define PROGRESS_INCLUDED
#define PROGRESS_INCLUDED
#endif
#include <control.h>
#endif
#ifndef SYSGRAF_INCLUDED
#include <sysgraf.h>
#endif
#endif
```

## Common #defines and typedefs

```c
// Labels style
#define psLabelsNumeric 0
#define psLabelsNone 1
#define psLabelsCustom 2

// Ticks style
#define psTicksSmall 0
#define psTicksFull 1
#define psTicksNone 2

// Direction style
#define psDirectionHorizontal 0  // horizontal indicator
#define psDirectionVertical 1    // vertical indicator

// Thickness style
#define psThicknessRelFont 0     // thickness varies with system font size
#define psThicknessFixed 1       // thickness is fixed

// Edge Styles
#define psEdgeNone
#define psEdgeMinLat
#define psEdgeMaxLat
#define psEdgeMinLong
#define psEdgeMaxLong
#define psEdgeAll
```

(psEdgeMinLat, psEdgeMaxLat, psEdgeMinLong, psEdgeMaxLong)
“Lat” is latitude, and “Long” is longitude. For horizontal progress bars, latitude is the y dimension (or minor axis), and longitude is the x dimension (or major axis). For vertical bars, lat is x, and long is y.

typedef struct PROGRESS_STYLE {
    U16 labels : 2,       // labels style
ticks : 2,              // style of ticks to paint
direction : 2,           // direction of major axis
units : 6,               // units for everything except labels
thickness : 2,           // thickness style for lines and ticks
labelRotation : 2;       // use lsRotate* from label.h
U16 labelScaleUnits : 6, // scale units for labels from border.h
edge : 4,                // bar edges to display
        // (separate from clsBorder edges)
labelFontType : 2,        // use lsFont* from label.h
spare : 4;                // unused (reserved)
U16 spare2 : 16;          // unused (reserved)
} PROGRESS_STYLE, *P_PROGRESS_STYLE;

Default PROGRESS_STYLE:

    labels = psLabelsNone
ticks = psTicksNone
direction = psDirectionHorizontal
units = bsUnitsPoints
thickness = psThicknessRelFont
labelRotation = lsRotateNone
labelFontType = lsFontSystem
labelScaleUnits = bsUnitsLayout
edge = psEdgeMinLat | psEdgeMinLong

typedef struct PROGRESS_REGION {
    U32 rgb;
    SYSDC_PATTERN pattern;
} PROGRESS_REGION, *P_PROGRESS_REGION;

typedef struct PROGRESS_METRICS {
    PROGRESS_STYLE style;       // overall style
    S32 numIntervals;
    S32 ticksPerLabel;          // gives period of labels
    S32 minNumericLabel;        // when psLabelsNumeric
    S32 maxNumericLabel;        // when psLabelsNumeric
    U16 thicknessBase;          // thickness (units or multiplier)
    U16 latitude;               // dimension of minor axis (in units)
    U16 longitude;              // dimension of major axis (in units)
    S32 maxValue;               // values are in [0..maxValue]
    S32 value;                  // current value
    SYSDC_FONT_SPEC font;       // spec to open if style.labelFontType == lsFontCustom
    U8 labelScale;              // scale for labels as in border.h
    U32 spare1;                 // unused (reserved)
    U32 spare2;                 // unused (reserved)
} PROGRESS_METRICS, *P_PROGRESS_METRICS;

metrics.latitude and .longitude are used only when the progress bar is shrink-wrapped in those dimensions. When not shrink-wrapped, the progress bar expands to fill the available space.

msgNew

Creates a progress indicator.

Takes P_PROGRESS_NEW, returns STATUS. Category: class message.

typedef struct PROGRESS_NEW_ONLY {
    PROGRESS_METRICS metrics;
    P CHAR fontName;       // font name from which to derive font.id
    U32 spare1;            // unused (reserved)
    U32 spare2;            // unused (reserved)
} PROGRESS_NEW_ONLY, *P_PROGRESS_NEW_ONLY;
```c
#define progressNewFields
controlNewFields
PROGRESS_NEW_ONLY progress;
typedef struct PROGRESS_NEW {
    progressNewFields
} PROGRESS_NEW, *P_PROGRESS_NEW;
```

The filled region looks are initialized with:

```c
gbg = SysDcGrayRGB(128)
pattern = sysDcPatForeground
```

The unfilled region looks are initialized with:

```c
gbg = sysDcRGBTransparent
pattern = sysDcPatNil
```

### msgNewDefaults

Initializes the PROGRESS_NEW structure to default values.

Takes P_PROGRESS_NEW, returns STATUS. Category: class message.

```c
typedef struct PROGRESS_NEW {
    progressNewFields
} PROGRESS_NEW, *P_PROGRESS_NEW;
```

Comments

Zeroes out pArgs->progress and sets:

```c
pArgs->win.flags.style |= wsShrinkWrapWidth | wsShrinkWrapHeight;
pArgs->border.style.previewAlter = bsAlterNone;
pArgs->border.style.selectedAlter = bsAlterNone;
pArgs->control.style.showDirty = false;
pArgs->progress.metrics.style.labels = psLabelsNone;
pArgs->progress.metrics.style.ticks = psTicksNone;
pArgs->progress.metrics.style.units = bsUnitsPoints;
pArgs->progress.metrics.style.labelScaleUnits = bsUnitsLayout;
pArgs->progress.metrics.style.edge = psEdgeAll;
pArgs->progress.metrics.numIntervals = 10;
pArgs->progress.metrics.ticksPerLabel = 2;
pArgs->progress.metrics.minNumericLabel = 0;
pArgs->progress.metrics.maxNumericLabel = 100;
pArgs->progress.metrics.thicknessBase = 1;
pArgs->progress.metrics.latitude = 18;
pArgs->progress.metrics.longitude = 144;
pArgs->progress.metrics.maxValue = 100;
pArgs->progress.metrics.value = 0;
pArgs->progress.metrics.labelScale = lsScaleMedium;
```

Also sets pArgs->progress.metrics.font to the default system font.

### msgProgressGetStyle

Passes back the current style.

Takes P_PROGRESS_STYLE, returns STATUS.

```c
#define msgProgressGetStyle MakeMsg(clsProgressBar, 1)
```
typedef struct PROGRESS_STYLE {
    U16 labels : 2, // labels style
    ticks : 2, // style of ticks to paint
direction : 2, // direction of major axis
    units : 6, // units for everything except labels
    thickness : 2, // thickness style for lines and ticks
    labelRotation : 2; // use lsRotate* from label.h
    edge : 4, // bar edges to display
    // (separate from clsBorder edges)
    labelScaleUnits : 6, // scale units for labels from border.h
    edge : 4, // bar edges to display
    // (separate from clsBorder edges)
    labelFontType : 2, // use lsFont* from label.h
    spare : 4; // unused (reserved)
    } PROGRESS_STYLE, *P_PROGRESS_STYLE;

msgProgressSetStyle
Sets the style.
Takes P_PROGRESS_STYLE, returns STATUS.
#define msgProgressSetStyle MakeMsg(clsProgressBar, 2)

typedef struct PROGRESS_STYLE {
    U16 labels : 2, // labels style
    ticks : 2, // style of ticks to paint
direction : 2, // direction of major axis
    units : 6, // units for everything except labels
    thickness : 2, // thickness style for lines and ticks
    labelRotation : 2; // use lsRotate* from label.h
    edge : 4, // bar edges to display
    // (separate from clsBorder edges)
    labelScaleUnits : 6, // scale units for labels from border.h
    edge : 4, // bar edges to display
    // (separate from clsBorder edges)
    labelFontType : 2, // use lsFont* from label.h
    spare : 4; // unused (reserved)
    } PROGRESS_STYLE, *P_PROGRESS_STYLE;

The progress bar will set its layout bit dirty (as in msgWinSetLayoutDirty) as necessary. It will use msgWinDirtyRect in a similar fashion. It is the client's responsibility to send msgWinLayout to the progress bar whenever the style changes would affect the layout.

msgProgressGetMetrics
Passes back the current metrics.
Takes P_PROGRESS_METRICS, returns STATUS.
#define msgProgressGetMetrics MakeMsg(clsProgressBar, 3)

typedef struct PROGRESS_METRICS {
    PROGRESS_STYLE style; // overall style
    numIntervals;
    ticksPerLabel; // gives period of labels
    minNumericLabel; // when psLabelsNumeric
    maxNumericLabel; // when psLabelsNumeric
    thicknessBase; // thickness (units or multiplier)
    latitude; // dimension of minor axis (in units)
    longitude; // dimension of major axis (in units)
    maxValue; // values are in [0..maxValue]
    value; // current value
    SYSDC_FONT_SPEC font; // spec to open if style.labelFontType == lsFontCustom
    labelScale; // scale for labels as in border.h
    spare1; // unused (reserved)
    spare2; // unused (reserved)
} PROGRESS_METRICS, *P_PROGRESS_METRICS;
msgProgressSetMetrics

Sets the metrics.

Takes P_PROGRESS_METRICS, returns STATUS.

```c
#define msgProgressSetMetrics MakeMsg(clsProgressBar, 4)
```

typedef struct PROGRESS_METRICS {
    PROGRESS_STYLE style;       // overall style
    S32 numIntervals;
    S32 ticksPerLabel;          // gives period of labels
    S32 minNumericLabel;        // when psLabelsNumeric
    S32 maxNumericLabel;        // when psLabelsNumeric
    U16 thicknessBase;          // thickness (units or multiplier)
    U16 latitude;               // dimension of minor axis (in units)
    U16 longitude;              // dimension of major axis (in units)
    S32 maxValue;               // values are in [0..maxValue]
    S32 value;                  // current value
    SYSDC_FONT_SPEC font;       // spec to open if style.labelFontType == lsFontCustom
    U8 labelScale;              // scale for labels as in border.h
    U32 spare1;                 // unused (reserved)
    U32 spare2;                 // unused (reserved)
} PROGRESS_METRICS, *P_PROGRESS_METRICS;

The progress bar will set its layout bit dirty (as in msgWinSetLayoutDirty) as necessary. It will use msgWinDirtyRect in a similar fashion. It is the client's responsibility to send msgWinLayout to the progress bar whenever the changes would affect the layout.

msgProgressGetFilled

Passes back the current filled region looks.

Takes P_PROGRESS_REGION, returns STATUS.

```c
#define msgProgressGetFilled MakeMsg(clsProgressBar, 5)
```

typedef struct PROGRESS_REGION {
    U32 rgb;
    SYSDC_PATTERN pattern;
} PROGRESS_REGION, *P_PROGRESS_REGION;

msgProgressSetFilled

Sets the current filled region looks.

Takes P_PROGRESS_REGION, returns STATUS.

```c
#define msgProgressSetFilled MakeMsg(clsProgressBar, 6)
```

typedef struct PROGRESS_REGION {
    U32 rgb;
    SYSDC_PATTERN pattern;
} PROGRESS_REGION, *P_PROGRESS_REGION;

The progress bar will self-send msgWinDirtyRect as necessary.

msgProgressGetUnfilled

Passes back the current unfilled region looks.

Takes P_PROGRESS_REGION, returns STATUS.

```c
#define msgProgressGetUnfilled MakeMsg(clsProgressBar, 7)
```
typedef struct PROGRESS_REGION {
    U32 rgb;
    SYS DC_PATTERN pattern;
} PROGRESS_REGION, *P_PROGRESS_REGION;

msgProgressSetUnfilled
Sets the current unfilled region looks.
Takes P_PROGRESS_REGION, returns STATUS.
#define msgProgressSetUnfilled MakeMsg(clsProgressBar, 8)

typedef struct PROGRESS_REGION {
    U32 rgb;
    SYS DC_PATTERN pattern;
} PROGRESS_REGION, *P_PROGRESS_REGION;

The progress bar will self-send msgWinDirtyRect as necessary.

msgProgressProvideLabel
Sent to the client when style.labels == psLabelsCustom.
Takes P_PROGRESS_PROVIDE_LABEL, returns STATUS. Category: client responsibility.
#define msgProgressProvideLabel MakeMsg(clsProgressBar, 9)

typedef struct PROGRESS_PROVIDE_LABEL {
    CONTROL progressBar;    // In: requestor
    U16 position;           // In: position of label (0 at minimum)
    P_CHAR pString;         // Out: a 256 byte buffer for the string
    U32 spare;              // unused (reserved)
} PROGRESS_PROVIDE_LABEL, *P_PROGRESS_PROVIDE_LABEL;

The client should copy a string for the indicated position into the provided buffer.

msgProgressGetVisInfo
Passes back information about the current state of the visuals.
Takes P_PROGRESS_VIS_INFO, returns STATUS.
#define msgProgressGetVisInfo MakeMsg(clsProgressBar, 10)

typedef struct PROGRESS_VIS_INFO {
    RECT32 filledRect,
    unfilledRect;
    U32 spare1;          // unused (reserved)
    U32 spare2;          // unused (reserved)
} PROGRESS_VIS_INFO, *P_PROGRESS_VIS_INFO;

All measurements are in LWC (device units).

Messages from Other Classes

msgControlGetValue
Passes back the receiver's value (metrics.value).
Takes P_S32, returns STATUS.
**msgControlSetValue**
Sets the receiver's value.
Takes S32, returns STATUS.

Comments
The progress bar will self-send msgWinDirtyRect as necessary.

**msgSave**
Causes an object to file itself in an object file.
Takes P_OBJ_SAVE, returns STATUS.

Comments
clsProgressBar responds by filing away all of its state.

**msgRestore**
Creates and restores an object from an object file.
Takes P_OBJ_RESTORE, returns STATUS.

Comments
clsLabel responds by restoring all of its state.

**msgWinLayoutSelf**
Tell a window to layout its children.
Takes P_WIN_METRICS, returns STATUS.

Comments
clsProgressBar responds by recomputing its layout parameters.

If the receiver is shrink-wrapping in a dimension, it will use the latitude or longitude value as appropriate to determine the interior dimension of the progress bar (which does not include the inked edges of the bar). When not shrink-wrapping in a dimension, the corresponding latitude or longitude value is ignored.

**msgWinRepaint**
Tells a window to repaint itself.
Takes nothing, returns STATUS. Category: descendant responsibility.

Comments
clsProgressBar responds by painting the edges, bar, ticks, and labels.

First, the progressBar self-sends msgControlGetValue to get its current value and then msgBorderGetForegroundRGB to get the color in which to paint the edges, ticks, and labels. It then paints the edges.

Next, the progressBar will paint the unfilled portion of the bar if the unfilled pattern isn't sysDcPatNil. The pattern will be painted with the specified foreground RGB and a background RGB obtained by self-sending msgBorderGetBackgroundRGB. See msgProgressSetUnfilled.

The progressBar will then paint the filled portion of the bar if the filled pattern isn't sysDcPatNil. The method is as described above. See msgProgressSetFilled.

While drawing the tick marks, the progressBar will self-send msgBorderRGBToInk and use a foreground color that is opposite so that the ticks will show up against the filled/unfilled regions.

Finally, the labels are painted using a foreground RGB obtained by self-sending msgBorderGetForegroundRGB.
msgWinGetBaseline

Gets the desired x,y alignment of a window.
Takes P.Win_METRICS, returns STATUS.

Comments

clsProgressBar responds by setting pArgs->bounds.origin to the origin of the bar within self.
This file contains the API definition for clsScrollbar. clsScrollbar inherits from clsControl.

Scrollbars provide scrolling visuals and define a protocol for handling various kinds of scrolling actions.

**Debugging Flags**

The clsScrollbar debugging flag is 'K'. Defined values are:

- flag2 (0x0004) protocol messages
- flag6 (0x0040) painting
- flag10 (0x0400) input
- flag14 (0x4000) general debug info

```
#ifndef SBAR_INCLUDED
#define SBAR_INCLUDED
#include <control.h>
#ifndef CONTROL_INCLUDED
#endif
```

**Common #defines and typedefs**

```
#define hlpScrollbarVertical MakeTag(clsScrollbar, 1)
#define hlpScrollbarHorizontal MakeTag(clsScrollbar, 2)
#define hlpScrollbarGeneral hlpScrollbarVertical
typedef OBJECT SCROLLBAR;
```

**Direction**

```
#define sbDirectionVertical 0 // vertical scrollbar
#define sbDirectionHorizontal 1 // horizontal scrollbar

typedef struct SCROLLBAR_STYLE {
  U16 direction : 1,
  wide : 1, // no longer implemented
  spare : 14; // unused (reserved)
} SCROLLBAR_STYLE, *P_SCROLLBAR_STYLE;
```

Default SCROLLBAR_STYLE:

```
direction = sbDirectionVertical
```

Enum16(SCROLLBAR_ACTION) {
  // For vertical scrollbars:
  sbLineUp = 0,
  sbLineDown = 1,
  sbPageUp = 2,
  sbPageDown = 3,
  sbThumbUpDown = 4,
  sbLineToTop = 11,
  sbLineToBottom = 12,
  sbToTop = 15,
  sbToBottom = 16,
```
// For horizontal scrollbars:
sbLineLeft = 5,
sbLineRight = 6,
sbPageLeft = 7,
sbPageRight = 8,
sbThumbLeftRight = 9,
sbColumnToLeft = 13,
sbColumnToRight = 14,
sbToLeft = 17,
sbToRight = 18,
// Terminating action:
sbEndScroll = 10
};
typedef struct SCROLLBAR_SCROLL {
  SCROLLBAR sb;       // in: originating scrollbar
  SCROLLBAR_ACTION action; // in: current action
  S32 offset;         // in/out: current or new offset
  S32 lineCoord;      // in: coordinate of line in root win space
  U32 spare1;         // unused (reserved)
  U32 spare2;         // unused (reserved)
} SCROLLBAR_SCROLL, *P_SCROLLBAR_SCROLL;
typedef struct SCROLLBAR_PROVIDE {
  SCROLLBAR sb;       // in: originating scrollbar
  S32 viewLength;     // out: client-provided view width or height
  S32 docLength;      // out: client-provided document width or height
  S32 offset;         // out: client-provided current offset
  U32 spare;          // unused (reserved)
} SCROLLBAR_PROVIDE, *P_SCROLLBAR_PROVIDE;

**Messages**

**msgNew**

Creates a scrollbar window.

Takes P_SCROLLBAR_NEW, returns STATUS. Category: class message.

**Arguments**

typedef struct SCROLLBAR_NEW_ONLY {
  SCROLLBAR_STYLE style;
  U32 spare1;       // unused (reserved)
  U32 spare2;       // unused (reserved)
} SCROLLBAR_NEW_ONLY, *P_SCROLLBAR_NEW_ONLY;
#define scrollbarNewFields \
  controlNewFields  \
  SCROLLBAR_NEW_ONLY scrollbar;

typedef struct SCROLLBAR_NEW {
  scrollbarNewFields
} SCROLLBAR_NEW, *P_SCROLLBAR_NEW;

**Comments**

The fields you commonly set are:

pArgs->scrollbar.style.direction  whether horizontal or vertical

**msgNewDefaults**

Initializes the SCROLLBAR_NEW structure to default values.

Takes P_SCROLLBAR_NEW, returns STATUS. Category: class message.

**Message**

**Arguments**

typedef struct SCROLLBAR_NEW {
  scrollbarNewFields
} SCROLLBAR_NEW, *P_SCROLLBAR_NEW;
msgScrollbarGetStyle

Passes back the current style values.

Takes P_SCROLLBAR_STYLE, returns STATUS.

```c
#define msgScrollbarGetStyle MakeMsg(clsScrollbar, 1)
```

```
typedef struct SCROLLBAR_STYLE {
    U16 direction : 1,
    wide : 1,     // no longer implemented
    spare : 14;   // unused (reserved)
} SCROLLBAR_STYLE, *P_SCROLLBAR_STYLE;
```

msgScrollbarSetStyle

Sets the style values.

Takes P_SCROLLBAR_STYLE, returns STATUS.

```c
#define msgScrollbarSetStyle MakeMsg(clsScrollbar, 2)
```

```
typedef struct SCROLLBAR_STYLE {
    U16 direction : 1,
    wide : 1,     // no longer implemented
    spare : 14;   // unused (reserved)
} SCROLLBAR_STYLE, *P_SCROLLBAR_STYLE;
```

msgScrollbarUpdate

Forces the scrollbar to repaint with the latest info.

Takes nothing, returns STATUS.

```c
#define msgScrollbarUpdate MakeMsg(clsScrollbar, 14)
```

Causes msgScrollbarProvideVert/HorizInfo to be sent to client.

Self-Sent/Client Messages

msgScrollbarVertScroll

Client should perform vertical scroll.

Takes P_SCROLLBAR_SCROLL, returns STATUS. Category: client responsibility.

```c
#define msgScrollbarVertScroll MakeMsg(clsScrollbar, 5)
```

```
typedef struct SCROLLBAR_SCROLL {
    SCROLLBAR sb;       // in: originating scrollbar
    SCROLLBAR_ACTION action;     // in: current action
    S32 offset;          // in/out: current or new offset
    S32 lineCoord;       // in: coordinate of line in root win space
    U32 spare1;          // unused (reserved)
    U32 spare2;          // unused (reserved)
} SCROLLBAR_SCROLL, *P_SCROLLBAR_SCROLL;
```
The passed offset is an estimate computed by the scrollbar based on the information obtained from `msgScrollbarProvideVertInfo`.

If the client is unwilling to scroll to this offset, the client may scroll to a different offset. Be sure to set `pArgs->offset` to the new offset if it's different from the passed value.

---

**msgScrollbarHorizScroll**

Client should perform horizontal scroll.


```c
#define msgScrollbarHorizScroll MakeMsg(clsScrollbar, 6)
```

```c
typedef struct SCROLLBAR_SCROLL {  
  SCROLLBAR sb;  // in: originating scrollbar  
  SCROLLBAR_ACTION action;  // in: current action  
  S32 offset;  // in/out: current or new offset  
  S32 lineCoord;  // in: coordinate of line in root win space  
  U32 spare1;  // unused (reserved)  
  U32 spare2;  // unused (reserved)
} SCROLLBAR_SCROLL, *P_SCROLLBAR_SCROLL;
```

---

The passed offset is an estimate computed by the scrollbar based on the information obtained from `msgScrollbarProvideHorizInfo`.

If the client is unwilling to scroll to this offset, the client may scroll to a different offset. Be sure to set `pArgs->offset` to the new offset if it's different from the passed value.

---

**msgScrollbarProvideVertInfo**

Client should provide the document and view info.

Takes `P_SCROLLBAR_PROVIDE`, returns `STATUS`. Category: client responsibility.

```c
#define msgScrollbarProvideVertInfo MakeMsg(clsScrollbar, 9)
```

```c
typedef struct SCROLLBAR_PROVIDE {  
  SCROLLBAR sb;  // in: originating scrollbar  
  S32 viewLength;  // out: client-provided view width or height  
  S32 docLength;  // out: client-provided document width or height  
  S32 offset;  // out: client-provided current offset  
  U32 spare;  // unused (reserved)
} SCROLLBAR_PROVIDE, *P_SCROLLBAR_PROVIDE;
```

---

**msgScrollbarProvideHorizInfo**

Client should provide the document and view info.

Takes `P_SCROLLBAR_PROVIDE`, returns `STATUS`. Category: client responsibility.

```c
#define msgScrollbarProvideHorizInfo MakeMsg(clsScrollbar, 10)
```

```c
typedef struct SCROLLBAR_PROVIDE {  
  SCROLLBAR sb;  // in: originating scrollbar  
  S32 viewLength;  // out: client-provided view width or height  
  S32 docLength;  // out: client-provided document width or height  
  S32 offset;  // out: client-provided current offset  
  U32 spare;  // unused (reserved)
} SCROLLBAR_PROVIDE, *P_SCROLLBAR_PROVIDE;
```
Messages from Other Classes

**msgSave**
Causes an object to file itself in an object file.
Takes P_OBJ_SAVE, returns STATUS.

**Comments**
clsScrollbar responds by filing away its instance data.

**msgRestore**
Creates and restores an object from an object file.
Takes P_OBJ_RESTORE, returns STATUS.

**Comments**
clsScrollbar responds by restoring its instance data.

**msgWinRepaint**
Tells a window to repaint itself.
Takes nothing, returns STATUS. Category: descendant responsibility.

**Comments**
clsScrollbar responds by painting itself appropriately.

**msgWinLayoutSelf**
Tell a window to layout its children.
Takes P_WIN_METRICS, returns STATUS.

**Comments**
clsScrollbar does nothing if pArgs->options does not have wsLayoutResize turned on. Otherwise, it will set pArgs->bounds.size only for the dimensions for which shrinkwrapping is set. It will set pArgs->size.w and h to a value derived from msgBorderInsetToInnerRect and an internal constant (currently 13 points).

The visuals of the scrollbar are painted within the innerRect.

**See Also**
msgBorderInsetToInnerRect insets arbitrary rect to border rect.

**msgInputEvent**
Notification of an input event.
Takes P_INPUT_EVENT, returns STATUS.

**Comments**
clsScrollbar responds to input events by maintaining the state necessary to behave in the appropriate fashion. The types of input state a scrollbar can be in are:
- null
- pen up over an arrow
- pen down over an arrow
- pen up over the thumb
- pen down over the thumb
- dragging the thumb
- pen up over the shaft
- gesturing over the shaft
In particular, the scrollbar allows the normal msgControl*Preview protocol to ensue only when the pen is interacting with the scroll arrows.

**msgGWinGesture**
Called to process the gesture.
Takes P_GWIN_GESTURE, returns STATUS.

Comments 
clsScrollbar responds by returning stsMessageIgnored if the gesture has no meaning (e.g. xgsFlickUp on a horizontal scrollbar).
Otherwise, the scrollbar will fill out a SCROLLBAR_SCROLL struct and send either msgScrollbarVertScroll or msgScrollbarHorizScroll to the CONTROL_METRICS.client. Actually, the client will receive the message twice—once for the appropriate action, and once for the sbEndScroll action (although the second message with sbEndScroll may be dropped in the future).

**msgGWinComplete**
Causes the gesture to be completed.
Takes void, returns STATUS.
Comments 
clsScrollbar responds by clearing its internal state data left over from processing a gesture in the scrollbar shaft.

**msgControlBeginPreview**
Self-sent when msgPenDown is received.
Takes P_INPUT_EVENT, returns STATUS. Category: self-sent.

Comments 
clsScrollbar responds by returning stsControlCancelPreview if the penDown occurred in the shaft.
Otherwise, the scrollbar self-sends msgControlRepeatPreview so that at least one arrow scroll is done.
Note that the scrollbar won’t receive this message if the penDown occurred in the thumb, because in that case clsScrollbar’s response to msgInputEvent returned stsInputTerminate (after creating and starting an instance of clsTrack).

**msgControlAcceptPreview**
Self-sent when msgPenUp is received.
Takes P_INPUT_EVENT, returns STATUS. Category: self-sent.

Comments 
clsScrollbar responds by notifying its CONTROL_METRICS.client with msgScrollbar[Vert/Horiz]Scroll and an action of sbEndScroll.
Although one might expect this message to be sent when the pen is lifted from a scroll arrow, under normal circumstances a scrollbar will never receive this message. This is because clsScrollbar sees the msgInputEvent(msgPenUp) and self-sends msgGWinAbort to cancel the gesture (because the user really wasn’t gesturing over the repeating arrow). clsControl responds to msgGWinAbort by self-sending msgControlCancelPreview, which sends out the sbEndScroll.
**msgControlCancelPreview**
Self-sent when `style.previewGrab` is false and `msgPenExitDown` is received.


Comments: `clsScrollbar` responds by notifying its `CONTROL_METRICS.client` with `msgScrollbar[Vert/Horiz]Scroll` and an action of `sbEndScroll`.

**msgControlRepeatPreview**
Self-sent if `style.repeatPreview` is true.


Comments: `clsScrollbar` responds by notifying its `CONTROL_METRICS.client` with `msgScrollbar[Vert/Horiz]Scroll` and an action of `sbLine[Up/Down]` (if the scrollbar is vertical) or `sbLine[Left/Right]` (if horizontal).

If the client indicated that no scrolling took place (by not changing the `SCROLLBAR_SCROLL.offset` field), then the scrollbar will return `stsControlCancelRepeat`.

**msgTrackDone**
Sent by a tracker when it's done.

Takes `P_TRACK_METRICS`, returns `STATUS`. Category: client notification.

Comments: A `scrollBar` will receive this message when the user has lifted the pen after dragging the thumb.

If the `scrollBar`'s `style.direction` is `sbDirectionVertical`, the `scrollBar` will notify its `CONTROL_METRICS.client` with `msgScrollbarVertScroll` and one of these actions: `sbThumbUpDown`, `sbToTop`, or `sbToBottom`.

If the `scrollBar`'s `style.direction` is `sbDirectionHorizontal`, the `scrollBar` will notify its `CONTROL_METRICS.client` with `msgScrollbarHorizScroll` and one of these actions: `sbThumbLeftRight`, `sbToLeft`, or `sbToRight`. 
This file contains the API for clsSelChoiceMgr.

clsSelChoiceMgr inherits from clsManager.

Provides a choice manager that defines a protocol for managing the selection. Although clients may subclass clsSelChoiceMgr and add to or modify its behavior, there should be little reason to do so. clsSelChoiceMgr itself implements all of the standard UI for selectable choices.

Notes:

The selection choice manager works in a similar manner to the regular choice manager except it causes selection feedback to be displayed on the controls it manages. It also tells a client when to acquire the selection by sending msgSelChoiceMgrAcquireSel to the client. This message is sent every time one of the controls it manages turns on. selchmgr also sends msgSelChoiceMgrNullSel when someone programmatically turns off the selected control or sends the selchmgr msgChoiceMgrSetOnButton with an argument of objNull. The client should set the selection to null when it receives this message.

Note that msgNewDefaults to clsChoice results in a prototypical new struct whose values describe a button of contact style bsContactLockOn. This is correct for choices that always have one button on, but this is typically not what you'd want for selectable choices—the user should be able to deselect a selected button by tapping on it (so the choice then has no buttons on). To achieve this effect, do the equivalent of the following:

```
ObjCallWarn(msgNewDefaults, clsChoice, &choiceNew);
choiceNew.tkTable.pButtonNew->button.style.contact = bsContactToggle;
choiceNew.tkTable.manager = <uid of a selChoiceMgr>;
ObjCallRet(msgNew, clsChoice, &choiceNew, s);
```

See the documentation for msgTkTableChildDefaults in choice.h.

When a client receives msgSelYield from the selection manager it should send msgSelChoiceMgrNullCurrent to the selchmgr. This will cause it to turn off its currently chosen control and set its current choice to null. Here's how a client would typically respond to the relevant messages.

```
msgSelChoiceMgrAcquireSel:
<remember what kind of selection self will own
 by writing pArgs->id into self's instance data>
ObjCallRet(msgSelSelect, self, pNull, s);
<don't call ancestor>

msgSelChoiceMgrNullSel:
// The following will result in self receiving msgSelYield.
ObjCallRet(msgSelSetOwner, theSelectionManager, objNull, s);
<don't call ancestor>

msgSelYield:
// Ignore if self isn't the primary selection owner.
if ((BOOLEAN) (U32) pArgs == false)
  return ObjectCallAncestorCtx(ctx);
<get the choice by referencing the id value in self's instance data>
<get the manager of the choice via msgTkTableGetManager>
ObjCallRet(msgSelChoiceMgrNullCurrent, <manager>, pNull, s);
<clear the id field in self's instance data>
<don't call ancestor>
```
msgSelOptionTagOK:
<determine whether the kind of option sheet indicated by pArgs
(a TAG value) could be applied to the selection, and return
stsOK if so, stsFailed if not>
<dont call ancestor>
msgSelOptions:
<bring up an option sheet for the selection>
<dont call ancestor>

See sel.h for additional selection messages and their documentation.

 ifndef SELCHMGR_INCLUDED
#define SELCHMGR_INCLUDED

 ifndef CHMGR_INCLUDED
#include <chmgr.h>
endif

 ifndef XFER_INCLUDED
#include <xfer.h>
endif

 ifndef WIN_INCLUDED
#include <win.h>
endif

Common #defines and typedefs

typedef OBJECT SEL_CHOICE_MGR;

msgNew

Creates a selChoiceMgr object.

Takes P_SEL_CHOICE_MGR_NEW, returns STATUS. Category: class message.

Argumentstypedef struct SEL_CHOICE_MGR_NEW_ONLY {
   OBJECT client; // Object to send acquire/null messages to
   U32 id; // Id tag sent with acquire/null messages
   U32 spare;
} SEL_CHOICE_MGR_NEW_ONLY, *P_SEL_CHOICE_MGR_NEW_ONLY;
#define selChoiceMgrNewFields
   choiceMgrNewFields
   SEL_CHOICE_MGR_NEW_ONLY selChoiceMgr;

typedef struct SEL_CHOICE_MGR_NEW {
   selChoiceMgrNewFields
} SEL_CHOICE_MGR_NEW, *P_SEL_CHOICE_MGR_NEW;

typedef struct SEL_CHOICE_MGR_INFO {
   SEL_CHOICE_MGR selChoiceMgr; // Sender
   U32 id; // Client-specified id tag
   WIN button; // Current on button
} SEL_CHOICE_MGR_INFO, *P_SEL_CHOICE_MGR_INFO;

Comments

The fields you commonly set are:
pArgs->selChoiceMgr.client An object to manage the selection protocol. (Typically the app uid.)
pArgs->selChoiceMgr.id An id to distinguish among >1 selectable instances of clsChoice within the
client's domain.

msgNewDefaults

Initializes the SEL_CHOICE_MGR_NEW structure to default values.

Takes P_SEL_CHOICE_MGR_NEW, returns STATUS. Category: class message.
typedef struct SEL_CHOICE_MGR_NEW {
    selChoiceMgrNewFields
} SEL_CHOICE_MGR_NEW, *P_SEL_CHOICE_MGR_NEW;

Zeroes out pArgs->selChoiceMgr.

### msgSelChoiceMgrGetClient

Passes back the client uid held by the receiver.

Takes P_OBJECT, returns STATUS.

#define msgSelChoiceMgrGetClient MakeMsg(clsSelChoiceMgr, 1)

### msgSelChoiceMgrSetClient

Sets the client uid held by the receiver.

Takes OBJECT, returns STATUS.

#define msgSelChoiceMgrSetClient MakeMsg(clsSelChoiceMgr, 2)

### msgSelChoiceMgrGetId

Passes back the id held by the receiver.

Takes P_U32, returns STATUS.

#define msgSelChoiceMgrGetId MakeMsg(clsSelChoiceMgr, 3)

### msgSelChoiceMgrSetId

Sets the id held by the receiver.

Takes U32, returns STATUS.

#define msgSelChoiceMgrSetId MakeMsg(clsSelChoiceMgr, 4)

### msgSelChoiceMgrNullCurrent

Tells the receiver to clear the visuals and state of the choice.

Takes nothing, returns STATUS.

#define msgSelChoiceMgrNullCurrent MakeMsg(clsSelChoiceMgr, 5)

After receiving this message, the choice will have no current value. This message does not result in the sending of any side-effect messages such as msgSelYield.

### clsChoiceMgr Messages to which selChoiceMgs Respond

#### msgChoiceMgrGetOnButton

Gets the current on button. Passes back objNull if no button is on.

Takes P_UID, returns STATUS.
**msgChoiceMgrSetOnButton**
Sets the current on button.
Takes UID, returns STATUS.

**Comments**
Since the choiceMgr will use msgControlSetValue to turn the button on, that button's normal notification protocol will be invoked.
All buttons are turned off if message argument is objNull.

---

**Client Messages**

**msgSelChoiceMgrAcquireSel**
Sent to the client whenever a different button is selected.
Takes P_SEL_CHOICE_MGR_INFO, returns STATUS. Category: client responsibility.

```
#define msgSelChoiceMgrAcquireSel MakeMsg(clsSelChoiceMgr, 6)
```

**Message**

typedef struct SEL_CHOICE_MGR_INFO {
    SEL_CHOICE_MGR  selChoiceMgr;  // Sender
    U32              id;            // Client-specified id tag
    WIN              button;        // Current on button
} SEL_CHOICE_MGR_INFO, *P_SEL_CHOICE_MGR_INFO;

**Comments**
The client would typically respond by doing the following:
<remember what kind of selection self will own by writing pArgs->id into self's instance data>
ObjCallRet(msgSelSelect, self, pNull, s);
<don't call ancestor>

**msgSelChoiceMgrNullSel**
Sent to the client whenever a different button is selected.
Takes P_SEL_CHOICE_MGR_INFO, returns STATUS. Category: client responsibility.

```
#define msgSelChoiceMgrNullSel MakeMsg(clsSelChoiceMgr, 7)
```

**Message**

typedef struct SEL_CHOICE_MGR_INFO {
    SEL_CHOICE_MGR  selChoiceMgr;  // Sender
    U32              id;            // Client-specified id tag
    WIN              button;        // Current on button
} SEL_CHOICE_MGR_INFO, *P_SEL_CHOICE_MGR_INFO;

**Comments**
The client would typically respond by doing the following:
ObjCallRet(msgSelSetOwner, theSelectionManager, objNull, s);
<don't call ancestor>

As a consequence of this, the client would then receive msgSelYield.
This file contains the API definition for clsShadow.

clsShadow inherits from clsCustomLayout.

Implements a true shadow as a separate window underneath the shadowed window.

```c
ifndef SHADOW_INCLUDED
#define SHADOW_INCLUDED
#ifndef _INCLUDED
#include <clayout.h>
#endif
#endif
```

### Common #defines and typedefs

```c
typedef OBJECT SHADOW;
typedef struct SHADOW_STYLE {
    U16 trueShadow : 1,  // create a window for true-shadow effect
    spare : 15;        // unused (reserved)
} SHADOW_STYLE, *P_SHADOW_STYLE;
```

### Messages

#### msgNew

Creates an instance of clsShadow.

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

**Arguments**

```c
typedef struct SHADOW_NEW_ONLY {
    SHADOW_STYLE style;
    WIN borderWin;
    U32 spare;  // unused (reserved)
} SHADOW_NEW_ONLY, *P_SHADOW_NEW_ONLY;
```

**Comments**

If `pArgs->win.flags.style` has `wsTransparent` on, clsShadow will do the following:

- set `border.style.getDeltaWin` for `pArgs->shadow.borderWin` to true. This will forward any drag/resize operations on the border window to the shadow window.
- if `pArgs->shadow.style.trueShadow` is true the following is done:
  - if `pArgs->shadow.shadowWin` is `objNull`, an instance of clsBorder is created as the true shadow window.
  - self's `pArgs->border.style.shadow/resize` are copied to `shadowWin's border style. Also, border.style.getDeltaWin for `shadowWin` is set to true.
shadowWin is inserted as a child of self, underneath the borderWin, if any.

If pArgs->borderWin is not objNull, the wsShrinkWrapWidth/Height window flags of the borderWin are changed to match self's and the borderWin is inserted as a child of self, above the shadowWin.

**msgNewDefaults**

Initializes the SHADOW_NEW structure to default values.

Takes P_SHADOW_NEW, returns STATUS. Category: class message.

```c
typedef struct SHADOW_NEW {
    SHADOW NEW, *P_SHADOW_NEW;
}
```

Comments

Zeroes out pArgs->shadow and sets

- pArgs->win.flags.input = inputDisable | inputTransparent;
- pArgs->win.flags.style = wsTransparent | wsGrowBottom | wsGrowRight;
- pArgs->win.style.gestureEnable = false;
- pArgs->border.style.edge = bsEdgeAll;
- pArgs->border.style.shadow = bsShadowThickGray;
- pArgs->border.style.shadowGap = bsGapWhite;
- pArgs->border.style.borderlnk = bslnkGray66;
- pArgs->border.style.resize = bsResizeCorner;
- pArgs->border.style.drag = bsDragHoldDown;
- pArgs->border.style.top = bsTopUp;
- pArgs->customLayout.style.limitToRootWin = true;

Default SHADOW_STYLE:

trueShadow = false

**msgShadowGetStyle**

Passes back the current style values.

Takes P_SHADOW_STYLE, returns STATUS.

```c
#define msgShadowGetStyle MakeMsg(clsShadow, 1)
```

```c
typedef struct SHADOW STYLE {
    U16 trueShadow : 1, // create a window for true-shadow effect
    spare : 15; // unused (reserved)
} SHADOW STYLE, *P_SHADOW STYLE;
```

**msgShadowSetStyle**

Sets the style values.

Takes P_SHADOW_STYLE, returns STATUS.

```c
#define msgShadowSetStyle MakeMsg(clsShadow, 2)
```

```c
typedef struct SHADOW STYLE {
    U16 trueShadow : 1, // create a window for true-shadow effect
    spare : 15; // unused (reserved)
} SHADOW STYLE, *P_SHADOW STYLE;
```

Changes in self's border style are passed on to the borderWin and shadowWin.

**msgShadowGetBorderWin**

Passes back the border window.
Takes P_WIN, returns STATUS.

#define msgShadowGetBorderWin MakeMsg(clsShadow, 3)

**msgShadowSetBorderWin**
Sets the border window.
Takes WIN, returns STATUS.

#define msgShadowSetBorderWin MakeMsg(clsShadow, 4)

Comments
The new borderWin is altered as in msgNew.

**msgShadowGetShadowWin**
Passes back the shadow window.
Takes P_WIN, returns STATUS.

#define msgShadowGetShadowWin MakeMsg(clsShadow, 5)

Messages from Other Classes

**msgWinSetFlags**
Sets the window flags.
Takes P_WIN_METRICS, returns STATUS.

Comments
clsShadow will alter the borderWin's window flags to match the wsShrinkWrapWidth/Height flags of self.

**msgCstmLayoutGetChildSpec**
Passes back the current spec for the specified child.
Takes P_CSTM_LAYOUT_CHILD_SPEC, returns STATUS. Category: self-sent.

Comments
clsShadow responds by providing the custom layout constraints for borderWin and shadowWin.
The shadowWin is placed and sized to provide a gap area on the lower-left and upper-right.
The borderWin is placed above the bottom shadow of the shadowWin and sized width-wise to extend
to the left of the right shadow of the shadowWin.

**msgWinRepaint**
Tells a window to repaint itself.
Takes nothing, returns STATUS. Category: descendant responsibility.

Comments
If self has wsTransparent on, clsShadow prevents any painting by not calling ancestor and painting
nothing.
STDMSG.H

This file contains the API definition for the standard message package.

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

Introduction

The standard message package makes it easy to display error messages, modal dialog boxes, and progress notes. The package hides many of the details of finding resources and creating UI objects. The package uses clsNote to display messages. (See note.h.)

Messages are stored as strings in string array resources. A 32 bit value identifies the proper resource. For error messages the value is a STATUS; for dialog boxes and progress notes the value is a TAG constructed using the MakeDialogTag macro.

Road Map

To display a dialog box, use:
• StdMsg

To display an error message when you know about the error, use:
• StdError

To display an error message when you don't know about the error, use:
• StdUnknownError

To display a progress note, use:
• StdProgressUp
• StdProgressDown

To display messages extracted from a specified resource file or path, use:
• StdMsgRes
• StdErrorRes

To construct a customized message, use:
• StdMsgCustom

PenPoint-internal use only:
•StdSystemError
A Typical Scenario

[This scenario illustrates some features of the package that haven’t been described yet. See the sections "Button Definition" and "Text Substitution and Formatting" for more information.]

The first step in using the standard message package is to define a tag or status for each string:

```c
// #define stsFooError1 MakeStatus(clsFoo, 0)
// #define stsFooError2 MakeStatus(clsFoo, 1)
// #define tagFooDialog1 MakeDialogTag(clsFoo, 0)
// #define tagFooDialog2 MakeDialogTag(clsFoo, 1)
```

The next thing to do is to construct resources which contain the text strings. Standard message strings live in string array resources (see resfile.h). Application string arrays should reside in the application’s app.res file. (PenPoint’s string arrays reside in penpoint.res.)

There is one string array for error strings and another separate array for dialog box and progress strings. A single string array resource holds all of the strings for a given class.

Typically the string arrays are defined in a .rc file which is compiled with the PenPoint SDK’s resource compiler. The position of each string in the string array resource must match its tag or status index (starting from 0).

```c
static P_STRING errorClsFoo[] = {
    "This is the first error message.",
    "[Retry] [Cancel] This is the second error message. count: " 1d"};
static P_STRING dialogClsFoo[] = {
    "This is the first dialog message.",
    "[Go] [Stop] This is the second dialog message. str: " 1s"};
static RC_INPUT errorTabClsFoo = {
    resForStdMsgError(clsFoo), errorClsFoo, 0, resStringArrayResAgent};
static RC_INPUT dialogTabClsFoo = {
    resForStdMsgDialog(clsFoo), dialogClsFoo, 0, resStringArrayResAgent};
static RC_INPUT resInput[] = {
    &errorTabClsFoo,     // String array for std msg error strings
    &dialogTabClsFoo,    // String array for other std msg strings
    pNull};
```

Finally create a note by simply calling one of the appropriate function. This example uses StdMsg(), StdError() and StdUnknownError().

```c
buttonHit = StdMsg(tagFooDialog2, "String");
s = ObjectCall(...);
if (s < stsOK) {
    if (s == stsFooError1) {
        StdError(stsFooError1);
    } else {
        StdUnknownError(s);
    }
}
```

### Button Definition

Message strings may contain button definitions. A button definition is a substring enclosed in square brackets at the beginning of the message string. Any number of buttons may be defined but all buttons must appear at the front of the string. If no buttons are defined then a single "OK" button is created.

StdMsg(), StdError(), StdMsgRes(), StdErrorRes() and StdSystemError() return the button number that the user hit to dismiss the note. Button numbers start with 0. For example, a note constructed with the following string:
"[Button0] [Button1] [Button2] Here's your message!"
returns the value 1 if the user hits Button1. These functions might also return a negative error status if a
problem occurred inside the function.
See the section "A Typical Scenario" for an example.

- **Text Substitution and Formatting**
  Message strings may contain parameter substitutions, as defined in cmpstext.h. Text substitution also
  works inside the button substrings.
  See the section "A Typical Scenario" for an example.
  You can break your message up into paragraphs by putting 2 newlines at the paragraph breaks. For
  example:
  "Here's the first paragraph.

Here's the second one."

- **Progress Notes**
  Clients can put up a progress note to inform the user that a lengthy operation has begun, and take down
  the progress note to indicate that the operation has been completed.
  Cancellation of the operation is not supported in PenPoint 1.0. Progress notes do not have buttons.
  Here's an example of progress indication usage:

```c
SP_TOKEN token;
StdProgressUp(tagFooProgress1, &token, param1, param2);
... Lengthy operation ...
StdProgressDown(&token);
```

- **Searching for Resources**
  Most of the functions in this package search for resources as follows:
  - If the process is an application process (OSThisApp() returns non-null), then the application's
    resource list is searched. Otherwise **theSystemResFile** is searched.
  - If the desired resource is not found in the above resource files or lists, then **theServiceResList** is
    searched.
  The exceptions to this rule are:
  - **StdSystemError()**, which only checks **theSystemResFile**.
  - **StdMsgRes()**, which takes as one of its parameters the resource file or list to search.
  - **StdErrorRes()**, which takes as one of its parameters the resource file or list to search.

- **Note Titles and Reference Field**
  Notes will be titled "Note from {App}..." if the string was found in the app resource file, or "Note from
  PenPoint..." if the string was found in the system resource file.
  You can use **StdMsgCustom()** if you want to have some other title.
  Error messages will also have an additional line at the bottom of the note of the form:
  Reference: xxx-xxx
  where xxx-xxx is the status code that generated the error.
Customization of Standard Message Package Notes

StdMsgCustomO allows clients to customize a standard message package note. It returns the note object, without displaying it. Developers can modify this object as they wish and then display it themselves.

```c
#ifndef STDMSG_INCLUDED
#define STDMSG_INCLUDED
#ifndef GD_INCLUDE
#include <go.h>
#endif
#ifndef OSTYPES_INCLUDED
#include <ostypes.h>
#endif
#endif
```

Common #defines

Constructing Standard Message Tags

Use MakeStatusO (defined in go.h) to construct string tags for errors.

Use MakeDialogTagO to construct string tags for dialog and progress strings.

```c
#define MakeDialogTag(wkn, index) MakeIndexedResId(wkn, 1, index)
```

Constructing Standard Message Resource Ids

In a .rc file, use resForStdMsgDialog to construct the resource id for a class's dialog string array. Use resForStdMsgError to construct the resource id for a class's error string array.

See the section "A Typical Scenario" for an example.

```c
#define resForStdMsgDialog(wkn) MakeListResId(wkn, resGrpStdMsg, 1)
#define resForStdMsgError(wkn) MakeListResId(wkn, resGrpStdMsg, 0)
```

Public Functions

StdUnknownError

Displays an error message when the client doesn't recognize the error.

Returns STATUS.

**Function Prototype**

```c
STATUS CDECL StdUnknownError(
    STATUS status);
```

**Comments**

Use this function to display an error message when the error status is one that you don't pay special attention to.

StdUnknownError searches for an error message that matches the status parameter. If the specified status isn't found then a note with just the error code is displayed.

StdUnknownError does not allow parameter substitution or multiple command buttons. Any parameter substitution specifications in the text are replaced with "???". A single "OK" command button is always displayed.

See the section "A Typical Scenario" for an example. See the section "Searching for Resources" for a description of which resource files are searched.
**StdMsg**

Displays a dialog box from a resource file.

Returns S32.

Function Prototype

```c
S32 CDECL StdMsg (  
    const TAG tag,  
    ...  );
```

Comments

Use this function to display a dialog box.

StdMsg searches for an dialog string that matches the tag parameter. A dialog box with the message and buttons defined in the message string is displayed.

See the section "A Typical Scenario" for an example. See the section "Searching for Resources" for a description of which resource files are searched.

Like printf, this function takes a variable number of parameters. There is no error checking on the number and type of the parameters.

Return Value

- `stsResResourceNotFound` the specified tag is not found.
- `< stsOK` some other error occurred.
- `>= stsOK` number of button the user hit (0 based).

**StdError**

Displays an error message from a resource file.

Returns S32.

Function Prototype

```c
S32 CDECL StdError (  
    const STATUS status,  
    ...  );
```

Comments

Use this function to display an error message.

StdError searches for an error message string that matches the status parameter. A note with the message and buttons defined in the error message string is displayed. The note also contains an Error Code number line.

See the section "A Typical Scenario" for an example. See the section "Searching for Resources" for a description of which resource files are searched.

Like printf, this function takes a variable number of parameters. There is no error checking on the number and type of the parameters.

Return Value

- `stsResResourceNotFound` the specified tag is not found.
- `< stsOK` some other error occurred.
- `>= stsOK` number of button the user hit (0 based).

**StdSystemError**

For PenPoint internal use only. Displays an error message for a standard PenPoint error.

Returns S32.

Function Prototype

```c
S32 CDECL StdSystemError (  
    const STATUS status,  
    ...  );
```
StdSystemError searches theSystemResFile (penpoint.res) for an error message string that matches the status parameter. A note with the message and buttons defined in the string is displayed.

Like printf, this function takes a variable number of parameters. There is no error checking on the number and type of the parameters.

Return Value

stsResResourceNotFound  the specified tag is not found.
< stsOK  some other error occurred.
>= stsOK  number of button the user hit (0 based).

StdProgressUp
Displays a progress note from a resource file.

Returns STATUS.

Arguments

typedef struct SP_TOKEN {
    OBJECT uid;
    OS_MILLISECONDS startTime;
    U32 spare[8];
} SP_TOKEN, *P_SP_TOKEN;

Function Prototype

STATUS CDECL StdProgressUp(
    const TAG tag,
    P_SP_TOKEN pToken,
    ...);

Comments

Use this function to inform the user that a lengthy operation has started.

StdProgressUp searches for an dialog message that matches the tag parameter. A dialog box with the message string is displayed. This dialog box stay ups until StdProgressDown is called.

The pToken parameter, as filled in by StdProgressUp, must be passed to StdProgressDown. The client shouldn't touch it!

Example:

    SP_TOKEN token;
    StdProgressUp(tagFoo, &token, param1, param2);
    ...
    StdProgressDown(&token);

Progress notes do not contain a command bar. Any button definitions are ignored.

Like printf, this function takes a variable number of parameters. There is no error checking on the number and type of the parameters.

The pToken parameter must be the same as that passed to StdProgressUp().

See Also

StdProgressDown

StdProgressDown
Brings down a progress note that was put up with StdProgressUp().

Returns STATUS.

Function Prototype

STATUS CDECL StdProgressDown(
    P_SP_TOKEN pToken);

Comments

The pToken parameter must be the same as that passed to StdProgressUp().
See Also
StdProgressUp

**StdMsgRes**

Just like StdMsg() except that the resource path is specified.

Returns S32.

**Function Prototype**

```c
S32 CDECL StdMsgRes(
    OBJECT resFile,
    const TAG tag,
    ...);
```

**Comments**

Use StdMsgRes when you need the functionality of StdMsg, but need to look up the string in a specified resource file or resource list.

StdMsgRes searches the specified resource file or list for a dialog message string that matches the tag parameter. A dialog box with the message and buttons defined in the message string is displayed.

Like printf, this function takes a variable number of parameters. There is no error checking on the number and type of the parameters.

**Return Value**

- `stsResResourceNotFound` the specified tag is not found.
- `< stsOK` some other error occurred.
- `>= stsOK` number of button the user hit (0 based).

**See Also**
StdMsg

**StdErrorRes**

Just like StdError() except that the resource path is specified.

Returns S32.

**Function Prototype**

```c
S32 CDECL StdErrorRes(
    OBJECT resFile,
    const STATUS status,
    ...);
```

**Comments**

Use StdMsgError when you need the functionality of StdError, but need to look up the string in a specified resource file or resource list.

StdErrorRes searches the specified resource file or list for an error message string that matches the status parameter. A note with the message and buttons defined in the error message string is displayed.

Like printf, this function takes a variable number of parameters. There is no error checking on the number and type of the parameters.

**Return Value**

- `stsResResourceNotFound` the specified tag is not found.
- `< stsOK` some other error occurred.
- `>= stsOK` number of button the user hit (0 based).

**See Also**
StdError

**StdMsgCustom**

Creates a note object in the manner of StdMsg().
Returns OBJECT.

Function Prototype

```c
OBJECT CDECL StdMsgCustom(
    OBJECT resFile,
    const TAG tag,
    ...);
```

Comments

Use StdMsgCustom when you want to create a note using the facilities of the standard message package but need to customize the note before displaying it.

The client is responsible for displaying the note object. The note has `autoDestroy` on, so it self-destructs when dismissed.

StdMsgCustom allows the specification of a resource file or list to search. If `resFile` is `objNull` then searching occurs as described in the section "Searching for Resources." The tag parameter can either be a dialog tag (created with `MakeDialogTag()`) or an error status (created with `MakeStatus()`).

Here's an example:

```c
#define tagFooDialog1 MakeDialogTag(clsFoo, 0)

S32 buttonHit;
OBJECT note;

note = StdMsgCustom(objNull, tagFooDialog1, arg1, arg2);
if (note == objNull) {
    ... // Handle error, probably resource not found.
    goto error;
}
... // Customize the note.
ObjCallRet(msgNoteShow, note, &buttonHit, s);
```

Like `printf`, this function takes a variable number of parameters. There is no error checking on the number and type of the parameters.

Return Value

- `objNull`: No match, or some other error occurred.
This file contains the API for clsStringListBox.

clsStringListBox inherits from clsListBox.

Implements a listbox that behaves as a choice or a group of toggles.

As with clsListBox, the client supplies entry information on demand. With clsStringListBox, however, the client supplies strings, not windows. These strings are used to create instances of clsButton, and it is these buttons that are used as entry windows within the listBox.

A stringListBox may behave in one of three manners: as a list of individual toggles (as in clsToggleTable), as choice that has zero or one of its buttons ‘on’ at a time, or as a choice that always has exactly one of its buttons ‘on’ at once. When a stringListBox is behaving as a choice, its value is the ‘data’ field of the entry that is currently chosen.

```c
#ifndef STRLBOX_INCLUDED
#define STRLBOX_INCLUDED

#include <listbox.h>
#endif

Common #defines and typedefs

// String ListBox behavior styles (roles)
#define slbRoleToggles 0 // Act like a toggle table.
#define slbRoleChoice01 1 // Act like a choice (<=1 entries chosen)
#define slbRoleChoice1 2 // Act like a choice (always 1 entry chosen)

// String ListBox entry looks
#define slbLookInvert 0 // Chosen entries have inverted appearance.
#define slbLookDecorate 1 // Chosen entries have decorated appearance.

typedef struct {
    U16 role 4,
        look 2,
        dirty 1,
        spare 9;
} STRLB_STYLE, *P_STRLB_STYLE;

Default STRLB_STYLE:
    role = slbRoleToggles
    look = slbLookInvert
    dirty = false

msgNew

Creates a string listbox window.

Takes P_STRLB_NEW, returns STATUS. Category: class message.

Arguments
typedef struct {
    STRLB_STYLE style; // overall style
    U32 value; // initial value (if slbRoleChoice01
                 // or slbRoleChoice1)
    U32 spare; // reserved
} STRLB_NEW_ONLY, *P_STRLB_NEW_ONLY;

The value is the 'data' field of the entry that is currently chosen.
#define stringListBoxNewFields
    listBoxNewFields
    STRLB_NEWONLY
    stringListBox;

typedef struct {
    stringListBoxNewFields
} STRLB_NEW, *P_STRLB_NEW;

#define staStrListBoxNoValue
MakeStatus(clsStringListBox, 1)

Comments
The fields you commonly set are:

- pArgs->stringListBox.style.role  overall behavior
- pArgs->stringListBox.style.look  entry looks
- pArgs->stringListBox.value  initial value

---

**msgNewDefaults**

Initializes the STRLB_NEW structure to default values.

Takes P_STRLB_NEW, returns STATUS. Category: class message.

```c
typedef struct {
    stringListBoxNewFields
} STRLB_NEW, *P_STRLB_NEW;

msgStrListBoxGetStyle
Passes back the style of the receiver.

Takes P_STRLB_STYLE, returns STATUS.

```c
typedef struct {
    U16    role : 4,  // Overall behavior.
    look   : 2,  // Controls looks of entries.
    dirty  : 1,  // Dirty status (ref. control.h)
    spare  : 9;  // reserved
} STRLB_STYLE, *P_STRLB_STYLE;
```

---

**msgStrListBoxGetDirty**

Passes back true if the listbox has been altered since dirty was set false.

Takes P_BOOLEAN, returns STATUS.

```c
#define msgStrListBoxGetDirty
MakeMsg(clsStringListBox, 2)
```

---

**msgStrListBoxSetDirty**

Sets the dirty bit of a string listbox.

Takes BOOLEAN, returns STATUS.

```c
#define msgStrListBoxSetDirty
MakeMsg(clsStringListBox, 3)
```

Comments
The receiver will send msgControlSetDirty(pArgs) to every entry window.
msgStrListBoxGetValue

Passes back the value of a string listbox.

Takes P_U32, returns STATUS.

#define msgStrListBoxGetValue MakeMsg(clsStringListBox, 4)

Comments

The value is the data field of the entry that is currently chosen. This message may be used on instances whose role is either slbRoleChoice01 or slbRoleChoice1. For instances whose role is slbRoleToggles, use msgListBoxEnum with enum.flags set to lbSelected.

Return Value

stsFailed the role is set to slbRoleToggles.

stsStrListBoxNoValue there's no entry selected.

msgStrListBoxSetValue

Sets the value of a string listbox whose role is one of slbRoleChoice*.

Takes U32, returns STATUS.

#define msgStrListBoxSetValue MakeMsg(clsStringListBox, 5)

Comments

Will deselect any selected entry if the arg is maxU32 and the role is set to slbRoleChoice1. For instances whose role is slbRoleToggles, send as many msgListBoxSetEntry messages as required.

Return Value

stsFailed the role is set to slbRoleToggles.

Client Messages

msgStrListBoxProvideString

This message requests the client (or subclass) to provide a string.

Takes P_STRLB_PROVIDE, returns STATUS. Category: self-sent/client responsibility.

#define msgStrListBoxProvideString MakeMsg(clsStringListBox, 6)

typedef struct {
  OBJECT strListBox; // In: requestor
  U16 position; // In: position of requested entry
  P_CHAR pString; // Out: a 256 byte buffer for the string
  U32 data; // Out: data for the entry
  U32 spare; // reserved
} STRLB_PROVIDE, *P_STRLB_PROVIDE;

Comments

This message is sent when clsStringListBox receives msgListBoxProvideEntry.

The string listbox is constructing an entry to be put into the listbox, and it needs the string and some data for the entry. The client (or subclass) should copy the string bytes into the pString buffer, and set the data field as desired.

msgStrListBoxProvideString is sent first to the string listbox itself. If the message reaches the standard clsStringListBox message handler, this message is forwarded on to the client of the listbox.

A string listbox will send this message even when the position it's asking for is >= the number of entries specified for the listBox (same behavior as msgListBoxProvideEntry). In this case, the client is free to return a non-zero status value, indicating to the string listbox that no entry should be created for that position. Providing another string in this case allows A listBox will send this message even when the position it's asking for is >= the number of entries specified for the listBox. In this case, the client is free to return a non-zero status value, indicating to the listBox that no entry should be created for that
position. Providing another entry window in this case allows the user to create new entries by merely scrolling past the end of the list.

Subclasses of clsStringListBox may choose to respond to msgStrListBoxProvideString, or bypass this mechanism altogether and respond instead to msgListBoxProvideEntry.

**msgStrListBoxNotify**
This message is sent out whenever the value of a string listbox changes.

```c
#define msgStrListBoxNotify MakeMsg(clsStringListBox, 7)
```

Comments
The pArgs will be undefined when role is set to slbRoleToggles (use msgListBoxEnum with enum.flags set to lbSelected).

clsStringListBox responds by forwarding the message to the client of the listbox.

**Messages from Other Classes**

**msgSave**
Causes an object to file itself in an object file.
Takes P_OBJ_SAVE, returns STATUS.

Comments
clsStringListBox responds by filing away its style and value. Note that clsListBox will have filed its data first according to the value of LIST_BOX_STYLE.

**msgRestore**
Creates and restores an object from an object file.
Takes P_OBJ_RESTORE, returns STATUS.

Comments
clsStringListBox responds by restoring its style and value.

**msgWinSend**
Sends a message up a window ancestry chain.
Takes P_WIN_SEND, returns STATUS.

Comments
clsStringListBox responds when pArgs->msg is msgButtonBeginPreview, msgButtonCancelPreview, or msgButtonDone. If pArgs->msg is anything else, clsStringListBox just returns the result of calling its ancestor.

For these three messages, clsStringListBox will make the set of entry windows act as a group (as does clsChoiceMgr) and return stsManagerContinue.

Return Value
stsManagerContinue returned for one of the above three messages.

**msgListBoxProvideEntry**
Self-sent when a listBox needs a window for display.
Takes P_LIST_BOX_ENTRY, returns STATUS. Category: self-sent/client responsibility.
clsStringListBox responds by self-sending `msgStrListBoxProvideString`, using the resulting information to create an instance of clsButton, and passing back the new button in pArgs->win.

---

**msgListBoxAppendEntry**

Appends an entry to the list box after the specified position.

Takes `P_LIST_BOX_ENTRY`, returns `STATUS`.

Comments: clsStringListBox responds by keeping its state in synch--if the position that is currently on is greater than the new entry, it's incremented.

---

**msgListBoxInsertEntry**

Insert an entry to the list box before the specified position.

Takes `P_LIST_BOX_ENTRY`, returns `STATUS`.

Comments: clsStringListBox responds by keeping its state in synch--if the position that is currently on is greater than the new entry, it's incremented.

---

**msgListBoxRemoveEntry**

Removes an entry from the list box.

Takes U16, returns `STATUS`.

Comments: clsStringListBox responds by keeping its state in synch--if the position that is currently on is less than the new entry, it's decremented.

If the entry being removed is the current 'on' button, the receiver sets its current value to zero (if the role is slbRoleChoice1) or to `maxU32` (if the role is slbRoleChoice1). `msgStrListBoxNotify` will be sent.

---

**msgListBoxSetEntry**

Sets an entry's information.

Takes `P_LIST_BOX_ENTRY`, returns `STATUS`.

Comments: clsStringListBox responds by setting the tag and data for any new replacement entry window.
This file contains the API definition for clsScrollWin.

clsScrollWin inherits from clsBorder.

A scrollWin positions, sizes, and displays a client window (optionally part of a "deck" of other child windows) together with optional scrollbars, and can scroll the client window by repositioning it.

**Debugging Flags**

The clsScrollWin debugging flag is 'r'. Defined values are:

```c
#ifndef SWIN_INCLUDED
#define SWIN_INCLUDED
#include <sbar.h>
#endif
```

**Common #defines and typedefs**

```c
typedef OBJECT SCROLL_WIN;
```

**Client styles for scrollbar client**

```c
#define swClientScrollWin 0 // scrollWin is the scrollbar client
#define swClientWin 1 // clientWin is the scrollbar client
#define swClientOther 2 // scrollWin will not set the client
#define swClientNone 3 // unused (reserved)
```

**Alignment styles**

```c
#define swAlignLeft 0 // left-justified
#define swAlignCenter 1 // centered
#define swAlignRight 2 // right-justified
#define swAlignTop swAlignLeft // top-justified
#define swAlignBottom swAlignRight // bottom-justified
#define swAlignSelf 3 // clientWin will align itself
```

**Forward styles**

```c
#define swForwardNone 0 // don't forward anything
#define swForwardGesture 1 // forward msgGWinGesture
#define swForwardXList 2 // forward msgGWinXList
```

```c
typedef struct SCROLL_WIN_STYLE {
    UI6 vertScrollbar : 1, // vertical scrollbar on/off
    horizScrollbar : 1, // horizontal scrollbar on/off
    autoVertScrollbar : 1, // vert scrollbar on/off based on clientWin
    autoHorizScrollbar : 1, // horiz scrollbar on/off based on clientWin
    maskScrollbars : 1, // mask out vertScrollbar and horizScrollbar
    expandChildWidth : 1, // expand the child's width to avail width
    expandChildHeight : 1, // expand the child's height to avail height
    contractChildWidth : 1, // contract the child's width to avail width
    contractChildHeight : 1, // contract the child’s height to avail height
} SCROLL_WIN_STYLE;
```
getDelta : 1,  // send msgScrollWinProvideDelta to client
getSize : 1,  // send msgScrollWinProvideSize
wideVertScrollbar : 1,  // make the vertical scrollbar wide
wideHorizScrollbar : 1,  // make the horizontal scrollbar wide
forward : 2,  // what to forward from margins to clientWin
maskAll : 1;  // mask out maskScrollbars
U16 xAlignment : 2,  // x Alignment if innerWin wider than clientWin
yAlignment : 2,  // y Alignment if innerWin taller than clientWin
vertClient : 2,  // choice of vertical sb client
horizClient : 2,  // choice of horizontal sb client
xAlignRigorous : 1,  // use xAlignment continuously
yAlignRigorous : 1,  // use yAlignment continuously
private1 : 1;  // private
spare1 : 5;  // unused (reserved)
U16 spare2 : 16;  // unused (reserved)

} SCROLL_WIN_STYLE, *P_SCROLL_WIN_STYLE;

Default: SCROLL_WIN_STYLE:

getDelta = false
vertScrollbar = false
horizScrollbar = false
autoVertScrollbar = true
autoHorizScrollbar = true
expandChildWidth = false
expandChildHeight = false
vertClient = swClientScrollWin
horizClient = swClientScrollWin
xAlignment = swAlignLeft
yAlignment = swAlignTop
getSize = false
contractChildWidth = false
contractChildHeight = false
forward = swForwardNone
maskScrollbars = false
xAlignRigorous = true
yAlignRigorous = true

The x- and yAlignment styles are used primarily when the innerWin is wider/taller than the clientWin. However, they are also used when a clientWin that is wider/taller than the innerWin is changing size. In this case, the innerWin alters the origin to compensate for the size change so that the appropriate edge of the clientWin is held fixed (either by doing the math itself or sending out msgScrollWinAlign if the alignment is set to swAlignSelf). An example: a top-aligned clientWin of height 100 in an innerWin of height 50 is growing by 20. The innerWin would subtract 20 from the clientWin's new origin.y.

Clients can disable the adjustments that occur in the second case (clientWin is wider/taller than the innerWin) by setting the appropriate x- or yAlignRigorous flag to false.

typedef struct SCROLL_WIN_METRICS {
    SCROLL_WIN_STYLE style;  // style bits
    OBJECT client;  // for msgScrollWinProvideDelta
    WIN clientWin;  // current window to scroll
    U16 colDelta, rowDelta;  // metrics in device units
    U32 spare1;  // unused (reserved)
    U32 spare2;  // unused (reserved)
} SCROLL_WIN_METRICS, *P_SCROLL_WIN_METRICS;

typedef struct SCROLL_WIN_DELTA {
    SCROLL_WIN scrollWin;  // in: requesting scroll win
    SCROLLBAR_ACTION action;  // in: action to resolve
    S32 offset;  // in: current or new offset
    RECT32 viewRect;  // in/out: viewable portion of clientWin
    S32 lineCoord;  // in: line coordinate, if any
    U32 spare;  // unused (reserved)
} SCROLL_WIN_DELTA, *P_SCROLL_WIN_DELTA;
Messages

**msgNew**

Creates a scrollWin.

Takes P_SCROLL_WIN_NEW, returns STATUS. Category: class message.

typedef SCROLL_WIN_METRICS SCROLL_WIN_NEW_ONLY, *P_SCROLL_WIN_NEW_ONLY;
define scrollWinNewFields

borderNewFields

SCROLL_WIN_NEWONLY scrollWin;

Arguments
typedef struct SCROLL_WIN_NEW {

scrollWinNewFields

} SCROLL_WIN_NEW, *P_SCROLL_WIN_NEW;

Comments

The fields you commonly set are:

pArgs->scrollWin.style appropriate style values

pArgs->scrollWin.clientWin a window to scroll

**msgNewDefaults**

Initializes the SCROLL_WIN_NEW structure to default values.

Takes P_SCROLL_WIN_NEW, returns STATUS. Category: class message.

Message Arguments

typedef struct SCROLL_WIN_NEW {

scrollWinNewFields

} SCROLL_WIN_NEW, *P_SCROLL_WIN_NEW;

Comments

Zeroes out pArgs->scrollWin and sets:

pArgs->win.flags.style = wsSendFile | wsClipChildren | wsClipSiblings;

pArgs->win.flags.style &= ~wsParentClip;

pArgs->scrollWin.style.autoVertScrollbar = true;

pArgs->scrollWin.style.autoHorizScrollbar = true;

pArgs->scrollWin.style.xAlignRigorous =

pArgs->scrollWin.style.yAlignRigorous = true;

pArgs->scrollWin.colDelta = 10;

pArgs->scrollWin.rowDelta = 10;

**msgScrollWinGetStyle**

Passes back the current style values.

Takes P_SCROLL_WIN_STYLE, returns STATUS.

#define msgScrollWinGetStyle MakeMsg(clsScrollWin, 13)

Message Arguments

typedef struct SCROLL_WIN_STYLE {

U16 vertScrollbar : 1, // vertical scrollbar on/off

horizScrollbar : 1, // horizontal scrollbar on/off

autoVertScrollbar : 1, // vert scrollbar on/off based on clientWin

autoHorizScrollbar : 1, // horiz scrollbar on/off based on clientWin

maskScrollbars : 1, // mask out vertScrollbar and horizScrollbar

extendChildWidth : 1, // expand the child’s width to avail width

extendChildHeight : 1, // expand the child’s height to avail height

contractChildWidth : 1, // contract the child’s width to avail width

contractChildHeight : 1, // contract the child’s height to avail height

getDelta : 1, // send msgScrollWinProvideDelta to client

getSize : 1, // send msgScrollWinProvideSize
wideVertScrollbar : 1, // make the vertical scrollbar wide
wideHorizScrollbar : 1, // make the horizontal scrollbar wide
forward : 2, // what to forward from margins to clientWin
maskAll : 1; // mask out maskScrollbars
U16 xAlignment : 2, // x Alignment if innerWin wider than clientWin
yAlignment : 2, // y Alignment if innerWin taller than clientWin
vertClient : 2, // choice of vertical sb client
horizClient : 2, // choice of horizontal sb client
xAlignRigorous : 1, // use xAlignment continuously
yAlignRigorous : 1, // use yAlignment continuously
private1 : 1, // private
spare1 : 5; // unused (reserved)
U16 spare2 : 16; // unused (reserved)
} SCROLL_WIN_STYLE, *P_SCROLL_WIN_STYLE;

msgScrollWinSetStyle
Sets the style values.
Takes P_SCROLL_WIN_STYLE, returns STATUS.

#define msgScrollWinSetStyle MakeMsg(clsScrollWin, 14)

typedef struct SCROLL_WIN_STYLE {
  U16 vertScrollbar : 1, // vertical scrollbar on/off
  horizScrollbar : 1, // horizontal scrollbar on/off
  autoVertScrollbar : 1, // vert scrollbar on/off based on clientWin
  autoHorizScrollbar : 1, // horiz scrollbar on/off based on clientWin
  maskScrollbars : 1, // mask out vertScrollbar and horizScrollbar
  expandChildWidth : 1, // expand the child's width to avail width
  expandChildHeight : 1, // expand the child's height to avail height
  contractChildWidth : 1, // contract the child's width to avail width
  contractChildHeight : 1, // contract the child's height to avail height
  getDelta : 1, // send msgScrollWinProvideDelta to client
  getSize : 1, // send msgScrollWinProvideSize
  wideVertScrollbar : 1, // make the vertical scrollbar wide
  wideHorizScrollbar : 1, // make the horizontal scrollbar wide
  forward : 2, // what to forward from margins to clientWin
  maskAll : 1; // mask out maskScrollbars
  U16 xAlignment : 2, // x Alignment if innerWin wider than clientWin
  yAlignment : 2, // y Alignment if innerWin taller than clientWin
  vertClient : 2, // choice of vertical sb client
  horizClient : 2, // choice of horizontal sb client
  xAlignRigorous : 1, // use xAlignment continuously
  yAlignRigorous : 1, // use yAlignment continuously
  private1 : 1, // private
  spare1 : 5; // unused (reserved)
  U16 spare2 : 16; // unused (reserved)
} SCROLL_WIN_STYLE, *P_SCROLL_WIN_STYLE;

The scrollWin self-sends msgWinSetLayoutDirty(true). It is the caller's responsibility to re-layout the scrollWin.

msgScrollWinGetMetrics
Passes back the metrics.
Takes P_SCROLL_WIN_METRICS, returns STATUS.

#define msgScrollWinGetMetrics MakeMsg(clsScrollWin, 1)
typedef struct SCROLL_WIN_METRICS {
    SCROLL_WIN_STYLE style;  // style bits
    OBJECT client;  // for msgScrollWinProvideDelta
    WIN clientWin;  // current window to scroll
    U16 colDelta, rowDelta;  // metrics in device units
    U32 spare1;  // unused (reserved)
    U32 spare2;  // unused (reserved)
} SCROLL_WIN_METRICS, *P_SCROLL_WIN_METRICS;

**msgScrollWinSetMetrics**
Sets the metrics.
Takes P_SCROLL_WIN_METRICS, returns STATUS.

#define msgScrollWinSetMetrics MakeMsg(clsScrollWin, 2)

**msgScrollWinGetClientWin**
Passes back the current clientWin.
Takes P_WIN, returns STATUS.

#define msgScrollWinGetClientWin MakeMsg(clsScrollWin, 3)

Comments
The current clientWin is the last window to be shown using msgScrollWinShowClientWin.

**msgScrollWinShowClientWin**
Sets the current clientWin; the specified window is be made visible.
Takes WIN, returns STATUS.

#define msgScrollWinShowClientWin MakeMsg(clsScrollWin, 4)

Comments
If P_ARGS is not a child of the scrollWin's inner window, msgScrollWinAddClientWin is self-sent followed by msgWinLayout.

**msgScrollWinAddClientWin**
Adds another clientWin, inserting the specified window as a child of the scrollWin's inner window.
Takes WIN, returns STATUS.

#define msgScrollWinAddClientWin MakeMsg(clsScrollWin, 11)

Comments
The specified window is set to be invisible (window flag wsVisible off).

**msgScrollWinRemoveClientWin**
Extracts the specified child window from the scrollWin.
Takes WIN, returns STATUS.

#define msgScrollWinRemoveClientWin MakeMsg(clsScrollWin, 12)
**msgScrollWinGetVertScrollbar**

Passes back the vertical scrollbar.

Takes P_WIN, returns STATUS.

```c
#define msgScrollWinGetVertScrollbar MakeMsg(clsScrollWin, 6)
```

**msgScrollWinGetHorizScrollbar**

Passes back the horizontal scrollbar.

Takes P_WIN, returns STATUS.

```c
#define msgScrollWinGetHorizScrollbar MakeMsg(clsScrollWin, 7)
```

**msgScrollWinGetInnerWin**

Passes back the inner window of the scrollWin.

Takes P_WIN, returns STATUS.

```c
#define msgScrollWinGetInnerWin MakeMsg(clsScrollWin, 9)
```

**msgScrollWinProvideDelta**

Self-sent when style.getDelta is set to true so that descendant or client can normalize the scroll if desired.

Takes P_SCROLL_WIN_DELTA, returns STATUS. Category: descendant/client responsibility.

```c
#define msgScrollWinProvideDelta MakeMsg(clsScrollWin, 5)
```

**Message Structure**

```c
typedef struct SCROLL_WIN_DELTA {
    SCROLL_WIN scrollWin; // in: requesting scroll win
    SCROLLBAR_ACTION action; // in: action to resolve
    S32 offset; // in: current or new offset
    RECT32 viewRect; // in/out: viewable portion of clientWin
    S32 lineCoord; // in: line coordinate, if any
    U32 spare; // unused (reserved)
} SCROLL_WIN_DELTA, *P_SCROLL_WIN_DELTA;
```

**Comments**

clsScrollWin responds by forwarding this message to the current clientWin. If you receive this message, you can send msgScrollWinGetDefaultDelta to pArgs->scrollWin to fill out pArgs with the default scrollWin response.

**msgScrollWinProvideSize**

Self-sent to determine bubble location and size.

Takes P_SCROLL_WIN_SIZE, returns STATUS. Category: descendant/client responsibility.

```c
#define msgScrollWinProvideSize MakeMsg(clsScrollWin, 10)
```

**Message Structure**

```c
typedef struct SCROLL_WIN_SIZE {
    SCROLL_WIN scrollWin; // in: requesting scroll win
    SIZE32 viewSize; // out: desired view size (device units)
    SIZE32 docSize; // out: logical doc size (device units)
    U32 spare; // unused (reserved)
} SCROLL_WIN_SIZE, *P_SCROLL_WIN_SIZE;
```

**Comments**

clsScrollWin responds by forwarding to the current clientWin (if style.getSize is true), or sending msgWinGetDesiredSize to the current clientWin (if style.getSize is false). In the latter case if there is no current clientWin, clsScrollWin uses 0 for docSize.
A clientWin responding to msgScrollWinProvideSize should fill out pArgs->viewSize and pArgs->docSize.

msgScrollWinCheckScrollbars
Determines whether the on/off state of either scrollbar needs to change and passes back the result.
Takes P_BOOLEAN, returns STATUS.
#define msgScrollWinCheckScrollbars MakeMsg(clsScrollWin, 15)

Clients wishing to fix up the states should dirty the layout of the scrollWin and then send msgWinLayout.

msgScrollWinAlign
Sent to client when style.xAlignment or style.yAlignment is swAlignSelf.
Takes P_SCROLL_WIN_ALIGN, returns STATUS. Category: client responsibility.
#define msgScrollWinAlign MakeMsg(clsScrollWin, 16)

clsScrollWin sends this message to the scrollWin's client or clientWin if the client is objNull. This message is sent when the child window changes size or the scrollWin inner window changes size. See the comment after "Default SCROLL_WIN_STYLE" for a further description of alignment.

msgScrollWinGetDefaultDelta
Compute the default response to msgScrollWinProvideDelta.
Takes P_SCROLL_WIN_DELTA, returns STATUS.
#define msgScrollWinGetDefaultDelta MakeMsg(clsScrollWin, 17)

You can send this message to a scrollWin to compute the default scroll values for a given scrolling action.

See Also
msgScrollWinProvideDelta

msgScrollWinRefreshSize
Informs the receiver that msgScrollWinProvideSize would now return different size.
Takes P_SIZE32, returns STATUS.
#define msgScrollWinRefreshSize MakeMsg(clsScrollWin, 18)
A client would send this to a scrollWin when the scrollWin has style.getSize set true and the client would now return different size in response to msgScrollWinProvideSize. The client passes the current size, and the scrollWin sends msgScrollWinProvideSize to the client (or the clientWin if that's null), then adjust the positions as if the clientWin had changed size.

clsScrollWin just returns stsOK if style.getSize is false.

Messages from other classes

**msgSave**
Causes an object to file itself in an object file.
Takes P_OBJ_SAVE, returns STATUS.

Comments
clsScrollWin responds by filing away all its state, including any child windows that have wsSendFile turned on (wsSendFile is the default for clsBorder and its descendents).

**msgRestore**
Creates and restores an object from an object file.
Takes P_OBJ_RESTORE, returns STATUS.

Comments
clsScrollWin responds by restoring all of its state, including the child windows that were filed with the last msgSave.

**msgWinLayoutSelf**
Tell a window to layout its children.
Takes P_WIN_METRICS, returns STATUS.

Comments
The scrollWin first gets the dimensions of its current clientWin by using msgWinGetDesiredSize. If there is no current clientWin, the scrollWin uses a width and height of 0 in its computations.

If the scrollWin did not shrinkwrap around the current clientWin, then the expandChild* and contractChild* styles come into play. If the clientWin's width is less than the width of the scrollWin's inner window (a direct child of the scrollWin that serves as a clipping window) and expandChildWidth is true, then the clientWin's width is expanded to fit. If the clientWin's width is greater than the inner window's and contractChildWidth is true, then the clientWin's width is reduced to fit. These rules hold for the height as well.

Finally, if the clientWin's (possibly modified) width is still less than the inner window's, then the xAlignment style is used to place the clientWin within the inner window. This is also done in the vertical direction using yAlignment.

The scrollWin adds or remove a vertical scrollBar as necessary if style.autoVertScrollBar is on, and the same is done for the horizontal direction (when both style.maskScrollbars and style.maskAll are off).

**msgWinSetFlags**
Sets the window flags.
Takes P_WIN_METRICS, returns STATUS.

Comments
clsScrollWin responds by first propagating the shrinkwrap values to the inner window, then calling its ancestor.
**msgWinSend**

Sends a message up a window ancestry chain.

Takes **WIN_SEND**, returns **STATUS**.

Comments

*clsScrollWin* responds by checking to see if **pArgs->msg** is **msgScrollbarUpdate**. If so, the *scrollWin* sends **msgScrollbarUpdate** to both of its scrollbars and then return **stsOK**. If not, then the *scrollWin* just calls its ancestor.

**msgGWinGesture**

Self-sent to process the gesture.

Takes **P_GWIN_GESTURE**, returns **STATUS**.

Comments

If there is a current **clientWin** and **style.forward** is **swForwardGesture**, then the **pArgs** are transformed to the **clientWin**, and the **clientWin** is sent **msgGWinGesture**. The *scrollWin* returns the resulting status to the caller.

Otherwise, the *scrollWin* compares the **pArgs->msg** with the state of the corresponding scrollbar. If the message is not a scrolling gesture, then the *scrollWin* returns **stsMessageIgnored**. If it is a vertical scrolling gesture and the vertical scrollbar is not active, then the *scrollWin* returns **stsOK**. Finally, if the message is a vertical scrolling gesture and the vertical scrollbar is active, the *scrollWin* transforms the **pArgs** to the scrollbar's space and return the result of sending **msgGWinGesture** to the scrollbar (unless the **msgGWinGesture** originated with the scrollbar, i.e. **pArgs->uid == the scrollbar**) -- in this case the *scrollWin* returns **stsMessageIgnored**. This processing is also done for horizontal scrolling gestures and the horizontal scrollbar.

The above processing also is done whenever the *scrollWin*’s inner window receives **msgGWinGesture**.

Return Value

**stsOK** a scrolling gesture would have had to be sent to an inactive scrollbar.

**stsMessageIgnored** not a scrolling gesture, or message originated with the scrollbar to which **msgGWinGesture** would be sent.

**msgGWinForwardedGesture**

Message received when object is forwarded a gesture.

Takes **P_GWIN_GESTURE**, returns **STATUS**.

Comments

The *scrollWin* compares the **pArgs->msg** with the state of the corresponding scrollbar. If the message is not a scrolling gesture, then the *scrollWin* returns **stsMessageIgnored**. If it is a vertical scrolling gesture and the vertical scrollbar is not active, then the *scrollWin* returns **stsOK**. Finally, if the message is a vertical scrolling gesture and the vertical scrollbar is active, the *scrollWin* transforms the **pArgs** to the scrollbar’s space and return the result of sending **msgGWinGesture** to the scrollbar (unless the **msgGWinGesture** originated with the scrollbar, i.e. **pArgs->uid == the scrollbar**) -- in this case the *scrollWin* returns **stsMessageIgnored**. This processing is also done for horizontal scrolling gestures and the horizontal scrollbar.

The above processing also is done whenever the *scrollWin*’s inner window receives **msgGWinGesture**.

Return Value

**stsOK** a scrolling gesture would have had to be sent to an inactive scrollbar.

**stsMessageIgnored** not a scrolling gesture, or message originated with the scrollbar to which **msgGWinGesture** would be sent.
**msgGWinXList**

Call back to announce gesture translation completed.

Takes P_XLIST, returns STATUS.

**Comments**

If there is no current clientWin, or style.forward is not swForwardXList, then the scrollWin just calls its ancestor.

Otherwise, the scrollWin transforms the pArgs to the clientWin and return the result of sending msgGWinXList to the clientWin.

The above processing also is done whenever the scrollWin’s inner window receives msgGWinXList.

**msgScrollbarVertScroll**

Client should perform vertical scroll.

Takes P_SCROLLBAR_SCROLL, returns STATUS. Category: client responsibility.

**Comments**

Responding to this message is one of the key functions that scrollWins provide. Since the default scrollWin style.vertClient and .horizClient are both swClientScrollWin, it is usually the case that the scrollbars send their scrolling messages to the scrollWin.

If there is no current clientWin, or the pArgs->action is sbEndScroll, the scrollWin just returns stsOK.

If style.getDelta is true, the scrollWin sends msgScrollbarProvideDelta to the client (if that is non-null) or the clientWin (if the client was null). Otherwise, the scrollWin uses metrics.rowDelta for the sbLine* actions, and the inner window’s height - metrics.rowDelta for the sbPage* actions.

Once the scrollWin has determined the amount to scroll, it sends msgWinDelta to the clientWin.

**msgScrollbarHorizScroll**

Client should perform horizontal scroll.

Takes P_SCROLLBAR_SCROLL, returns STATUS. Category: client responsibility.

**Comments**

Responding to this message is one of the key functions that scrollWins provide. Since the default scrollWin style.vertClient and .horizClient are both swClientScrollWin, it is usually the case that the scrollbars send their scrolling messages to the scrollWin.

If there is no current clientWin, or the pArgs->action is sbEndScroll, the scrollWin just returns stsOK.

If style.getDelta is true, the scrollWin sends msgScrollbarProvideDelta to the client (if that is non-null) or the clientWin (if the client was null). Otherwise, the scrollWin uses metrics.colDelta for the sbLine* actions, and the inner window’s width - metrics.colDelta for the sbPage* actions.

Once the scrollWin has determined the amount to scroll, it sends msgWinDelta to the clientWin.

**msgScrollbarProvideVertInfo**

Client should provide the document and view info.

Takes P_SCROLLBAR_PROVIDE, returns STATUS. Category: client responsibility.

**Comments**

cIsScrollWin responds by filling out the pArgs fields. It sets the viewLength to the height of the inner window. If there is a current clientWin, then the scrollWin sets the docLength to the height of the clientWin, and the offset to the difference between the top of the clientWin and the top of the inner window. If there is no current clientWin, the scrollWin sets both docLength and offset to 0.
**msgScrollbarProvideHorizInfo**

Client should provide the document and view info.

Takes P_SCROLLBAR_PROVIDE, returns STATUS. Category: client responsibility.

**Comments**

clsScrollWin responds by filling out the pArgs fields. It sets the **viewLength** to the width of the inner window. If there is a current **clientWin**, then the scrollWin sets the **docLength** to the width of the **clientWin**, and the offset to the negative of the **clientWin**’s x. If there is no current **clientWin**, the scrollWin sets both **docLength** and offset to 0.

**msgEmbeddedWinShowChild**

Display a given area of an embeddedWin to the user

Takes P_EMBEDDED_WIN_SHOW_CHILD, returns STATUS.

**Comments**

clsScrollWin responds by sending messages to the vertical and/or horizontal scrollbars to scroll the client window area into view.
This file contains the API definition for clsTabBar. clsTabBar inherits from clsTkTable. Implements a window that lays out its children in a single column or row. TabBars are most often seen at the side of Notebooks. clsTabBar will overlap its children in a regular fashion if they won’t fit in the long dimension. clsTabBar also handles flick gestures forwarded to it by rearranging the children.

### Debugging Flags

The clsTabBar debugging flag is 'K'. Defined values are:

```c
flag12 (0x1000)  general debug info
#ifndef TABBAR_INCLUDED
#define TABBAR_INCLUDED
#endif
#include <tktable.h>
#endif
```

### Common #defines and typedefs

```c
typedef OBJECT TAB_BAR, *P_TAB_BAR;
```

### Direction

```c
#define tbDirectionVertical   0   // vertical tab bar
#define tbDirectionHorizontal 1   // horizontal tab bar
typedef struct TAB_BAR_STYLE {
    U16 direction : 1, // vertical or horizontal
    incrementalLayout : 1, // careful about add and remove children
    spare : 14; // unused (reserved)
} TAB_BAR_STYLE, *P_TAB_BAR_STYLE;
```

Default TabBar style:

```c
direction = tbDirectionVertical
incrementalLayout = true
```

### Messages

#### msgNew

Creates a tabBar window.

Takes P_TAB_BAR_NEW, returns STATUS. Category: class message.
typedef struct TAB_BAR_NEW_ONLY {
    TAB_BAR_STYLE     style;    // overall style
    U32               spare;   // unused (reserved)
} TAB_BAR_NEW_ONLY, *P_TAB_BAR_NEW_ONLY;
#define tabBarNewFields
    tkTableNewFields
    TAB_BAR_NEW_ONLY tabBar;
typedef struct TAB_BAR_NEW {
    tabBarNewFields
} TAB_BAR_NEW, *P_TAB_BAR_NEW;

The fields you commonly set are:

pArgs->tabBar.style.direction  whether horizontal or vertical

msgNewDefaults

Initializes the TAB_BAR_NEW structure to default values.

Takes P_TAB_BAR_NEW, returns STATUS. Category: class message.

Message:
typedef struct TAB_BAR_NEW {
    tabBarNewFields
} TAB_BAR_NEW, *P_TAB_BAR_NEW;

Comments:

Zeroes out pArgs->tabBar and sets

pArgs->win.flags.style  |= wsTransparent | wsClipChildren;
pArgs->win.flags.input  |= inputDisable | inputTransparent;
pArgs->gWin.style.gestureEnable = false;
pArgs->border.style.backgroundInk |= bsInkExclusive;
pArgs->border.style.leftMargin = bsMarginNone;
pArgs->border.style.rightMargin = bsMarginNone;
pArgs->border.style.bottomMargin = bsMarginNone;
pArgs->border.style.topMargin = bsMarginNone;
pArgs->tableLayout.style.tblXAlignment = tlAlignCenter;
pArgs->tableLayout.style.tblYAlignment = tlAlignCenter;
pArgs->tableLayout.style.growChildHeight = false;
pArgs->tableLayout.style.reverseY = true;
pArgs->tableLayout.numCols.constraint = tlAbsolute;
pArgs->tableLayout.numCols.value = 1;
pArgs->tableLayout.numRows.constraint = tlInfinite;
pArgs->tableLayout.colWidth.constraint = tlChildrenMax;
pArgs->tableLayout.colWidth.gap = 0;
pArgs->tableLayout.rowHeight.constraint = tlGroupMax;
pArgs->tableLayout.rowHeight.gap = defaultRowGap;

pArgs->tabBar.style.incrementalLayout = true;

Also sets the default child structure in pArgs->tkTable.pButtonNew to be appropriate for labels and buttons that may be rotated 270 degrees and have curved overlapping "tabs".

msgTabBarGetStyle

Passes back the style values.

Takes P_TAB_BAR_STYLE, returns STATUS.

#define msgTabBarGetStyle MakeMsg(clsTabBar, 1)
typedef struct TAB_BAR_STYLE {
    U16 direction : 1;  // vertical or horizontal
    incrementalLayout : 1;  // careful about add and remove children
    spare : 14;  // unused (reserved)
} TAB_BAR_STYLE, *P_TAB_BAR_STYLE;

msgTabBarSetStyle
Sets the style values.
Takes P_TAB_BAR_STYLE, returns STATUS.
#define msgTabBarSetStyle MakeMsg(clsTabBar, 2)

typedef struct TAB_BAR_STYLE {
    U16 direction : 1;  // vertical or horizontal
    incrementalLayout : 1;  // careful about add and remove children
    spare : 14;  // unused (reserved)
} TAB_BAR_STYLE, *P_TAB_BAR_STYLE;

Messages from Other Classes

msgSave
Causes an object to file itself in an object file.
Takes P_OBJ_SAVE, returns STATUS.

Comments
clsTabBar responds by filing away its instance data.

msgRestore
Creates and restores an object from an object file.
Takes P_OBJ_RESTORE, returns STATUS.

Comments
clsTabBar responds by restoring its instance data.

msgWinLayoutSelf
Tells a window to layout its children.
Takes P_WIN_METRICS, returns STATUS.

Comments
When a tabBar receives msgWinLayoutSelf, it will ignore the current positions of its children and do a full relayout, crunching the children toward the bottom (right) of itself, if necessary.

To insert or remove a child and cause the tabBar to incrementally fix up the tab positions (i.e., without doing a full relayout), use msgTkTableAdd* and msgTkTableRemove. When a tabBar receives these messages, it checks its incrementalLayout style bit. If this is on, the tabBar will fix up the area around the inserted/removed child. If the style bit is off, the tabBar will not do relayout.

If you want to add/remove more than a few tabs, turn incrementalLayout off, add/remove the children, then send msgWinLayout to the tabBar.

See Also
msgTkTableAdd* adds a child and will immediately fix up the layout of the tabBar’s children (if style.incrementalLayout is true).

msgTkTableRemove removes a child and will immediately fix up the layout of the tabBar’s children (if style.incrementalLayout is true).
**msgWinSend**

Sends a message up a window ancestry chain.

Takes `P_WIN_SEND`, returns `STATUS`.

Comments

When a `tabBar` receives this message, it is usually because the `tabBar` has an "expand" menu up, and the user has tapped on one of those menu buttons.

If the `pArgs->msg` is not `msgMenuDone`, or the `tabBar` does not have a menu up, the `tabBar` will just return the result of calling its ancestor.

Otherwise, the `tabBar` will take down the menu via `msgMenuShow`, post a `msgDestroy` to it, and then return `stsOK`. This is all the `tabBar` must do at this point, since the principle work of the `menuButton` was done when it sent its message to its client (in this case, the client is the `tabBar`).

**msgGWinForwardedGesture**

Message received when object is forwarded a gesture.

Takes `P_GWIN_GESTURE`, returns `STATUS`.

Comments

TabBars respond to flick gestures by potentially altering the layout of their child windows. This allows a user to rearrange the child buttons when there's not enough room to display all the children fully.

The `tabBar` will first test `pArgs->msg` to see if it is not a flick gesture or it is but it would have no meaning. If either is true, the `tabBar` will return `stsMessageIgnored`.

If all the children are fully displayed, the `tabBar` will return `stsOK`.

If `style.direction` is `tbDirectionVertical` and `pArgs->msg` is `xgsFlickLeft`, or the direction is `tbDirectionHorizontal` and `pArgs->msg` is `xgsFlickUp`, the `tabBar` will create and put up a menu over itself that looks like an expanded `tabBar`. The user then tap on one of the menu buttons; this will have the same effect as tapping on the corresponding `tabBar` child. After putting up the menu, the `tabBar` will return `stsOK`.

If all of the above checks failed, the `tabBar` will process the flick gesture by moving its children as appropriate and then returning `stsOK`.

**msgTkTableChild Defaults**

Sets the defaults in `P_ARGS` for a common child.

Takes `P_UNKNOWN`, returns `STATUS`.

Comments

Here is how a `tabBar` processes this message if `style.direction` is `tbDirectionVertical`:

```
pArgs->win.flags.style &= -wsParentClip;
pArgs->win.flags.style |= wsClipSiblings | wsClipChildren;
if <pArgs->object.class inherits from clsBorder> {
    pArgs->border.style.edge = bsEdgeTop | bsEdgeRight | bsEdgeBottom;
pArgs->border.style.join = bsJoinRound;
pArgs->border.style.backgroundInk = bsInkWhite;
pArgs->border.style.topMargin = bsMarginMedium;
pArgs->border.style.bottomMargin = bsMarginMedium;
pArgs->border.style.shadow = bsShadowThinBlack;
}
if <pArgs->object.class inherits from clsLabel> {
    pArgs->label.style.xAlignment = lsAlignCenter;
pArgs->label.style.yAlignment = lsAlignCenter;
pArgs->label.style.rotation = lsRotate270;
pArgs->label.scale = lsScaleMedium;
```
Here is how a tabBar processes this message if style.direction is tbDirectionHorizontal:

```c
pArgs->win.flags.style &= ~wsParentClip;
pArgs->win.flags.style |= wsClipSiblings | wsClipChildren;
if <pArgs->object.class inherits from clsBorder> {
    pArgs->border.style.edge = bsEdgeLeft | bsEdgeRight | bsEdgeBottom;
    pArgs->border.style.join = bsJoinRound;
    pArgs->border.style.backgroundColor = bsInkWhite;
    pArgs->border.style.leftMargin = bsMarginMedium;
    pArgs->border.style.rightMargin = bsMarginMedium;
    pArgs->border.style.topMargin = bsMarginSmall;
    pArgs->border.style.bottomMargin = bsMarginSmall;
    pArgs->border.style.shadow = bsShadowThinBlack;
    pArgs->border.style.shadowGap = bsGapNone;
}
if <pArgs->object.class inherits from clsLabel> {
    pArgs->label.style.textAlignment = lsAlignCenter;
    pArgs->label.style.yAlignment = lsAlignCenter;
    pArgs->label.style.rotation = lsRotateNone;
    pArgs->label.scale = lsScaleMedium;
}
```

**msgTkTableAddAsFirst**

Adds specified window as the first child in the table.

Takes WIN, returns STATUS.

Comments

clsTabBar responds by first calling its ancestor, then checking style.incrementalLayout. If this is false, the tabBar will just return stsOK.

Otherwise, the tabBar will do whatever layout is necessary to fix up the positions of its children.

**msgTkTableAddAsLast**

Adds specified window as the last child in the table.

Takes WIN, returns STATUS.

Comments

clsTabBar responds by first calling its ancestor, then checking style.incrementalLayout. If this is false, the tabBar will just return stsOK.

Otherwise, the tabBar will do whatever layout is necessary to fix up the positions of its children.

**msgTkTableAddAsSibling**

Inserts specified window in front of or behind an existing child.

Takes P_TK_TABLE_ADD_SIBLING, returns STATUS.

Comments

clsTabBar responds by first calling its ancestor, then checking style.incrementalLayout. If this is false, the tabBar will just return stsOK.

Otherwise, the tabBar will do whatever layout is necessary to fix up the positions of its children.
msgTkTableAddAt
Inserts specified window table at specified index.
Takes P_TK_TABLE_ADD_AT, returns STATUS.

Comments
clsTabBar responds by first calling its ancestor, then checking style.incrementalLayout. If this is false, the tabBar will just return stsOK.
Otherwise, the tabBar will do whatever layout is necessary to fix up the positions of its children.

msgTkTableRemove
Extracts specified window.
Takes WIN, returns STATUS.

Comments
Currently, the tabBar just calls its ancestor and does not attempt to fix up the layout of its children. This may change in the future.
This file contains the API definition for clsTitleBar.

clsTitleBar inherits from clsButton.

Title bars are the standard frame decorations which support dragging a frame, bringing a frame to the front, and flicking to zoom.

ifndef TBAR_INCLUDED
#define TBAR_INCLUDED

#include <button.h>

ifndef BUTTON_INCLUDED
#define BUTTON_INCLUDED
#endif

Common #defines and typedefs

typedef OBJECT TITLE_BAR;
typedef struct TITLE_BAR_STYLE {
    U16 spare    : 16; // unused (reserved)
} TITLE_BAR_STYLE, *P_TITLE_BAR_STYLE;

Messages

msgNew

Creates a title bar window.

Takes P_TITLE_BAR_NEW, returns STATUS. Category: class message.

Arguments
typedef struct TITLE_BAR_NEW_ONLY {
    TITLE_BAR_STYLE style;
    U32    spare1; // unused (reserved)
    U32    spare2; // unused (reserved)
} TITLE_BAR_NEW_ONLY, *P_TITLE_BAR_NEW_ONLY;

#define titleBarNewFields \
    buttonNewFields \
    TITLE_BAR_NEW_ONLY titleBar;

typedef struct TITLE_BAR_NEW {
    titleBarNewFields
} TITLE_BAR_NEW, *P_TITLE_BAR_NEW;

msgNewDefaults

Initializes the TITLE_BAR_NEW structure to default values.

Takes P_TITLE_BAR_NEW, returns STATUS. Category: class message.

Arguments
typedef struct TITLE_BAR_NEW {
    titleBarNewFields
} TITLE_BAR_NEW, *P_TITLE_BAR_NEW;
Zeroes out `pArgs->titleBar` and sets:

- `pArgs->border.style.join` = `bsJoinSquare`;
- `pArgs->border.style.shadow` = `bsShadowNone`;
- `pArgs->border.style.leftMargin` = `bsMarginMedium`;
- `pArgs->border.style.rightMargin` = `bsMarginMedium`;
- `pArgs->border.style.bottomMargin` = `bsMarginSmall + bsMarginSmall`;
- `pArgs->border.style.topMargin` = `bsMarginMedium + bsMarginSmall`;
- `pArgs->border.style.drag` = `bsDragHoldDown`;
- `pArgs->border.style.top` = `bsTopUp`;
- `pArgs->border.style.getDeltaWin` = `true`;
- `pArgs->control.style.previewEnable` = `false`;
- `pArgs->label.style.xAlignment` = `lsAlignCustom`;
- `pArgs->button.style.feedback` = `bsFeedbackNone`;

---

**msgTitleBarGetStyle**

Passes back the current style values.

Takes `P_TITLE_BAR_STYLE`, returns `STATUS`.

```c
#define msgTitleBarGetStyle MakeMsg(clsTitleBar, 1)
```

**Message Arguments**

```c
typedef struct TITLE_BAR_STYLE {
    UI16 spare : 16; // unused (reserved)
} TITLE_BAR_STYLE, *P_TITLE_BAR_STYLE;
```

---

**msgTitleBarSetStyle**

Sets the style values.

Takes `P_TITLE_BAR_STYLE`, returns `STATUS`.

```c
#define msgTitleBarSetStyle MakeMsg(clsTitleBar, 2)
```

**Message Arguments**

```c
typedef struct TITLE_BAR_STYLE {
    UI16 spare : 16; // unused (reserved)
} TITLE_BAR_STYLE, *P_TITLE_BAR_STYLE;
```
This file contains the API definition for clsTabButton.

clsTabButton inherits from clsButton.

Provides a class of button useful in the popup choice contained in the title of option sheets, because tab buttons hold some flags, a window uid, and an extra client.

```c
#ifndef TBUTTON_INCLUDED
#define TBUTTON_INCLUDED

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

#ifndef BUTTON_INCLUDED
#include <button.h>
#endif

typedef struct TAB_BUTTON_METRICS {
  U16 flags;  // arbitrary flags
  WIN win;    // associated window uid
  OBJECT client; // associated client
  U32 clientData[2]; // arbitrary client data
  U32 spare; // reserved
} TAB_BUTTON_METRICS, *P_TAB_BUTTON_METRICS;

msgNew
Creates a tab button.

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

Arguments
```c
typedef struct TAB_BUTTON_NEW_ONLY {
  TAB_BUTTON_METRICS metrics;
  U32 spare; // reserved
} TAB_BUTTON_NEW_ONLY, *P_TAB_BUTTON_NEW_ONLY;
```

```c
#define tabButtonNewFields
buttonNewFields \
TAB_BUTTON_NEW_ONLY tabButton;
```

```c
typedef struct TAB_BUTTON_NEW {
  tabButtonNewFields
} TAB_BUTTON_NEW, *P_TAB_BUTTON_NEW;
```

Comments
The fields you commonly set are:

- `pArgs->tabButton.metrics.win` a window uid to hold
- `pArgs->tabButton.metrics.client` a client uid to hold
msgNewDefaults
Initializes the TAB_BUTTON_NEW structure to default values.
Takes P_TAB_BUTTON_NEW, returns STATUS. Category: class message.

Message Arguments
typedef struct TAB_BUTTON_NEW {
    tabButtonNewFields
} TAB_BUTTON_NEW, *P_TAB_BUTTON_NEW;
Comments
Zeroes out pArgs->tabButton.

msgTabButtonGetMetrics
Passes back the metrics of a tab button.
Takes P_TAB_BUTTON_METRICS, returns STATUS.
#define msgTabButtonGetMetrics MakeMsg(clsTabButton, 1)
typedef struct TAB_BUTTON_METRICS {
    U16 flags;  // arbitrary flags
    WIN win;    // associated window uid
    OBJECT client; // associated client
    U32 clientData[2]; // arbitrary client data
    U32 spare; // reserved
} TAB_BUTTON_METRICS, *P_TAB_BUTTON_METRICS;

msgTabButtonSetMetrics
Sets the metrics of a tab button.
Takes P_TAB_BUTTON_METRICS, returns STATUS.
#define msgTabButtonSetMetrics MakeMsg(clsTabButton, 2)
typedef struct TAB_BUTTON_METRICS {
    U16 flags;  // arbitrary flags
    WIN win;    // associated window uid
    OBJECT client; // associated client
    U32 clientData[2]; // arbitrary client data
    U32 spare; // reserved
} TAB_BUTTON_METRICS, *P_TAB_BUTTON_METRICS;

msgTabButtonGetFlags
Passes back the flags of a tab button.
Takes P_U16, returns STATUS.
#define msgTabButtonGetFlags MakeMsg(clsTabButton, 3)

msgTabButtonSetFlags
Sets the flags of a tab button.
Takes U16, returns STATUS.
#define msgTabButtonSetFlags MakeMsg(clsTabButton, 4)
Messages from Other Classes

msgSave
Causes an object to file itself in an object file.
Takes P_OBJ_SAVE, returns STATUS.

Comments
clsTabButton will save its instance data.
If the TAB_BUTTON_METRICS.win is not null and the window's wsSendFile flag is on, the window will be filed with msgResPutObject (the window's wsFileInline flag is cleared first).
If the TAB_BUTTON_METRICS.client is OSThisApp(), this fact is saved so that clsTabButton's response to msgRestore will restore the client to OSThisApp() again. If the client is not OSThisApp(), msgRestore will set the client to null.

msgRestore
Creates and restores an object from an object file.
Takes P_OBJ_RESTORE, returns STATUS.

Comments
clsTabButton restores its instance data.
If the TAB_BUTTON_METRICS.client had been OSThisApp() at msgSave time, msgRestore will set the client to OSThisApp() again.
This file contains the API definitions for clsDateField, clsFixedField, clsIntegerField, and clsTextField.

clsDateField inherits from clsField.
Provides a field that treats its label string as a date.

clsFixedField inherits from clsField.
Provides a field that treats its label string as a number in hundredths.

clsIntegerField inherits from clsField.
Provides a field that treats its label string as an integer.

clsTextField inherits from clsField.
Provides a field that treats its label string as a string.

These four classes are used mainly on option sheets. Because these subclasses provide a simple API and somewhat limited functionality, clients should consider subclassing clsField rather than these.

```c
#ifndef TKFIELD_INCLUDED
#define TKFIELD_INCLUDED

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

#ifndef FIELD_INCLUDED
#include <field.h>
#endif

#include <time.h>
```

## clsDateField

This section describes the API for clsDateField.

### Debugging Flags

The clsDateField debugging flag is 'K'. Defined values are:

flag0 (0x0001) general

### Common #defines and typedefs

```c
#define stsDateFieldEmpty MakeStatus(clsDateField, 1)
#define stsDateFieldInvalid MakeStatus(clsDateField, 2)

// Date Flags
#define dfsMonthName flag0
#define dfsFullName flag1

typedef struct {
    U16 flags;
    U16 spare;
} DATE_FIELD_STYLE, *P_DATE_FIELD_STYLE;
```
Default DATE_FIELD_STYLE:

    flags = 0

typedef struct tm TIME_DESC, *P_TIME_DESC;

msgNew

Creates a date field.

Takes P_DATE_FIELD_NEW, returns STATUS. Category: class message.

Arguments
typedef struct {
    DATE_FIELD_STYLE style;
    U32 spare;
} DATE_FIELD_NEW_ONLY, *P_DATE_FIELD_NEW_ONLY;
#define dateFieldNewFields
    fieldNewFields
    DATE_FIELD_NEW_ONLY dateField;

typedef struct DATE_FIELD_NEW {
    dateFieldNewFields
} DATE_FIELD_NEW, *P_DATE_FIELD_NEW;

Comments
The fields you commonly set are:

pArgs->dateField.style.flags appropriate flags

msgNewDefaults

Initializes the DATE_FIELD_NEW structure to default values.

Takes P_DATE_FIELD_NEW, returns STATUS. Category: class message.

Arguments
typedef struct DATE_FIELD_NEW {
    dateFieldNewFields
} DATE_FIELD_NEW, *P_DATE_FIELD_NEW;

Comments
Zeroes out pArgs->dateField and sets:

    pArgs->border.style.edge = bsEdgeNone;
    pArgs->border.style.borderInk = bsInkGray66;
    pArgs->field.style.editType = fstOverWrite;

msgDateFieldGetStyle

Passes back the receiver's style.

Takes P_DATE_FIELD_STYLE, returns STATUS.

#define msgDateFieldGetStyle MakeMsg(clsDateField, 1)

Arguments
typedef struct {
    U16 flags;
    U16 spare;
} DATE_FIELD_STYLE, *P_DATE_FIELD_STYLE;

msgDateFieldSetStyle

Sets the receiver's style.

Takes P_DATE_FIELD_STYLE, returns STATUS.

#define msgDateFieldSetStyle MakeMsg(clsDateField, 2)

Arguments
typedef struct {
    U16 flags;
    U16 spare;
} DATE_FIELD_STYLE, *P_DATE_FIELD_STYLE;
msgDateFieldGetValue
Passes back the receiver's value in the time descriptor.
Takes P_TIME_DESC, returns STATUS.

#define msgDateFieldGetValue MakeMsg(clsDateField, 3)

Return Value
stsDateFieldEmpty field has no content (*pArgs not set).
stsDateFieldInvalid field's content unrecognized (*pArgs not set).

msgDateFieldSetValue
Sets the receiver's label string from the time descriptor.
Takes P_TIME_DESC, returns STATUS.

#define msgDateFieldSetValue MakeMsg(clsDateField, 4)

msgControlGetValue
Passes back the receiver's value in YYYYMMDD format.
Takes P_U32, returns STATUS.

Return Value
stsDateFieldEmpty field has no content (*pArgs not set).
stsDateFieldInvalid field's content unrecognized (*pArgs not set).

msgControlSetValue
Sets the receiver's label string from a U32 in YYYYMMDD format.
Takes U32, returns STATUS.

msgControlSetDirty
Sets style.dirty.
Takes BOOLEAN, returns STATUS.

Comments
The date field will alter the ink of its bottom edge (if it has one) to bsInkBlack if dirty, bsInkGray66 if not.

In PenPoint 1.0, clsDateField does not respond to msgControlSetStyle or msgControlSetMetrics to watch for the CONTROL_STYLE.enable bit changing.

clsFixedField
This section describes the API for clsFixedField.

Common #defines and typedefs

#define stsFixedFieldEmpty MakeStatus(clsFixedField, 1)
#define stsFixedFieldInvalid MakeStatus(clsFixedField, 2)
typedef struct {
  U16  flags;
  U16  spare;
} FIXED_FIELD_STYLE, *P_FIXED_FIELD_STYLE;
**msgNew**

Creates a fixed field.

Takes P_FIXED_FIELD_NEW, returns STATUS. Category: class message.

**Arguments**

typedef struct {
    FIXED_FIELD_STYLE style;
    U32 spare;
} FIXED_FIELD_NEW_ONLY, *P_FIXED_FIELD_NEW_ONLY;
#define fixedFieldNewFields \
    fieldNewFields \
    FIXED_FIELD_NEW ONLY fixedField;
typedef struct FIXED_FIELD_NEW {
    fixedFieldNewFields
} FIXED_FIELD_NEW, *P_FIXED_FIELD_NEW;

**msgNewDefaults**

Initializes the FIXED_FIELD_NEW structure to default values.

Takes P_FIXED_FIELD_NEW, returns STATUS. Category: class message.

**Message**

typedef struct FIXED_FIELD_NEW {
    fixedFieldNewFields
} FIXED_FIELD_NEW, *P_FIXED_FIELD_NEW;

**Arguments**

Zeroes out pArgs->fixedField and sets:
- pArgs->border.style.edge = bsEdgeNone;
- pArgs->border.style.borderInk = bsInkGray66;
- pArgs->field.style.editType = fstOverWrite;
- pArgs->field.style.noSpace = true;
- pArgs->field.style.veto = true;

**msgFixedFieldGetStyle**

Passes back the receiver’s style.

Takes P_FIXED_FIELD_STYLE, returns STATUS.

```
#define msgFixedFieldGetStyle MakeMsg(clsFixedField, 1)
```

**Message**

typedef struct {
    U16 flags;
    U16 spare;
} FIXED_FIELD_STYLE, *P_FIXED_FIELD_STYLE;

**Arguments**

**msgFixedFieldSetStyle**

Sets the receiver’s style.

Takes P_FIXED_FIELD_STYLE, returns STATUS.

```
#define msgFixedFieldSetStyle MakeMsg(clsFixedField, 2)
```

**Message**

typedef struct {
    U16 flags;
    U16 spare;
} FIXED_FIELD_STYLE, *P_FIXED_FIELD_STYLE;
msgControlGetValue
Get the receiver’s value as an S32 in hundredths.
Takes P_S32, returns STATUS.

Return Value
stsFixedFieldEmpty  field has no content (*pArgs not set).
stsFixedFieldInvalid field’s content unrecognized (*pArgs not set).

msgControlSetValue
Sets the receiver’s label string from a S32 in hundredths.
Takes S32, returns STATUS.

msgControlSetDirty
Sets style.dirty.
Takes BOOLEAN, returns STATUS.

Comments
The fixed field will alter the ink of its bottom edge (if it has one) to bsInkBlack if dirty, bsInkGray66 if not.
In PenPoint 1.0, clsFixedField does not respond to msgControlSetStyle or msgControlSetMetrics to watch for the CONTROL_STYLE.enable bit changing.

\section*{clsIntegerField}
This section describes the API for clsIntegerField.

\section*{Common #defines and typedefs}
\begin{verbatim}
#define stsIntegerFieldEmpty MakeStatus(clsIntegerField, 1)
#define stsIntegerFieldInvalid MakeStatus(clsIntegerField, 2)
typedef struct {
   U16 flags;
   U16 spare;
} INTEGER_FIELD_STYLE, *P_INTEGER_FIELD_STYLE;
\end{verbatim}

msgNew
Creates an integer field.
Takes P_INTEGER_FIELD_NEW, returns STATUS. Category: class message.

Arguments
\begin{verbatim}
typedef struct {
   INTEGER_FIELD_STYLE style;
   U32 spare;
} INTEGER_FIELD_NEW_ONLY, *P_INTEGER_FIELD_NEW_ONLY;
define integerFieldNewFields \ fieldNewFields \ INTEGER_FIELD_NEW_ONLY integerField;
typedef struct INTEGER_FIELD_NEW {
   integerFieldNewFields
} INTEGER_FIELD_NEW, *P_INTEGER_FIELD_NEW;
\end{verbatim}
msgNewDefaults

Initializes the INTEGER_FIELD_NEW structure to default values.

Takes P_INTEGER_FIELD_NEW, returns STATUS. Category: class message.

typedef struct INTEGER_FIELD_NEW {
    integerFieldNewFields
} INTEGER_FIELD_NEW, *P_INTEGER_FIELD_NEW;

Zeroes out pArgs->integerField and sets:

    pArgs->border.style.edge = bsEdgeNone;
    pArgs->border.style.borderInk = bsInkGray66;
    pArgs->field.style.editType = fstOverWrite;
    pArgs->field.style.noSpace = true;
    pArgs->field.style.veto = true;

msgIntegerFieldGetStyle

Passes back the receiver's style.

Takes P_INTEGER_FIELD_STYLE, returns STATUS.

#define rnsgIntegerFieldGetStyle MakeMsg(clsIntegerField, 1)

typedef struct {
    U16 flags;
    U16 spare;
} INTEGER_FIELD_STYLE, *P_INTEGER_FIELD_STYLE;

msgIntegerFieldSetStyle

Sets the receiver's style.

Takes P_INTEGER_FIELD_STYLE, returns STATUS.

#define rnsgIntegerFieldSetStyle MakeMsg(clsIntegerField, 2)

typedef struct {
    U16 flags;
    U16 spare;
} INTEGER_FIELD_STYLE, *P_INTEGER_FIELD_STYLE;

msgControlGetValue

Passes back the receiver's value as an S32.

Takes P_S32, returns STATUS.

Return Value

  stsIntegerFieldEmpty   field has no content (*pArgs not set).
  stsIntegerFieldInvalid field's content unrecognized (*pArgs not set).

msgControlSetValue

Sets the receiver's label string from a S32.

Takes S32, returns STATUS.
**msgControlSetDirty**

Sets style.dirty.

Takes BOOLEAN, returns STATUS.

*Comments*

The integer field will alter the ink of its bottom edge (if it has one) to bsInkBlack if dirty, bsInkGray66 if not.

In PenPoint 1.0, clsIntegerField does not respond to msgControlSetStyle or msgControlSetMetrics to watch for the CONTROL_STYLE.enable bit changing.

**clsTextField**

This section describes the API for clsTextField.

**Common #defines and typedefs**

```c
typedef struct {
    U16 flags;
    U16 spare;
} TEXT_FIELD_STYLE, *P_TEXT_FIELD_STYLE;
```

**msgNew**

Creates a text field.

Takes P_TEXT_FIELD_NEW, returns STATUS. Category: class message.

**Arguments**

```c
typedef struct {
    TEXT_FIELD_STYLE style;
    U32 spare;
} TEXT_FIELD_NEW_ONLY, *P_TEXT_FIELD_NEW_ONLY;
#define textFieldNewFields
    fieldNewFields
    TEXT_FIELD_NEW_ONLY textField;
typedef struct TEXT_FIELD_NEW {
    textFieldNewFields
} TEXT_FIELD_NEW, *P_TEXT_FIELD_NEW;
```

**msgNewDefaults**

Initializes the TEXT_FIELD_NEW structure to default values.

Takes P_TEXT_FIELD_NEW, returns STATUS. Category: class message.

**Message**

```c
typedef struct TEXT_FIELD_NEW {
    textFieldNewFields
} TEXT_FIELD_NEW, *P_TEXT_FIELD_NEW;
```

**Comments**

Zeroes out pArgs->textField and sets:

```c
pArgs->border.style.edge = bsEdgeBottom;
pArgs->border.style.borderInk = bsInkGray66;
```

**msgTextFieldGetStyle**

Passes back the receiver's style.

Takes P_TEXT_FIELD_STYLE, returns STATUS.

```c
#define msgTextFieldGetStyle MakeMsg(clsTextField, 1)
```
typedef struct {
    U16 flags;
    U16 spare;
} TEXT_FIELD_STYLE, *P_TEXT_FIELD_STYLE;

#define msgTextFieldSetStyle MakeMsg(clsTextField, 2)

msgTextFieldSetStyle
Sets the receiver's style.
Takes P_TEXT_FIELD_STYLE, returns STATUS.

msgControlSetDirty
Sets style.dirty.
Takes BOOLEAN, returns STATUS.

Comments
The text field will alter the ink of its bottom edge (if it has one) to bsInkBlack if dirty, bsInkGray66 if not.

In PenPoint 1.0, clsTextField does not respond to msgControlSetStyle or msgControlSetMetrics to watch for the CONTROL_STYLE.enable bit changing.
TKTABLE.H

This file contains the API definition for clsTkTable. clsTkTable inherits from clsTableLayout. Toolkit tables support complex nested arrangements of buttons, labels, and even other toolkit tables.

(Debugging Flags)

The clsTkTable debugging flag is 'K'. Defined values are:

flag12 (0x1000) general debug info

ifndef TKTABLE_INCLUDED
#define TKTABLE_INCLUDED

#include <ostypes.h>

ifndef TTABLE_INCLUDED
#define TTABLE_INCLUDED

#include <tlayout.h>

ifndef BUTTON_INCLUDED
#define BUTTON_INCLUDED

#include <button.h>

endif

 ifndef OS TYPES_INCLUDED
# endif
 ifndef TLAYOUT_INCLUDED
# endif
 ifndef BUTTON_INCLUDED
# endif

 Common #defines and typedefs

typedef OBJECT TK_TABLE;
typedef struct TK_TABLE_STYLE {
   U16 spare : 16; // unused (reserved)
} TK_TABLE_STYLE, *P_TK_TABLE_STYLE;

(TK_TABLE_ENTRY Flags)

#define tkLabelEntry ((U32)flag2) // arg1 is a P_TK_TABLE_ENTRY
#define tkLabelStringId ((U32)flag14) // arg1 is a string resid
#define tkPNew ((U32)flag4) // arg1 is a pNew
#define tkLabelBold ((U32)flag3) // use a bold system font
#define tkLabelWordWrap ((U32)flag25) // word-wrap the label string
#define tkButtonPargsValue ((U32)flag5) // send value instead of Data
#define tkButtonPargsUID ((U32)flag6) // send UID instead of Data
#define tkButtonOn ((U32)flag7) // turn on the button
#define tkButtonHalfHeight ((U32)flag19) // use half-height button border
#define tkButtonManagerNone ((U32)flag20) // set button manager to bsManagerNone
#define tkButtonToggle ((U32)flag8) // make button a toggle
#define tkButtonBox ((U32)flag1) // use bsFeedbackBox
#define tkMenuPullRight ((U32)flag9) // arg2 is pEntries for pull-right
#define tkMenuPullDown ((U32)flag10) // arg2 is pEntries for pull-down
#define tkContentsSection ((U32)flag9) // arg2 is pEntries for section contents
#define tkInputDisable ((U32)flag21) // disable input
#define tkBorderEdgeTop ((U32)flag11) // turn on top border
#define tkBorderEdgeBottom ((U32)flag12) // turn on bottom border
#define tkBorderMarginNone ((U32)flag22) // turn off all margins
#define tkBorderLookInactive ((U32)flag13) // make entry inactive
#define tkTableWideGap ((U32)flag15) // wide gap between col 1 & 2
#define tkTableHorizontal ((U32)flag17) // table is horizontal
#define tkTableVertical ((U32)flag24) // table is vertical
#define tkTableXAlignBaseline ((U32)flag9) // childXAlignment = tlAlignBaseline
#define tkTableYAlignBaseline ((U32)flag27) // childYAlignment = tlAlignBaseline
#define tkNoProto ((U32)flag18) // don't use prototypical pButtonNew
#define tkNoClient ((U32)flag23) // don't copy client field
#define tkPopupChoiceFont ((U32)flag26) // use current font names
#define tkControlDynamicClient ((U32)flag27) // dynamicEnable = csDynamicClient
#define tkControlDynamicObject ((U32)flag27) // dynamicEnable = csDynamicObject
#define tkControlDynamicPargs ((U32)flag28) // dynamicEnable = csDynamicPargs
#define tkControlCallSel tkControlDynamicObject
#define tkControlSelLocal tkControlDynamicPargs
#define tkMenuButtonGetMenu ((U32)flag29) // send msgMenuButtonProvideMenu
#define tkMenuButtonEnableMenu ((U32)flag30) // send msgControlEnable

typedef struct TK_TABLE_ENTRY {
    P UNKNOWN arg1; // argument for class, e.g. pString
    U32 arg2; // argument for class, e.g. msg
    U32 arg3; // argument for class, e.g. data
    U32 tag; // window tag
    U32 flags; // e.g. tkLabelBold | tkButtonPargs
    CLASS childClass; // class to create or objNull for default
    U32 helpId; // help id for clsGWIn
    U32 spare; // unused (reserved)
} TK_TABLE_ENTRY, *pTK_TABLE_ENTRY;

Interpretation of arg1, arg2, and arg3 for different classes:

  clsLabel   pString
  clsButton  pString, msg, data
  clsMenuButton pString, pEntries if (tkMenuPullRight || tkMenuPullDown)
  clsMenuButton pString, msg, data if (!tkMenuPullRight || tkMenuPullDown))
  clsContentsButton pString, pEntries if (!tkContentsSection)
  clsContentsButton pString, msg, data if (!tkContentsSection)
  clsTkTable pEntries, numRows/cols
  clsChoice  pEntries, numRows/cols
  clsToggleTable pEntries, numRows/cols
  clsPopupChoice pEntries, numRows/cols // if (!tkPopupChoiceFont)
  clsPopupChoice pEntries, numRows/cols if (!tkPopupChoiceFont)
  clsField   pString, numCols, maxLen
  clsListBox nEntries, nEntriesToView
  clsFontListBox role, nEntriesToView, look

**Messages**

**msgNew**

Creates a tk table window.

Takes P _TK_TABLE_NEW_, returns STATUS. Category: class message.

```c
typedef struct TK_TABLE_NEW_ONLY {
    TK_TABLE_STYLE style; // overall style
    OBJECT client; // client for each button
    P_TK_TABLE_ENTRY pEntries; // in/out: description for each child
    U32 spare4; // unused (reserved)
    P_BUTTON_NEW pButtonNew; // default new struct
    U16 spare3; // unused (reserved)
    BUTTON_NEW buf; // default storage
    OBJECT manager; // manager to notify
    U32 spare1; // unused (reserved)
    U32 spare2; // unused (reserved)
} TK_TABLE_NEW_ONLY, *P_TK_TABLE_NEW_ONLY;
```
clsTkTable will create and insert a child window for each entry in pArgs->tkTable.pEntries. After msgNew returns, pArgs->tkTable.pEntries will be left pointing to the null-terminating entry. Note that pArgs->tkTable.pEntries is used during msgNew only, and the original value can be freed (if allocated) after msgNew returns.

For each entry, pArgs->pButtonNew will be used as the "prototypical" child new struct. The fields argl, arg2, arg3, tag, helpId and the semantics of each flag will be applied to the child new struct before creating the child.

pArgs->client will be used to set the client for entries which inherit from clsTkTable, clsListBox, or clsControl, unless the tkNoClient flag is on for the entry.

Before msgNew is sent to each child’s class, msgTkTableInit will be sent to the child’s class with the following TK_TABLE_INIT parameters:

```c
pTkTableNew = pArgs;
pChildNew = pointer to child’s new struct;
pEntry = pointer to child’s TK_TABLE_ENTRY struct;
```

This allows other classes to define mappings for TK_TABLE_ENTRY to child new structs.

### msgNewDefaults

Initializes the TK_TABLE_NEW structure to default values.

Takes P_TK_TABLE_NEW, returns STATUS. Category: class message.

```c
typedef struct TK_TABLE_NEW {
   tkTableNewFields
} TK_TABLE_NEW, *P_TK_TABLE_NEW;
```

Zeroes out pArgs->tkTable and sets

```c
pArgs->tableLayout.style.growChildWidth = false;
pArgs->tableLayout.style.growChildHeight = true;
```

```c
pArgs->tableLayout.numCols.constraint = tlInfinite;
pArgs->tableLayout.numRows.constraint = tlAbsolute;
pArgs->tableLayout.numRows.value = 1;
```

```c
pArgs->tableLayout.colWidth.constraint = tlGroupMax;
pArgs->tableLayout.colWidth.gap = defaultColGap;
pArgs->tableLayout.rowHeight.constraint = tlChildrenMax;
pArgs->tableLayout.rowHeight.gap = defaultRowGap;
```

// default is a table of regular buttons
pArgs->tkTable.pButtonNew = &pArgs->tkTable.buf;

Sends msgNewDefaults(pArgs->tkTable.pButtonNew) to clsButton, then alters pArgs->tkTable.pButtonNew as described in msgTkTableChildDefaults.
### msgTkTableGetStyle

Passes back the current style values.

Takes P_TK_TABLE_STYLE, returns STATUS.

```c
#define msgTkTableGetStyle MakeMsg(clsTkTable, 1)
```

```c
typedef struct TK_TABLE_STYLE {
    U16 spare : 16; // unused (reserved)
} TK_TABLE_STYLE, *P_TK_TABLE_STYLE;
```

### msgTkTableSetStyle

Sets the style values.

Takes P_TK_TABLE_STYLE, returns STATUS.

```c
#define msgTkTableSetStyle MakeMsg(clsTkTable, 2)
```

```c
typedef struct TK_TABLE_STYLE {
    U16 spare : 16; // unused (reserved)
} TK_TABLE_STYLE, *P_TK_TABLE_STYLE;
```

### msgTkTableGetClient

Passes back the client of the first child in the table. Note that the children may have been created with different clients.

Takes P_UID, returns STATUS.

```c
#define msgTkTableGetClient MakeMsg(clsTkTable, 3)
```

clsTkTable sends msgControlGetClient(pArgs) to the first (bottom-most) child to retrieve the client.

### msgTkTableSetClient

Sets the client of each child in the table to pArgs.

Takes UID, returns STATUS.

```c
#define msgTkTableSetClient MakeMsg(clsTkTable, 4)
```

clsTkTable sends msgControlSetClient(pArgs) to each child.

### msgTkTableGetManager

Passes back the manager.

Takes P_UID, returns STATUS.

```c
#define msgTkTableGetManager MakeMsg(clsTkTable, 7)
```

### msgTkTableSetManager

Sets the manager.

Takes UID, returns STATUS.

```c
#define msgTkTableSetManager MakeMsg(clsTkTable, 8)
```
msgTkTableGetMetrics
Passes back the metrics.
Takes P_TK_TABLE_METRICS, returns STATUS.

```c
#define msgTkTableGetMetrics MakeMsg(clsTkTable, 5)
```

**typedef struct TK_TABLE_METRICS {**

```c
  TK_TABLE_STYLE style;  // overall style
  OBJECT manager;       // manager to notify
  U32 spare1;           // unused (reserved)
  U32 spare2;           // unused (reserved)
} TK_TABLE_METRICS, *P_TK_TABLE_METRICS;
```

msgTkTableSetMetrics
Sets the metrics.
Takes P_TK_TABLE_METRICS, returns STATUS.

```c
#define msgTkTableSetMetrics MakeMsg(clsTkTable, 6)
```

**typedef struct TK_TABLE_METRICS {**

```c
  TK_TABLE_STYLE style;  // overall style
  OBJECT manager;       // manager to notify
  U32 spare1;           // unused (reserved)
  U32 spare2;           // unused (reserved)
} TK_TABLE_METRICS, *P_TK_TABLE_METRICS;
```

msgTkTableChildDefaults
Sets the defaults in pArgs for a common child.
Takes P_UNKNOWN, returns STATUS.

```c
#define msgTkTableChildDefaults MakeMsg(clsTkTable, 14)
```

**pArgs** should be an initialized (msgNewDefaults) P_NEW struct.

Clients should use this on children manually inserted into the table. For example, send msgNewDefaults to class of child, then send msgTkTableChildDefaults to the table, then send msgNew to class of child, then add child to table with, e.g., msgTkTableAddAsLast.

clsTkTable responds to msgTkTableChildDefaults as follows:

- sets pArgs->win.device to self's device
- turns on shared parent/child/sibling clipping:
  ```c
  pArgs->win.flags.style |= wSParentClip;
  pArgs->win.flags.style &= ~(wsClipSiblings | wsClipChildren);
  ```
- if pArgs->object.class inherits from clsBorder, sets pArgs->border.style.backgroundlnk to bslnk Transparent
- if pArgs->object.class inherits from clsButton, sets pArgs->button.style.manager to bsManagerParent

msgTkTableAddAsFirst
Inserts pArgs as the first child in the table.
Takes WIN, returns STATUS.

```c
#define msgTkTableAddAsFirst MakeMsg(clsTkTable, 9)
```
**msgTkTableAddAsLast**

Inserts `pArgs` as the last child in the table.

Takes `WIN`, returns `STATUS`.

```c
#define msgTkTableAddAsLast MakeMsg(clsTkTable, 10)
```

**msgTkTableAddAsSibling**

Inserts `pArgs->newChild` in front of or behind `pArgs->sibling`.

Takes `P_TK_TABLE_ADD_SIBLING`, returns `STATUS`.

```c
#define msgTkTableAddAsSibling MakeMsg(clsTkTable, 11)
```

**Arguments**

```c
typedef struct TK_TABLE_ADD_SIBLING {
  WIN newChild;       // new child to add
  WIN sibling;        // existing child already in tkTable
  BOOLEAN before;     // true: add before sibling; false: after
  U32 spare;          // unused (reserved)
} TK_TABLE_ADD_SIBLING, *P_TK_TABLE_ADD_SIBLING;
```

**msgTkTableAddAt**

Inserts `pArgs->newChild` table at zero-based index `pArgs->index`.

Takes `P_TK_TABLE_ADD_AT`, returns `STATUS`.

```c
#define msgTkTableAddAt MakeMsg(clsTkTable, 12)
```

**Arguments**

```c
typedef struct TK_TABLE_ADD_AT {
  WIN newChild;       // new child to add
  U16 index;          // zero-based desired index of newChild
  U32 spare;          // unused (reserved)
} TK_TABLE_ADD_AT, *P_TK_TABLE_ADD_AT;
```

**msgTkTableRemove**

Extracts `pArgs` from the table.

Takes `WIN`, returns `STATUS`.

```c
#define msgTkTableRemove MakeMsg(clsTkTable, 13)
```

**msgTkTableInit**

Sent to `TK_TABLE_ENTRY.class` after default entry-to-`pChildNew` mappings.

Takes `P_TK_TABLE_INIT`, returns `STATUS`. Category: third-party notification.

```c
#define msgTkTableInit MsgNoError(MakeMsg(clsTkTable, 15))
```

**Arguments**

```c
typedef struct TK_TABLE_INIT {
  P_TK_TABLE_NEW pTkTableNew;  // in: tkTable traversing the entry
  P UNKNOWN pChildNew;         // in: child new struct
  P_TK_TABLE_ENTRY pEntry;     // in: this entry; out: last entry used
  U32 spare;                   // unused (reserved)
} TK_TABLE_INIT, *P_TK_TABLE_INIT;
```

**Comments**

The receiver should be sure to advance `pArgs->pEntry` to the last entry used.
TkTableFillArrayWithFonts
Fills in an array of entries with the names of the currently installed fonts.

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED TkTableFillArrayWithFonts (  
  OS_HEAP_ID heapId, // In: heap from which to allocate entries  
  UI8 prune, // In: controls pruning (see fontmgr.h)  
  P_TK_TABLE_ENTRY * ppEntries // Out: pointer to array of entries
);
```

Comments
This function allocates an array of TK_TABLE_ENTRY's from the heap given and then fills it in with the names of the fonts that are currently installed on the machine. The function sets each field of every entry to null except for arg1, which is set to point at a string allocated from the given heap. It is the client's responsibility to free this array and its strings when done using it. clsTkTable provides the utility function TkTableFreeArray() for freeing this allocated storage.

This function also sets the tag field of each entry to be the FIM_SHORT_ID of the corresponding font.

TkTableFreeArray
Frees an array of TK_TABLE_ENTRY's allocated by TkTableFillArrayWithFonts().

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED TkTableFreeArray (  
  P_TK_TABLE_ENTRY pEntries // In: pointer to array of entries
);
```

Comments
This function enumerates an array of TK_TABLE_ENTRY's, frees the string pointed to by the arg1 fields, and then frees the array itself. This function is meant to be used in concert with TkTableFillArrayWithFonts().

Messages from Other Classes

msgFree
Sent as the last of three msgs to destroy an object.

Takes OBJ_KEY, returns STATUS.

Comments
Note that clsTkTable does not destroy metrics.manager.

msgSave
Causes an object to file itself in an object file.

Takes P_OBJ_SAVE, returns STATUS.

Comments
Note that clsTkTable will not save metrics.manager.

msgControlGetClient
Passes back the control's client.

Takes P_UID, returns STATUS.

Comments
clsTkTable responds as in msgTkTableGetClient.
**msgControlSetClient**
Sets the control's client.
Takes UID, returns STATUS.

**Comments**
clsTkTable responds as in msgTkTableSetClient.

**msgControlGetDirty**
Passes back true if the control has been altered since dirty was set false.
Takes P_BOOLEAN, returns STATUS.

```c
#define msgControlGetDirty MakeMsg(clsControl, 15)
```

**Comments**
clsTkTable passes back true if any child is dirty. Each child is sent msgControlGetDirty.

**msgControlSetDirty**
Clears/sets the control's dirty bit.
Takes BOOLEAN, returns STATUS.

**Comments**
clsTkTable sets the dirty bit on each child by sending msgControlSetDirty to each child.

**msgWinSend**
Sends a message up a window ancestry chain.
Takes WIN_SEND, returns STATUS.

**Comments**
clsTkTable will pass msgWinSend on to the tkTable's manager.
If metrics.manager is objNull, does nothing and calls ancestor.
Sends msgWinSend(pArgs) to metrics.manager. If the manager returns stsManagerContinue, calls ancestor; otherwise returns manager's return status.
This file contains the API definition for cIsTableLayout. cIsTableLayout inherits from cIsBorder. Table layout windows position (and optionally size) their child windows in a grid whose parameters you specify.

**Debugging Flags**

The cIsTableLayout debugging flag is '0'. Defined values are:

- `flag4 (0x0010)` msgWinLayoutSelf info
- `flag7 (0x0080)` layout timing

```c
#ifndef TLAYOUT_INCLUDED
#define TLAYOUT_INCLUDED

#include <border.h>

#define tlAlignLeft 0          // left-justified
#define tlAlignCenter 1        // centered
#define tlAlignRight 2         // right-justified
#define tlAlignBottom tAlignLeft // bottom-justified
#define tlAlignTop tAlignRight // top-justified
#define tlAlignBaseline 3      // vertical/horizontal baseline aligned

#ifndef BORDER_INCLUDED
#endif

Common #defines and typedefs  ***********

typedef OBJECT TBL_LAYOUT;
```

**X and Y Alignment Styles**

```c
#define tlAlignLeft 0          // left-justified
#define tlAlignCenter 1        // centered
#define tlAlignRight 2         // right-justified
#define tlAlignBottom tAlignLeft // bottom-justified
#define tlAlignTop tAlignRight // top-justified
#define tlAlignBaseline 3      // vertical/horizontal baseline aligned
```

**Placement Styles**

```c
#define tlPlaceRowMajor 0      // across each row first
#define tlPlaceColMajor 1      // down each column first
#define tlPlaceStack 2         // stack on top of each other
#define tlPlaceOrientation 3   // landscape: RowMajor, portrait: ColMajor
```

**Extra Space Styles**

```c
#define tlExtraNone 0          // leave extra space alone
#define tlExtraFirst 1         // add extra space to 1st row/col
#define tlExtraAfterFirst 2    // put extra space after 1st row/col
#define tlExtraLast 3          // add extra space to last row/col
#define tlExtraBeforeLast 4    // put extra space before last row/col
#define tlExtraAll 5           // add extra space evenly to each row/col
#define tlExtraBetweenAll 6    // divide extra space after each row/col
#define tlExtraBetweenAll 7    // unused (reserved)
#define tlExtraBetweenAll 8    // unused (reserved)
#define tlExtraBetweenAll 15   // unused (reserved)
```
typedef struct TBL_LAYOUT_STYLE {
  U16 tblXAlignment : 2,  // table x alignment within window
  tblYAlignment : 2,       // table y alignment within window
  childXAlignment : 2,     // child x alignment within grid cell
  childYAlignment : 2,     // child y alignment within grid cell
  placement : 2,           // order for placing children in the table
  growChildWidth : 1,      // true to size child to col width
  growChildHeight : 1,     // true to size child to row height
  senseOrientation : 1,    // adjust according to current orientation
  reverseX : 1,            // layout from right to left
  reverseY : 1,            // layout from bottom to top
  wrap : 1;                // wrap around row/column
  U16 widthExtra : 4,      // what to do with extra width
  heightExtra : 4,         // what to do with extra height
  spare1 : 8;              // unused (reserved)
} TBL_LAYOUT_STYLE, *P_TBL_LAYOUT_STYLE;

Default TBL_LAYOUT_STYLE:

tblXAlignment = tlLeft
tblYAlignment = tlTop
childXAlignment = tlLeft
childYAlignment = tlBottom
growChildWidth = true
growChildHeight = true
placement = tlPlaceRowMajor
reverseX = false
reverseY = false
widthExtra = tlExtraNone
heightExtra = tlExtraNone

constraints for Table Layout

Enum16 (TBL_LAYOUT_CONSTRAINT) {
  // for numRows, numCols, colWidth, rowHeight
  tlAbsolute = 0,         // fixed
  // for colWidth, rowHeight; can also or-in tlBaselineBox
  tlChildrenMax = 1,     // max of all children
  tlGroupMax = 2,        // max of all children on same row/column
  // for numRows, numCols, colWidth, rowHeight
  tlMaxFit = 3,          // as many rows/cols as fit given current
                         // rowHeight, colWidth, gaps, and parent size
  // or as wide a col/high a row as possible
  // given current numRows, numCols
  // for numRows, numCols
  tlInfinite = 4,        // unbounded number of rows/cols
};

The following can be OR’ed into tlChildrenMax or tlGroupMax to use max. ascender and descender of
each child. Note: not implemented for tlChildrenMax

#define tlBaselineBox flag7

The following can be OR’ed into any colWidth/rowHeight constraint to use the provided baseline
rather than the max. baseline

Note: not implemented yet.

#define tlAbsoluteBaseline flag6

The following can be OR’ed into any colWidth/rowHeight constraint to use tlMaxFit if the
width/height is constrained during layout (i.e. wsLayoutResize is off or wsShrinkWrapWidth/Height is
off).

#define tlMaxFitIfConstrained flag8

macros to extract the parts of a constraint
```c
#define T1Constraint(c) ((c) & 0xF)

typedef struct TBL_LAYOUT_COUNT {
    TBL_LAYOUT_CONSTRAINT constraint; // see above
    S16 value; // absolute value
    U32 spare; // unused (reserved)
} TBL_LAYOUT_COUNT, *P_TBL_LAYOUT_COUNT;

typedef struct TBL_LAYOUT_SIZE {
    TBL_LAYOUT_CONSTRAINT constraint; // see above
    S16 value; // absolute value
    S16 gap; // space between rows/columns
    S16 baseline; // absolute baseline (not implemented)
    U16 valueUnits : 6, // units for value/gap/baseline
             : 10; // (e.g. bsUnitsLayout)
    U32 spare; // unused (reserved)
} TBL_LAYOUT_SIZE, *P_TBL_LAYOUT_SIZE;

typedef struct TBL_LAYOUT_METRICS {
    TBL_LAYOUT_COUNT numRows, numCols;
    TBL_LAYOUT_SIZE rowHeight, colWidth;
    TBL_LAYOUT_STYLE style;
    U32 spare; // unused (reserved)
} TBL_LAYOUT_METRICS, *P_TBL_LAYOUT_METRICS;
```

### Status Values

These are possible return values from `msgWinLayoutSelf`

- `#define stsTblLayoutLoop` MakeStatus(clsTableLayout, 1)
- `#define stsTblLayoutBadConstraint` MakeStatus(clsTableLayout, 2)

---

**msgNew**

Creates a table layout window.

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

```c
typedef TBL_LAYOUT_METRICS TBL_LAYOUT_NEW_ONLY, *P_TBL_LAYOUT_NEW_ONLY;
#define tableLayoutNewFields |
    borderNewFields |
    TBL_LAYOUT_NEW_ONLY tableLayout;
```

**Arguments**

- `typedef struct {
    tableLayoutNewFields
} TBL_LAYOUT_NEW, *P_TBL_LAYOUT_NEW;`

**Comments**

You first create a table layout window, then insert the children, then send `msgWinLayout` to layout the children.

Note: if you are using `tlAlignBaseline` for the childX/YAlignment, you must use a `colWidth/rowHeight` constraint of `tlGroupMax` | `tlBaselineBox`. Baseline alignment is not implemented with other `colWidth` or `rowHeight` constraints.

**See Also**

- `msgWinLayoutSelf`

---

**msgNewDefaults**

Initializes the `TBL_LAYOUT_NEW` structure to default values.

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

```c
typedef struct {
    tableLayoutNewFields
} TBL_LAYOUT_NEW, *P_TBL_LAYOUT_NEW;
```
Zeroes out pArgs->tableLayout and sets
pArgs->win.flags.style |=
    wsShrinkWrapWidth | wsShrinkWrapHeight | wsFileInline;
pArgs->tableLayout.style.tblXAlignment = tlAlignLeft;
pArgs->tableLayout.style.tblYAlignment = tlAlignTop;
pArgs->tableLayout.style.chfldXAlignment = tlAlignLeft;
pArgs->tableLayout.style.chfldYAlignment = tlAlignBottom;
pArgs->tableLayout.style-growChildWidth = true;
pArgs->tableLayout.style-growChildHeight = true;

// Default is horizontal layout.
pArgs->tableLayout.numRows.constraint = tlAbsolute;
pArgs->tableLayout.numRows.value = 1;
pArgs->tableLayout.numCols.constraint = tlInfinite;
pArgs->tableLayout.numCols.value = 0;
pArgs->tableLayout.rowHeight.constraint = tlChildrenMax;
pArgs->tableLayout.rowHeight.value = 0;
pArgs->tableLayout.rowHeight.gap = 0;
pArgs->tableLayout.colWidth.constraint = tlGroupMax;
pArgs->tableLayout.colWidth.value = 0;
pArgs->tableLayout.colWidth.gap = 0;

**msg TblLayoutGetMetrics**
Passes back current metrics.
Takes P_TBL_LAYOUT_METRICS, returns STATUS.

*define msgTblLayoutGetMetrics MakeMsg(clsTableLayout, 1)

```c
typedef struct TBL_LAYOUT_METRICS {
    TBL_LAYOUT_COUNT numRows, numCols;
    TBL_LAYOUT_SIZE rowHeight, colWidth;
    TBL_LAYOUT_STYLE style;
    U32 spare; // unused (reserved)
} TBL_LAYOUT_METRICS, *P_TBL_LAYOUT_METRICS;
```

**msg TblLayoutSetMetrics**
Sets current metrics.
Takes P_TBL_LAYOUT_METRICS, returns STATUS.

*define msgTblLayoutSetMetrics MakeMsg(clsTableLayout, 2)

```c
typedef struct TBL_LAYOUT_METRICS {
    TBL_LAYOUT_COUNT numRows, numCols;
    TBL_LAYOUT_SIZE rowHeight, colWidth;
    TBL_LAYOUT_STYLE style;
    U32 spare; // unused (reserved)
} TBL_LAYOUT_METRICS, *P_TBL_LAYOUT_METRICS;
```

**clsTableLayout** self-sends msgWinLayoutDirty(true).

**msg TblLayoutGetStyle**
Passes back current style values.
Takes P_TBL_LAYOUT_STYLE, returns STATUS.

*define msgTblLayoutGetStyle MakeMsg(clsTableLayout, 3)
typedef struct TBL_LAYOUT_STYLE {
    U16 tblXAlignment : 2,   // table x alignment within window
    tblYAlignment : 2,   // table y alignment within window
    childXAlignment : 2,   // child x alignment within grid cell
    childYAlignment : 2,   // child y alignment within grid cell
    placement : 2,     // order for placing children in the table
    growChildWidth : 1,   // true to size child to col width
    growChildHeight : 1,   // true to size child to row height
    senseOrientation: 1,   // adjust according to current orientation
    reverseX : 1,   // layout from right to left
    reverseY : 1,   // layout from bottom to top
    wrap : 1;     // wrap around row/column
    U16 widthExtra : 4,   // what to do with extra width
    heightExtra : 4,   // what to do with extra height
    spare : 8;     // unused (reserved)
} TBL_LAYOUT_STYLE, *p TBL_LAYOUT_STYLE;

msgTblLayoutSetStyle

Sets style values.

Takes P_TBL_LAYOUT_STYLE, returns STATUS.

#define msgTblLayoutSetStyle MakeMsg(clsTableLayout, 4)

typedef struct TBL_LAYOUT_INDEX {
    XY32 xy;   // In: table-relative coords
    U16 index;   // Out: zero-based position at which to insert a child
    U32 spare;   // unused (reserved)
} TBL_LAYOUT_INDEX, *P_TBL_LAYOUT_INDEX;

The index returned is such that if a child were inserted there and the table layed out, that child would be at the given xy.
**msgTblLayoutAdjustSections**

Adjusts the border edges and margins of children to correctly reflect a sectioned table.

Takes BOOLEAN, returns STATUS.

```c
#define msgTblLayoutAdjustSections MakeMsg(clsTableLayout, 6)
```

**Comments**

If you have a table layout window in one column and many rows, and the children have top or bottom border edges on to demarcate groups, you should send `msgTblLayoutAdjustSections` to the table layout window after you add or remove children. `clsTableLayout` will turn off borders that are not needed.

If the table needs to be relayed out, `msgWinLayout` will be self-sent if `pArgs` is true; otherwise `msgWinSetLayoutDirty(true)` will be self-sent.

Note that the current implementation assumes the table is one column, infinite rows.

**msgTblLayoutComputeGrid**

Computes the table grid parameters given the current constraints.

Takes P_TBL_LAYOUT_GRID, returns STATUS.

```c
#define msgTblLayoutComputeGrid MakeMsg(clsTableLayout, 7)
#define tblLayoutAvgChildren 10
```

**Arguments**

```c
typedef struct TBL_LAYOUT_GRID_VALUE {
    S32   value; // value in device units
    S32   maxBaseline; // max. baseline for the column/row
    S32   gap; // gap after row/col, in device units
    U32   spare; // unused (reserved)
} TBL_LAYOUT_GRID_VALUE, *P_TBL_LAYOUT_GRID_VALUE;
```

```c
typedef struct TBL_LAYOUT_GRID {
    U16   numCols; // # of columns
    U16   numRows; // # of rows
    S32   colWidth; // column width if pColWidths is pNull
    S32   rowHeight; // row height if pRowHeights is pNull
    P_TBL_LAYOUT_GRID_VALUE   pColWidths; // per-column widths, if not pNull
    P_TBL_LAYOUT_GRID_VALUE   pRowHeights; // per-row heights, if not pNull
    TBL_LAYOUT_METRICS   metrics; // actual metrics
    SIZE32   gap; // col/row gap, in device units
    U8    placement; // actual placement
    XY32   xy; // 1st grid cell in parent space
    // default storage for column widths, row heights
    TBL_LAYOUT_GRID_VALUE   colWidthBuf[tblLayoutAvgChildren];
    TBL_LAYOUT_GRID_VALUE   rowHeightBuf[tblLayoutAvgChildren];
    P_UNKNOWN pData; // reserved for clsTableLayout
    U32   spare1; // unused (reserved)
    U32   spare2; // unused (reserved)
} TBL_LAYOUT_GRID, *P_TBL_LAYOUT_GRID;
```

**Comments**

This message is self-sent by `clsTableLayout` in response to `msgWinLayoutSelf`. `clsTableLayout` responds by computing all of the grid information based on the current `TBL_LAYOUT_METRICS` and current children.

You can send this message at any time to determine the grid parameters.

When you send `msgTblLayoutComputeGrid`, you must set `pArgs->pData` to `pNull`.

You should send `msgTblLayoutFreeGrid(pArgs)` when finished to free any storage allocated by `msgTblLayoutComputeGrid`.

If you subclass `clsTableLayout`, you can respond to this message and compute custom grid parameters (e.g. different per-column absolute column widths).
Note that pArgs->xy is not computed here. The location of the first grid cell can be computed by sending msgTblLayoutComputeGridXY.

msgTblLayoutFreeGrid

typical number of children in a table layout window

msgTblLayoutComputeGridXY

Computes the table grid start xy given the other grid parameters.

Takes P_TBL_LAYOUT_GRID, returns STATUS.

#define msgTblLayoutComputeGridXY MakeMsg(clsTableLayout, 8)

typedef struct TBL_LAYOUT_GRID {
    U16 numCols; // # of columns
    U16 numRows; // # of rows
    S32 colWidth; // column width if pColWidths is pNull
    S32 rowHeight; // row height if pRowHeights is pNull
    P_TBL_LAYOUT_GRID_VALUE pColWidths; // per-column widths, if not pNull
    P_TBL_LAYOUT_GRID_VALUE pRowHeights; // per-row heights, if not pNull
    TBL_LAYOUT_METRICS metrics; // actual metrics
    SIZE32 gap; // col/row gap, in device units
    U8 placement; // actual placement
    XY32 xy; // 1st grid cell in parent space
    // default storage for column widths, row heights
    TBL_LAYOUT_GRID_VALUE colWidthBuf[tblLayoutAvgChildren];
    TBL_LAYOUT_GRID_VALUE rowHeightBuf[tblLayoutAvgChildren];
    P_UNKNOWN pData; // reserved for clsTableLayout
    U32 spare1; // unused (reserved)
    U32 spare2; // unused (reserved)
} TBL_LAYOUT_GRID, *P_TBL_LAYOUT_GRID;

This message is self-sent by clsTableLayout in response to msgWinLayoutSelf. clsTableLayout responds by computing the lower-left of the first grid cell given the specified grid information.

You should first send msgTblLayoutComputeGrid(pArgs) to compute the grid parameters, then send msgTblLayoutComputeGridXY to determine the location of the first cell.

If style.reverseX is true, the first grid cell is actually at pArgs->xy.x - pArgs->colWidth.

If style.reverseY is true, the first grid cell is actually at pArgs->xy.y - pArgs->rowHeight.

If you subclass clsTableLayout, you can respond to this message and compute a custom grid starting location (e.g. something not based on style.tblXAlignment or style.tblYAlignment).

msgTblLayoutComputeGrid

msgTblLayoutFreeGrid

Frees any storage allocated by msgTblLayoutComputeGrid.

Takes P_TBL_LAYOUT_GRID, returns STATUS.

#define msgTblLayoutFreeGrid MakeMsg(clsTableLayout, 9)

typedef struct TBL_LAYOUT_GRID {
    U16 numCols; // # of columns
    U16 numRows; // # of rows
    S32 colWidth; // column width if pColWidths is pNull
    S32 rowHeight; // row height if pRowHeights is pNull
    P_TBL_LAYOUT_GRID_VALUE pColWidths; // per-column widths, if not pNull
    P_TBL_LAYOUT_GRID_VALUE pRowHeights; // per-row heights, if not pNull
    TBL_LAYOUT_METRICS metrics; // actual metrics
} TBL_LAYOUT_GRID, *P_TBL_LAYOUT_GRID;
SIZE32 gap; // col/row gap, in device units
U8 placement; // actual placement
XY32 xy; // 1st grid cell in parent space
// default storage for column widths, row heights
TBL_LAYOUT_GRID_VALUE colWidthBuf[tblLayoutAvgChildren];
TBL_LAYOUT_GRID_VALUE rowHeightBuf[tblLayoutAvgChildren];
P_UNKNOWN pData; // reserved for clsTableLayout
U32 spare1; // unused (reserved)
U32 spare2; // unused (reserved)

} TBL_LAYOUT_GRID, *P_TBL_LAYOUT_GRID;

This message is self-sent by clsTableLayout after self-sending msgTblLayoutComputeGrid.
You should send msgTblLayoutFreeGrid when finished with the grid information computed using
msgTblLayoutComputeGrid to free any storage allocated by msgTblLayoutComputeGrid.

See Also
msgTblLayoutComputeGrid

## Messages from other classes

### msgRestore
Creates and restores an object from an object file.

Takes P_OBJ_RESTORE, returns STATUS.

Comments
clsTableLayout will self-send msgWinSetLayoutDirty(true) if the system font or system font scale
changed since the table was filed. pArgs->pEnv is cast to a P_WIN_RESTORE_ENV and must be a valid
window environment pointer.

### msgWinLayoutSelf
Tell a window to layout its children (sent during layout).

Takes P_WIN_METRICS, returns STATUS.

Comments
clsTableLayout responds by laying out its children. The grid cells of the table are computed based on the
TBL_LAYOUT_METRICS specified. Each child is placed in the corresponding grid cell.
clsTableLayout will self-send msgTblLayoutComputeGrid to compute the grid in which the children
will be placed. msgTblLayoutComputeGridXY will be self-sent to determine the origin of the grid in
self’s window.

The number of columns and rows are computed based on the numCols and numRows constraints. The
width and height of each column and row are computed based on the colWidth and rowHeight
constraints.

The children are placed according to style.placement. For example, if style.placement is
tlPlaceRowMajor, the children are placed across the first row, then the next row, etc.. If style.placement
is tlPlaceOrientation, then the placement will be based on the current orientation of self’s window
device:

<table>
<thead>
<tr>
<th>Orientation</th>
<th>Placement</th>
</tr>
</thead>
<tbody>
<tr>
<td>orientPortraitNormal</td>
<td>tlPlaceColMajor</td>
</tr>
<tr>
<td>orientPortraitReverse</td>
<td>tlPlaceColMajor</td>
</tr>
<tr>
<td>orientLandscapeNormal</td>
<td>tlPlaceRowMajor</td>
</tr>
<tr>
<td>orientLandscapeReverse</td>
<td>tlPlaceRowMajor</td>
</tr>
</tbody>
</table>
If `style.senseOrientation` is true and the orientation is Landscape, the layout metrics are "swapped" as follows:

- If `style.placement` is `tlPlaceRowMajor`, `tlPlaceColMajor` is used.
- If `style.placement` is `tlPlaceColMajor`, `tlPlaceRowMajor` is used.

`metrics.numRows` and `metrics.numCols` are swapped. `rowHeight` and `metrics.colWidth` are swapped.

So if you want a layout that is sensitive to the orientation, set the constraints to make sense for Portrait orientation and turn on `style.senseOrientation`. If the orientation is Landscape when the window is laid out, the metrics will be altered for you.

Within each grid cell, each child is aligned according to `style.childXAlignment` and `style.yAlignment`. For example, if `style.childXAlignment` and `style.childYAlignment` are both `tlAlignCenter`, the children are centered in each grid cell.

If `style.growChildWidth/Height` is true, the width/height of each child is set to the width/height of the child's grid cell.

The entire table is aligned within self according to `style.tblXAlignment` and `style.tblYAlignment`. For example, if `style.tblXAlignment` and `style.tblYAlignment` are both `tlAlignCenter`, the table is centered in self's window.

The rows and columns of the table are normally filled out top to bottom, left to right. If `style.reverseY` is true, the rows are filled out bottom to top. If `style.reverseX` is true, the columns are filled out right to left.

If `pArgs->options` has `wsLayoutResize` on and self has `shrink wrap width/height` on, the width and height of the resulting table will be passed back in `pArgs->bounds.size`.

**Return Value**

- `stsTblLayoutLoop` The specified set of constraints results in a circular layout loop. For example, `tlMaxFit` for `numCols` and `tlMaxFit` for `colWidth`.
- `stsTblLayoutBadConstraint` A constraint specified is not a valid value.

**msgWinGetBaseline**

Gets the desired x,y alignment of a window.

Takes `P_WIN_METRICS`, returns `STATUS`.

**Comments**

If the table is one column, `clsTableLayout` will return the x-baseline of the first child in the table (i.e. send `msgWinGetBaseline` to the first child). Otherwise the x-baseline will be zero.

If the table is one row, `clsTableLayout` will return the y-baseline of the first child in the table (i.e. send `msgWinGetBaseline` to the first child). Otherwise the y-baseline will be zero.

**msgControlEnable**

The control re-evaluates whether it is enabled.

Takes `P_CONTROL_ENABLE`, returns `STATUS`.

**Comments**

`clsTableLayout` recursively enumerates its children (i.e. `wsEnumRecursive` option to `msgWinEnum`) and forwards this message to each child that inherits from `clsControl`. This allows each control in the table to respond to alter its enabled state.

This is used by, for example, `clsMenuButton` when `menuButton.style.enableMenu` is set to true.

**See Also**

`clsMenuButton`
This file contains the API definition for clsTrack. clsTrack inherits from clsObject.

Provides transient drawing feedback for various pen dragging situations, such as resizing and dragging frames.

**Debugging Flags**

The clsTrack debugging flag is 'K'. Defined values are:

flag15 (0x8000) general debug info

```c
ifndef TRACK INCLUDED
#define TRACK_INCLUDED

ifndef WIN_INCLUDED
#include <win.h>
endif
```

Common #defines and typedefs **********

**Track Styles**

```c
#define tsTrackMove 0
#define tsTrackResize 1
```

**Anchor Styles**

```c
#define tsAnchorUL 0 // upper-left
#define tsAnchorUR 1 // upper-right
#define tsAnchorLR 2 // lower-right
#define tsAnchorLL 3 // lower-left
```

**Draw Styles**

```c
#define tsDrawRect 0 // simple rectangle
#define tsDrawTabBarRect 1 // rectangle with vertical tab bar on right
#define tsDrawCmdBarRect 2 // rectangle with command bar at bottom
#define tsDrawTabCmdBarRect 3 // rectangle with both tab and command bars
#define tsdrawBitmap 4 // not implemented
#define tsDrawViaMessages 5 // forward msgTrackShow/Hide to client
#define tsDrawDoubleRect 6 // double rect as in clsBorder double thickness
```

**Thickness Styles**

```c
#define tsThicknessSingle 0 // single-thick lines
#define tsThicknessDouble 1 // double-thick lines
```
Line Pattern Styles

```c
#define tsPatForeground 0  // foreground ink
#define tsPatDashed 1  // sysDCPathLD50
  //
#define tsPatDashed 2  // unused (reserved)
  //
typedef struct TRACK_STYLE {
    U16 track : 2,  // track style (move or resize)
    anchor : 2,  // corner to anchor (tsTrackResize only)
    draw : 4,  // visual to draw
    update : 1,  // send msgTrackUpdate to client
    autoDestroy : 1,  // destroy self when done
    thickness : 2,  // thickness of drawn lines
    pattern : 2,  // line pattern of drawn lines
    startThickness : 2,  // thickness of initial drawn lines
    U16 useThreshold : 1,  // start tracking after msgPenMoveDown
    spare : 15;  // reserved
} TRACK_STYLE, *P_TRACK_STYLE;
```

msgNew

Creates a tracker.

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

Arguments

```c
typedef struct TRACK_NEW_ONLY {
    TRACK_STYLE style;
    WIN win;  // objNull means use theRootWindow
    OBJECT client;  // client to send msgTrackDone to
    P_UNKNOWN image;  // optional image instead of box (not implemented)
    P_UNKNOWN clientData;  // data for client to set
    OBJECT tracker;  // ignored in msgInit
    RECT32 initRect;  // in device units, relative to win
    RECT32 rect;  // in device units, relative to win
    S32 tabBarW;  // tsDrawTabBarRect | tsDrawTabCmdBarRect
    S32 cmdBarH;  // tsDrawCmdBarRect | tsDrawTabCmdBarRect
    XY32 origXY;  // in device units, relative to win
    XY32 curXY;  // in device units, relative to win
    TAG tag;  // optional distinguishing tag
    // if tsTrackMove
    RECT32 keepRect;  // in device units, relative to win
    RECT32 constrainRect;  // in device units, relative to win
    // if tsTrackResize
    SIZE32 minWH;  // in device units
    SIZE32 maxWH;  // in device units
    U32 spare;  // unused (reserved)
    U32 spare1;  // unused (reserved)
} TRACK_METRICS, *P_TRACK_METRICS,
TRACK_NEW_ONLY, *P_TRACK_NEW_ONLY;
```

```c
#define trackNewFields \
    objectNewFields \
    TRACK_NEW_ONLY track;
```

Comments

Note that if you change the default value for `pArgs->track.constrainRect` you should also insure `pArgs->track.keepRect` is correct for your new `constrainRect`.

Here is some sample code for creating an instance of `clsTrack` to resize a window. This is taken from `clsGrabBox`. `plinst->client` is the window to be resized.

```c
// distance to stay away from edge of parent after resize, in device units
#define trBottomParentMargin 0
#define trRightParentMargin 0
```

// min. distance from bottom of child to top of parent, in device units
#define trTopParentMargin 12

// min. distance from right of child to left of parent, in device units
#define trLeftParentMargin 12

// absolute minimum resize width and height, in device units
#define trMinResizeWidth 20
#define trMinResizeHeight 20

TRACK_NEW tn;

// start a resize tracker
ObjCallRet(msgNewDefaults, clsTrack, &tn, s);

ObjCallRet(msgWinGetMetrics, pInst->client, &wm, s);

tn.track.style.track = tsTrackResize;
tn.track.win = wm.parent;
tn.track.client = self;
// window being resized
tn.track.clientData = pInst->client;
tn.track.tag = tagBorderResize;
tn.track.initRect = wm.bounds;

// don't allow the grabbox to go off the edge of client's parent
ObjCallRet(msgWinGetMetrics, wm.parent, &rm, s);

tn.track.maxWH.w = rm.bounds.size.w - trRightParentMargin -
wm.bounds.origin.x;

tn.track.maxWH.h = RectTop(&wm.bounds) - trBottomParentMargin;

tn.track.minWH.w = RectRight(&wm.bounds) - (rm.bounds.size.w - trLeftParentMargin);

tn.track.minWH.h = RectTop(&wm.bounds) - (rm.bounds.size.h - trTopParentMargin);

tn.track.minWH.w = Max(tn.track.minWH.w, trMinResizeWidth);

switch (pInst->style.loc) {
  case gbLocTopEdge:
  case gbLocULCorner:
    tn.track.style.anchor = tsAnchorLR;
    break;

  case gbLocRightEdge:
  case gbLocURCorner:
    tn.track.style.anchor = tsAnchorLL;
    break;

  case gbLocLeftEdge:
  case gbLocLLCorner:
    tn.track.style.anchor = tsAnchorUR;
    break;

  case gbLocBottomEdge:
  default:
    tn.track.style.anchor = tsAnchorUL;
    break;
}

switch (pInst->style.loc) {
  default:
    // unconstrained
    break;
}
case gbLocLeftEdge:
case gbLocRightEdge:
    // constrained to horizontal
    tn.track.minWH.h = wm.bounds.size.h;
    tn.track.maxWH.h = wm.bounds.size.h;
    break;

case gbLocBottomEdge:
case gbLocTopEdge:
    // constrained to vertical
    tn.track.minWH.w = wm.bounds.size.w;
    tn.track.maxWH.w = wm.bounds.size.w;
    break;
}

ObjCallRet(msgTrackProvideMetrics, pInst->client, &tn.track, s);
ObjCallRet(msgNew, clsTrack, &tn, s);

// start tracking at the initial down point
wm.bounds.origin = *pXY;
ObjCallRet(msgWinTransformBounds, theRootWindow, &wm, s);
ObjCallRet(msgTrackStart, tn.object.uid, &wm.bounds.origin, s);

Here is some sample code for creating an instance of clsTrack to drag a window. This is taken from clsBorder. deltaWin is the window to be dragged.

    // keep rect size for drag, in device units
    #define trDefaultMoveKeep 12

    TRACK_NEW
    tn;
    ObjCallRet(msgNewDefaults, clsTrack, &tn, s);

    // constraint to parent's bounds
    ObjSendUpdateRet(msgWinGetMetrics, deltaWin, &clientMetrics, SizeOf(clientMetrics), s);
    if (!clientMetrics.parent)
        return stsOK;
    ObjSendUpdateRet(msgWinGetMetrics, clientMetrics.parent, &wrn, SizeOf(wrn), s);

    tn.track.style.startThickness = tsThicknessDouble;
    tn.track.win = clientMetrics.parent;
    tn.track.client = self;
    tn.track.clientData = deltaWin;
    tn.track.initRect = clientMetrics.bounds;
    tn.track.constrainRect.size = wrn.bounds.size;
    tn.track.tag = tagBorderDrag;

    // start tracking at the initial point
    wm.parent = clientMetrics.parent;
    wm.bounds.origin = *pXY;
    ObjCallRet(msgWinTransformBounds, self, &wm, s);

    tn.track.keepRect.size.w = tn.track.keepRect.size.h = trDefaultMoveKeep;
    tn.track.keepRect.origin = wm.bounds.origin;
    tn.track.keepRect.origin.x -= trDefaultMoveKeep / 2;
    tn.track.keepRect.origin.y -= trDefaultMoveKeep / 2;

    ObjSendUpdateRet(msgTrackProvideMetrics, deltaWin, &tn.track, SizeOf(tn.track), s);
    ObjCallRet(msgNew, clsTrack, &tn, s);

    ObjCallRet(msgTrackStart, tn.object.uid, &wm.bounds.origin, s);
msgNewDefaults
Initializes the TRACK_NEW structure to default values.
Takes P_TRACK_NEW, returns STATUS. Category: class message.

typedef struct TRACK_NEW {
    trackNewFields
} TRACK_NEW, *P_TRACK_NEW;

Sets all of pArgs->track to 0, then ...

    pArgs->object.cap |= objCapCall;
    pArgs->track.style.autoDestroy = true;
    pArgs->track.constrainRect.size.w = maxS32 / 2;
    pArgs->track.constrainRect.size.h = maxS32 / 2;
    pArgs->track.keepRect.origin.x = maxS32 / 4;
    pArgs->track.keepRect.origin.y = maxS32 / 4;
    pArgs->track.keepRect.size.w = 1;
    pArgs->track.keepRect.size.h = 1;
    pArgs->track.maxWH.w = maxS32 / 2;
    pArgs->track.maxWH.h = maxS32 / 2;

Default style:

    track = tsTrackMove
    anchor = tsAnchorUL (ignored when tsTrackMove)
    draw = tsDrawRect
    update = false
    autoDestroy = true
    thickness = tsThicknessSingle
    pattern = tsPatForeground
    startThickness = tsThicknessSingle
    useThreshold = false

msgTrackGetStyle
Passes back current style values.
Takes P_TRACK_STYLE, returns STATUS.

#define msgTrackGetStyle MakeMsg(clsTrack, 1)

typedef struct TRACK_STYLE {
    U16 track : 2, // track style (move or resize)
    anchor : 2, // corner to anchor (tsTrackResize only)
    draw : 4, // visual to draw
    update : 1, // send msgTrackUpdate to client
    autoDestroy : 1, // destroy self when done
    thickness : 2, // thickness of drawn lines
    pattern : 2, // line pattern of drawn lines
    startThickness : 2; // thickness of initial drawn lines
    U16 useThreshold : 1, // start tracking after msgPenMoveDown
    spare : 15; // reserved
} TRACK_STYLE, *P_TRACK_STYLE;

msgTrackSetStyle
Sets style values.
Takes P_TRACK_STYLE, returns STATUS.

#define msgTrackSetStyle MakeMsg(clsTrack, 2)
typedef struct TRACK_STYLE {
    // track style (move or resize)
    U16 track : 2,
    // corner to anchor (tsTrackResize only)
    anchor : 2,
    // visual to draw
    draw : 4,
    // send msgTrackUpdate to client
    update : 1,
    // destroy self when done
    autoDestroy : 1,
    // thickness of drawn lines
    thickness : 2,
    // line pattern of drawn lines
    pattern : 2,
    // thickness of initial drawn lines
    startThickness : 2,
    // start tracking after msgPenMoveDown
    spare : 15,
} TRACK_STYLE, *P_TRACK_STYLE;

If the new style values result in a different visual, and msgTrackStart has been sent, you should first send msgTrackHide with pArgs of the old TRACK_METRICS, then msgTrackSetStyle, then msgTrackShow with the new TRACK_METRICS.

msgTrackGetMetrics

Passes back the current metrics.

Takes P_TRACK_METRICS, returns STATUS.

#define msgTrackGetMetrics MakeMsg(clsTrack, 3)

typedef struct TRACK_NEW_ONLY {
    // track style
    TRACK_STYLE style;
    // objNull means use theRootWindow
    WIN win;
    // client to send msgTrackDone to
    OBJECT client;
    // optional image instead of box (not implemented)
    P_UNKNOWN image;
    // data for client to set
    P_UNKNOWN clientData;
    // ignored in msgInit
    OBJECT tracker;
    // in device units, relative to win
    RECT32 initRect;
    // in device units, relative to win
    RECT32 rect;
    // tsDrawTabBarRect | tsDrawTabCmdBarRect
    S32 tabBarW;
    // tsDrawCmdBarRect
    S32 cmdBarH;
    // in device units, relative to win
    XY32 origXY;
    // in device units, relative to win
    XY32 curXY;
    // optional distinguishing tag
    TAG tag;
    // if tsTrackMove
    RECT32 keepRect;
    // if tsTrackResize
    SIZE32 minWH;
    // in device units
    SIZE32 maxWH;
    // unused (reserved)
    U32 spare;
    // unused (reserved)
    U32 spare1;
} TRACK_METRICS, *P_TRACK_METRICS,
### TRACK.H

**Client Messages**

#### msgTrackStart

Starts the tracker.

Takes p_XY32, returns STATUS.

```c
#define msgTrackStart MakeMsg(clsTrack, 5)
```

**Comments**

The pArgs indicates the initial position of the pen (in device units, in the space of the metrics.win). If pArgs is pNull, then metrics.origXY is used as the initial pen position.

clsTrack will do the following:

- self-send msgTrackConstrain to constrain the initial point.
- grab all input events using InputSetGrab().
- self-send msgTrackShow(&metrics) to paint the tracker.

#### msgTrackDone

Sent by clsTrack to metrics.client when the track is done.

Takes P_TRACK_METRICS, returns STATUS. Category: client notification.

```c
#define msgTrackDone MakeMsg(clsTrack, 6)
```

**Message Arguments**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRACK_STYLE</td>
<td>style;</td>
</tr>
<tr>
<td>WIN</td>
<td>win;</td>
</tr>
<tr>
<td>OBJECT</td>
<td>client;</td>
</tr>
<tr>
<td>P_UNKNOWN</td>
<td>image;</td>
</tr>
<tr>
<td>P_UNKNOWN</td>
<td>clientData;</td>
</tr>
<tr>
<td>OBJECT</td>
<td>tracker;</td>
</tr>
<tr>
<td>RECT32</td>
<td>initRect;</td>
</tr>
<tr>
<td>RECT32</td>
<td>rect;</td>
</tr>
<tr>
<td>S32</td>
<td>tabBarW;</td>
</tr>
<tr>
<td>S32</td>
<td>cmdBarH;</td>
</tr>
<tr>
<td>XY32</td>
<td>origXY;</td>
</tr>
<tr>
<td>XY32</td>
<td>curXY;</td>
</tr>
<tr>
<td>TAG</td>
<td>tag;</td>
</tr>
<tr>
<td></td>
<td>// if tsTrackMove</td>
</tr>
<tr>
<td>RECT32</td>
<td>keepRect;</td>
</tr>
</tbody>
</table>

See `msgTrackSetStyle` for notes on changing metrics after `msgTrackStart` has been sent.

See Also

`msgTrackSetStyle`
msgTrackUpdate

Sent by clsTrack to metrics.client when the pen moves if style.update is true.

Takes P_TRACK_METRICS, returns STATUS. Category: client notification.

#define msgTrackUpdate MakeMsg(clsTrack, 7)

msgTrackProvideMetrics

Sent to a tracker client before tracker is created.

Takes P_TRACK_METRICS, returns STATUS. Category: third-party notification.

#define msgTrackProvideMetrics MsgNoError(MakeMsg(clsTrack, 9))
Before it sends msgNew to clsTrack, code creating a tracker may choose to send out this message to another object, allowing it to modify the tracker metrics. See frame.h for a sample response to msgTrackProvideMetrics.

**Self-sent Messages**

### msgTrackConstrain

Constrains a point.

Takes P_XY32, returns STATUS. Category: self-sent.

```c
#define msgTrackConstrain MakeMsg(clsTrack, 8)
```

**Comments**

If style. track is tsTrackMove, a new value for metrics.keepRect is computed based on the offset from metrics.origXY to pArgs. pArgs is altered to insure the new keepRect lies within metrics.constrainRect.

If style. track is tsTrackResize, a new value for metrics.rect is computed based on the offset from metrics.origXY to pArgs. pArgs is altered to insure the new rect.size lies within metrics.maxWH and metrics.minWH.

### msgTrackShow

Displays the tracker visuals at pArgs->rect.

Takes P_TRACK_METRICS, returns STATUS. Category: self-sent.

```c
#define msgTrackShow MakeMsg(clsTrack, 10)
```

```c
typedef struct TRACK_NEW_ONLY {
    TRACK_STYLE style;
    WIN win; // objNull means use theRootWindow
    OBJECT client; // client to send msgTrackDone to
    P_UNKNOWN image; // optional image instead of box (not implemented)
    P_UNKNOWN clientData; // data for client to set
    OBJECT tracker; // ignored in msgInit
    RECT32 initRect; // in device units, relative to win
    RECT32 rect; // in device units, relative to win
    S32 tabBarW; // tsDrawTabBarRect | tsDrawTabCmdBarRect
    S32 cmdBarH; // tsDrawCmdBarRect | tsDrawTabCmdBarRect
    XY32 origXY; // in device units, relative to win
    XY32 curXY; // in device units, relative to win
    TAG tag; // optional distinguishing tag
    // if tsTrackMove
    RECT32 keepRect; // in device units, relative to win
    RECT32 constrainRect; // in device units, relative to win
    // if tsTrackResize
    SIZE32 minWH; // in device units
    SIZE32 maxWH; // in device units
    U32 spare; // unused (reserved)
    U32 spare1; // unused (reserved)
} TRACK_METRICS, *P_TRACK_METRICS,
```

**Comments**

clsTrack will self-send this message when the tracker needs to be displayed.
**msgTrackHide**

Removes the tracker visuals at `pArgs->rect`.


```c
#define msgTrackHide MakeMsg(clsTrack, 11)
```

```c
typedef struct TRACK_NEW_ONLY {
    TRACK_STYLE style;
    WIN win;        // objNull means use theRootWindow
    OBJECT client;  // client to send msgTrackDone to
    P_UNKNOWN image;  // optional image instead of box (not implemented)
    P_UNKNOWN clientData; // data for client to set
    OBJECT tracker; // ignored in msgInit
    RECT32 initRect; // in device units, relative to win
    RECT32 rect; // in device units, relative to win
    S32 tabBarW; // tsDrawTabBarRect | tsDrawTabCmdBarRect
    S32 cmdBarH; // tsDrawCmdBarRect | tsDrawTabCmdBarRect
    XY32 origXY; // in device units, relative to win
    XY32 curXY; // in device units, relative to win
    TAG tag; // optional distinguishing tag
    // if tsTrackMove
    RECT32 keepRect; // in device units, relative to win
    RECT32 constrainRect; // in device units, relative to win
    // if tsTrackResize
    SIZE32 minWH; // in device units
    SIZE32 maxWH; // in device units
    U32 spare; // unused (reserved)
    U32 spare1; // unused (reserved)
} TRACK_METRICS, *P_TRACK_METRICS,
```

Comments:

`clsTrack` will self-send this message when the tracker needs to be erased.

---

**Messages from other classes**

**msgInputEvent**

Notification of an input event.

Takes `P_INPUT_EVENT`, returns `STATUS`.

Comments:

`clsTrack` will respond to input events by updating and/or terminating the tracker.

If `pArgs->devCode` is not one of `msgPenMoveDown`, `msgPenUp`, or `msgPenOutProxDown` `statusInputGrabTerminate` is returned.

The new point is constrained by self-sending `msgTrackConstrain`. The new value for `metrics.rect` and `metrics.curXY` is computed based on the constrained `pArgs->xy`.

If `pArgs->devCode` is `msgPenUp` or `msgPenOutProxDown`, `clsTrack` does the following:

- send `msgTrackDone(&metrics)` to `metrics.client`
- self-send `msgTrackHide` to remove the old tracker visuals
- if `style.autoDestroy` is true, self-send `msgDestroy(pNull)`

If `pArgs->devCode` is `msgPenMoveDown`, and the constrained version of `pArgs->xy` is different from `metrics.curXY`, `clsTrack` does the following:

- if `style.update` is true, send `msgTrackUpdate(&metrics)` to `metrics.client`
- self-send `msgTrackHide` to remove the old tracker visuals
- self-send `msgTrackShow` to paint the new tracker visuals
This file contains the API definition for clsToggleTable.

clsToggleTable inherits from clsTkTable.

Toggle tables implement non-exclusive choices.

```c
#ifndef TTABLE_INCLUDED
#define TTABLE_INCLUDED
#include <tktable.h>
#endif
```

## Common #defines and typedefs

ctypedef OBJECT TOGGLE_TABLE;

typedef struct TOGGLE_TABLE_NEW_ONLY {
    U32 spare; // unused (reserved)
} TOGGLE_TABLE_NEW_ONLY, *P_TOGGLE_TABLE_NEW_ONLY;

define toggleTableNewFields

typedef struct TOGGLE_TABLE_NEW {
    toggleTableNewFields
} TOGGLE_TABLE_NEW, *P_TOGGLE_TABLE_NEW;

msgNew

Creates a toggle table window.

Takes P_TOGGLE_TABLE_NEW, returns STATUS. Category: class message.

```c
typedef struct TOGGLE_TABLE_NEW_ONLY {
    U32 spare; // unused (reserved)
} TOGGLE_TABLE_NEW_ONLY, *P_TOGGLE_TABLE_NEW_ONLY;

define toggleTableNewFields

typedef struct TOGGLE_TABLE_NEW {
    toggleTableNewFields
} TOGGLE_TABLE_NEW, *P_TOGGLE_TABLE_NEW;
```

msgNewDefaults

Initializes the TOGGLE_TABLE_NEW structure to default values.

Takes P_TOGGLE_TABLE_NEW, returns STATUS. Category: class message.

```c
typedef struct TOGGLE_TABLE_NEW {
    toggleTableNewFields
} TOGGLE_TABLE_NEW, *P_TOGGLE_TABLE_NEW;
```
pArgs->tableLayout.colWidth.constraint = tlChildrenMax;
pArgs->tableLayout.colWidth.gap = 0;

pArgs->tableLayout.rowHeight.constraint = tlGroupMax;
pArgs->tableLayout.rowHeight.gap = 0;

### Messages from Other Classes

**msgTkTableChildDefaults**
Sets the defaults in P_ARGS for a common child.

Takes P_UNKNOWN, returns STATUS.

**Comments**

Here is how a choice processes this message:

```c
if (<pArgs->object.class inherits from clsGWin>  
  pArgs->gWin.style.gestureEnable = false;

if (<pArgs->object.class inherits from clsBorder> {  
  pArgs->border.style.edge = bsEdgeNone;
  pArgs->border.style.topMargin = 1;
  pArgs->border.style.bottomMargin = 1;
}

if (<pArgs->object.class inherits from clsLabel>  
  pArgs->label.style.xAlignment = lsAlignLeft;

if (<pArgs->object.class inherits from clsButton> {  
  pArgs->button.style.notifyDetail = true;
  pArgs->button.style.contact = bsContactToggle;
  pArgs->button.style.feedback = bsFeedbackDecorate;
  pArgs->button.style.offDecoration =  
    lsDecorationNonExclusiveOff;
  pArgs->button.style.onDecoration =    
    lsDecorationNonExclusiveOn;
```

**msgControlGetDirty**
Passes back the dirty state of the control.

Takes P_BOOLEAN, returns STATUS.

**Comments**

**clsToggleTable** responds by setting *pArgs up as a 32 bit collection of the results of sending msgControlGetDirty to its first 32 children. The result of the first (bottom) child is placed in bit 0, the second in bit 1, and so on.

The resulting *pArgs is undefined if the toggle table has more than 32 children.

**msgControlGetEnable**
Passes back whether the control is enabled.

Takes P_BOOLEAN, returns STATUS.

**Comments**

**clsToggleTable** responds by setting *pArgs up as a 32 bit collection of the results of sending msgControlGetEnable to its first 32 children. The result of the first (bottom) child is placed in bit 0, the second in bit 1, and so on.

The resulting *pArgs is undefined if the toggle table has more than 32 children.
**msgControlGetValue**

Passes back the value of the control.

Takes P_TAG, returns STATUS.

**Comments**

*clsToggleTable* responds by setting *pArgs up as a 32 bit collection of the results of sending msgControlGetValue to its first 32 children. The result of the first (bottom) child is placed in bit 0, the second in bit 1, and so on.

The resulting *pArgs is undefined if the toggle table has more than 32 children.

**msgControlSetDirty**

Sets dirty state of the control.

Takes BOOLEAN, returns STATUS.

**Comments**

*clsToggleTable* treats the pArgs as a 32 bit collection of values to send via msgControlSetDirty to its first 32 children. The value of bit 0 is sent to the first (bottom) child, bit 1 is sent to the second child, and so on.

**msgControlSetEnable**

Sets whether the control is enabled.

Takes BOOLEAN, returns STATUS.

**Comments**

*clsToggleTable* treats the pArgs as a 32 bit collection of values to send via msgControlSetEnable to its first 32 children. The value of bit 0 is sent to the first (bottom) child, bit 1 is sent to the second child, and so on.

**msgControlSetValue**

Sets the value of the control.

Takes TAG, returns STATUS.

**Comments**

*clsToggleTable* treats the pArgs as a 32 bit collection of values to send via msgControlSetValue to its first 32 children. The value of bit 0 is sent to the first (bottom) child, bit 1 is sent to the second child, and so on.
Part 5 / Input and Handwriting Translation
ACETATE.H

Interface file for the acetate. The functions described in this file are contained in INPUTLIB.

WARNING: Inking and the acetate layer are subject to major changes in future releases.

```c
#ifndef ACETATE_INCLUDED
#define ACETATE_INCLUDED
#endif
#ifndef GO_INCLUDED
#include <go.h>
#endif
#ifndef GEO_INCLUDED
#include <geo.h>
#endif

Ilprototypes

AcetateTransform
Converts coordinate to/from screen device root window and pen units.
Returns void.

Function Prototype void EXPORTED AcetateTransform(
    P_XY32 pXY,     // coordinates to transform
    U16 type        // 0 for root-to-pen
                    // 1 for pen-to-root
);

Comments Warning: This works only when transforming to or from the screen device root window. Other transforms must use drawing contexts.

AcetateCursorRequestVisible
Used to request that the cursor turn on or off.
Returns void.

Function Prototype void EXPORTED AcetateCursorRequestVisible(
    BOOLEAN requestVisibleOn
);

AcetateCursorThaw
Unfreezes the cursor for pen movements.
Returns STATUS.

Function Prototype STATUS EXPORTED AcetateCursorThaw(
    void
);
AcetateCursorFreezePosition

Freezes the cursor at the given Root window coordinate until the cursor image is reset to pNull (standard cursor).

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED AcetateCursorFreezePosition(
    P_XY32 pLoc
); // location in the root window
```

AcetateCursorXY

Sets the cursor position.

Returns void.

Function Prototype

```c
void EXPORTED AcetateCursorXY(
    COORD32 x,
    COORD32 y
); // location in the root window
```

AcetateCursorImage

Sets the cursor image.

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED AcetateCursorImage(
    P_UNKNOWN pNewCursor,
    BOOLEAN sticky
); // location in the root window
```

Comments

If pCursor == pNull, resets to the pen cursor and frees the substitute cursor memory.

AcetateCursorUpdateImage

Updates the current cursor image.

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED AcetateCursorUpdateImage(
    P_UNKNOWN pNewCursor
); // location in the root window
```

Comments

This interface should only be used for cursor animations and not to change the actual cursor to a different style.

AcetateClear

Clears (makes transparent) the entire acetate plane.

Returns void.

Function Prototype

```c
void EXPORTED AcetateClear(
    void
); // location in the root window
```

AcetateClearDisable

Used while grabbing to keep the acetate from being cleared.

Returns void.
**Function Prototype**

```c
void EXPORTED AcetateClearDisable(
    void
);
```

**Comments**

Call it during input event processing and return one of the grab status `returnValues`. While the Clear Disable is active, calls to AcetateClear will have no effect. Calls to AcetateClearRect will still work however.

---

**AcetateClearRect**

Clears (makes transparent) the indicated acetate rect. `pNull` implies the entire plane.

Returns void.

```c
void EXPORTED AcetateClearRect (
P_RECT32 pRect
);
```
This file contains the API definitions for clsAnimSPaper. clsAnimSPaper inherits from clsSPaper. Records pen strokes and plays them back at a reduced speed. Provides settable speed, interstroke delay, line attribute and scaling parameters.

Introduction

clsAnimSPaper "animates" the drawing of scribbles by painting a few points, then pausing for the specified number of milliseconds before continuing. The animated playback is performed in a separate task, so playbacks will not disturb other events on the screen. A semaphore is used internally to prevent multiple tasks from painting in the AnimSPaper window simultaneously. The painting task is created whenever playback starts, and terminated when it finishes.

The animation behavior is triggered by msgWinRepaint—that is, whenever the AnimSPaper is asked to paint itself. This means that you'll get slow, "animated" painting regardless of the cause of the msgWinRepaint: layout, resize, scrolling, unclipping, and so forth. If you want slow painting only under certain circumstances (e.g., when the user taps a button), set the Delay and Interstroke parameters to 0, then do this:

```c
OS_MILLISECONDS om;
om = yourDelay;
ObjectCall(msgAnimSPaperSetDelay, animSPaperInstance, &om);
om = yourInterstroke;
ObjectCall(msgAnimSPaperSetInterstroke, animSPaperInstance, &om);
ObjectCall(msgWinDirtyRect, animSPaperInstance, NULL);
ObjectCall(msgWinUpdate, animSPaperInstance, NULL);
om = 0;
ObjectCall(msgAnimSPaperSetDelay, animSPaperInstance, &om);
ObjectCall(msgAnimSPaperSetInterstroke, animSPaperInstance, &om);
```

clsAnimSPaper Parameters

There are four gettable/settable parameters having to do with scribble redisplay. Delay specifies the number of milliseconds to wait between painting line segments. It varies in inversely with the animation speed. Interstroke delay is a separate delay to be used between scribble strokes. It simulates the writer lifting and moving the pen from the end of one stroke to the beginning of the next. Line sets the thickness and other attributes used in playing back the scribble. Generally you shouldn’t need to set anything except thickness. Scale affects the size of the scribbles when they’re played back. The scale parameters will stretch/compress the scribble along the x and y axes, also scaling the scribble's distance from the lower-left corner (0,0). This is especially useful for applications which wish to scale in proportion to the system font size. Note that since scribble scaling is in proportion to the original scribble, you may need to save what the system font size was when the scribble was recorded.
Other Facilities

If pArgs->animSPaper.sendDone is true, an AnimSPaper will send msgAnimSPaperDone to its client when the animation is completed.

For convenience two messages are provided to read and write scribbles to/from resource files.

Note on Delay and Interstroke Parameters

AnimSPaper uses OSTaskDelay() to create the Delay and Interstroke delay. The minimum increment of OSTaskDelay is a system tick (systick), whose length is device dependent. Use OSSystemInfo() to find the length of a systick (see OS.H for details). On an average PC or 386 system the systick is 55 milliseconds, or about an eighteenth of a second. So micro-adjustments of Delay and Interstroke from, say, 60 milliseconds to 80 milliseconds will be ineffective.

Debugging Flags

cclsAnimSPaper uses the Handwriting debug flag set 'Z'. cclsAnimSPaper uses:

80000  Show all internal debugging messages

#ifdef ANIMSP_INCLUDED
#define ANIMSP_INCLUDED
#endif

#include <spaper.h> // ancestor flags
#include <sysgraf.h> // line & scale def'ns
#include <fs.h> // filing def'ns

#define animSPaperNewFields

typedef struct ANIM_SPAPER

Common #defines and typedefs

typedef struct ANIM_SPAPER_NEW ONLY {
    SYSDC_LINE  _line; // line attributes for scribble playback
    OS_MILLISECONDS delay; // delay between stroke segments on playback
                       // (inverse of playback speed)
    OS_MILLISECONDS interstroke; // delay between strokes on playback
    OBJECT client; // recipient of msgAnimSPaperDone
    BOOLEAN sendDone; // if TRUE, animSPaper will send client
                       // msgAnimSPaperDone when animation's done
    SCALE scale; // how much larger or smaller to scale the
                  // scribble when it's played back. (1,1)
                  // plays back at same scale as recorded.
    S32 spare1; // unused (reserved)
    S32 spare2; // unused (reserved)
} ANIM_SPAPER_NEW ONLY, *P_ANIM_SPAPER_NEW ONLY;

#define animSPaperNewFields

typedef struct ANIM_SPAPER_NEW {
    animSPaperNewFields
} ANIM_SPAPER_NEW, *P_ANIM_SPAPER_NEW;
Messages

**msgNew**

Creates an AnimSPaper window.

Takes P_ANIM_SPAPER_NEW, returns STATUS. Category: class message.

```
typedef struct ANIM_SPAPER_NEW {
    AnimSPaperNewFields
} ANIM_SPAPER_NEW, *P_ANIM_SPAPER_NEW;
```

The fields you commonly set are:

- `pArgs->animSPaper.line.thickness`: thickness of line on playback
- `pArgs->animSPaper.delay`: inverse of animation speed
- `pArgs->animSPaper.interstroke`: delay between strokes
- `pArgs->animSPaper.client`: whom to notify when animation is done
- `pArgs->animSPaper.sendDone`: whether to notify client
- `pArgs->animSPaper.scale`: playback size relative to input size

**msgNewDefaults**

Initialize `pArgs`.

Takes P_ANIM_SPAPER_NEW, returns STATUS. Category: class message.

```
typedef struct ANIM_SPAPER_NEW {
    AnimSPaperNewFields
} ANIM_SPAPER_NEW, *P_ANIM_SPAPER_NEW;
```

Sets:

- `pArgs->animSPaper.line.cap` = sysDcCapRound;
- `pArgs->animSPaper.line.join` = sysDcJoinRound;
- `pArgs->animSPaper.line.thickness` = 6;
- `pArgs->animSPaper.line.miterLimit` = 10;
- `pArgs->animSPaper.line.radius` = 0;
- `pArgs->animSPaper.delay` = 40;
- `pArgs->animSPaper.interstroke` = 160;
- `pArgs->animSPaper.client` = objNull;
- `pArgs->animSPaper.sendDone` = TRUE;
- `pArgs->animSPaper.scale.x` = FxIntToFx(1);
- `pArgs->animSPaper.scale.y` = FxIntToFx(1);
- `pArgs->spScribble.flags` &= ~spScribbleEdit
  & ~spRedisplay
  & ~spVRuling
  & ~spRuling
  & ~spBackground);
- `pArgs->win.flags.input` |= inputInkThrough;

**msgAnimSPaperReadScribble**

Reads a scribble from a resource file, sets it into the AnimSPaper and displays it.

Takes P_ANIM_SPAPER_SCRIBBLE, returns STATUS. Category: class message.

```
#define msgAnimSPaperReadScribble MakeMsg(clsAnimSPaper, 1)
```

```
typedef struct ANIM_SPAPER_SCRIBBLE {
    FS_LOCATOR locator;  // resource file locator
    RES_ID resid;        // resource id for the scribble
} ANIM_SPAPER_SCRIBBLE, *P_ANIM_SPAPER_SCRIBBLE;
```
msgAnimSPaperWriteScribble

Writes the AnimSPaper's current scribble to a resource file.
Takes P_ANIM_SPAPER_SCRIBBLE, returns STATUS. Category: class message.
#define msgAnimSPaperWriteScribble MakeMsg(clsAnimSPaper, 2)

typedef struct ANIM_SPAPER_SCRIBBLE {
    FS_LOCATOR locator;    // resource file locator
    RES_ID resid;          // resource id for the scribble
} ANIM_SPAPER_SCRIBBLE, *P_ANIM_SPAPER_SCRIBBLE;

msgAnimSPaperSetDelay

Specifies delay for scribble playback
Takes P_OS_MILLISECONDS, returns STATUS. Category: class message.
#define msgAnimSPaperSetDelay MakeMsg(clsAnimSPaper, 4)

msgAnimSPaperGetDelay

Passes back delay for scribble playback
Takes P_OS_MILLISECONDS, returns STATUS. Category: class message.
#define msgAnimSPaperGetDelay MakeMsg(clsAnimSPaper, 5)

msgAnimSPaperSetInterstroke

Specifies interstroke delay for scribble playback
Takes P_OS_MILLISECONDS, returns STATUS. Category: class message.
#define msgAnimSPaperSetInterstroke MakeMsg(clsAnimSPaper, 6)

msgAnimSPaperGetInterstroke

Passes back interstroke delay for scribble playback
Takes P_OS_MILLISECONDS, returns STATUS. Category: class message.
#define msgAnimSPaperGetInterstroke MakeMsg(clsAnimSPaper, 7)

msgAnimSPaperSetLine

Specifies line attributes for scribble playback
Takes P_SYSDC_LINE, returns STATUS. Category: class message.
#define msgAnimSPaperSetLine MakeMsg(clsAnimSPaper, 8)

msgAnimSPaperGetLine

Passes back line attributes for scribble playback
Takes P_SYSDC_LINE, returns STATUS. Category: class message.
#define msgAnimSPaperGetLine MakeMsg(clsAnimSPaper, 9)
**msgAnimSPaperSetScale**
Specifies scaling for scribble playback.
Takes P_SCALE, returns STATUS. Category: class message.
#define msgAnimSPaperSetScale MakeMsg(clsAnimSPaper, 11)

Comments
The scribble will be played back at a SCALE relative to the size at which it was recorded. X and Y scales may be set independently. The SCALE affects both the scribble and its distance from the lower-left corner (0,0).

**msgAnimSPaperGetScale**
Passes back scaling for scribble playback
Takes P_SCALE, returns STATUS. Category: class message.
#define msgAnimSPaperGetScale MakeMsg(clsAnimSPaper, 12)

**Notifications**

**msgAnimSPaperDone**
Sent to client when animation is complete.
Takes OBJECT, returns STATUS. Category: advisory message.
#define msgAnimSPaperDone MakeMsg(clsAnimSPaper, 3)

Comments
pArgs is the animSPaper's UID. This message is sent only if there is a client and pArgs->animSPaper.sendDone was true at msgNew time.
**GWIN.H**

This file contains the API definition for c1sGWin.

c1sGWin inherits from c1sWin.

### Introduction

c1sGWin provides a convenient default implementation of several important PenPoint features -- gesture and keyboard processing, quick help interaction and event forwarding.

c1sGWin is an ancestor of many of PenPoint's window-based classes, including all of the Toolkit classes.

Many tasks involving the input system and the handwriting recognition system can be handled very simply using only a few c1sGWin messages. Some tasks require use of some of c1sGWin's more sophisticated messages. And there are some task for which c1sGWin is not appropriate. For instance, even a modest drawing application or "ink editor" will almost certainly have to interact more directly with the input system and handwriting recognition system.

Several important task can be accomplished by using just few c1sGWin messages:

- To process gestures, see msgGWinGesture.
- To process keyboard input, see msgGWinKey.
- To implement quick help, use gWin's helpId; see GWIN_NEW_ONLY and msgGWinSetHelpId.
- To process gestures and keyboard events which occurred in child windows, see msgGWinForwardedGesture or msgGWinForwardedKey.
- To control whether or not a window responds to gestures, see the gestureEnable field in GWIN_STYLE.

More complex subclasses will need to understand more details, as described below.

### Debugging Flags

GWin's debugging flag set is '#' (0x23). Defined flags are:

0001 Display generally useful messages.
0004 Display messages during quick help processing.
0010 Display messages during timeout processing.

### Keyboard Processing

Keyboard processing and forwarding occurs when a gWin receives msgInputEvent with a key event message in pArgs->devCode. The steps taken are:

- gWin self sends msgGWinKey with the event.
If the response to msgGWinKey is stsRequestDenied, gWin self sends msgGWinBadKey. gWin's default response to msgGWinKey is to return stsRequestForward, which causes gWin to perform key event forwarding.

- If the response to msgGWinKey is stsRequestForward and style.keyboardForward is set, gWin self sends msgGWinForwardKey. In response to this message, the gWin packages up the data and uses msgWinSend to forward the key information. This results in parent windows potentially receiving msgGWinForwardedKey (see msgGWinForwardedKey description). gWin's default response to msgGWinForwardedKey is to return stsRequestForward, which causes the event forwarding to continue.

- If the response to msgGWinForwardKey is stsRequestDenied or stsRequestForward, gWin self sends msgGWinBadKey.

### Gesture Processing

A gWin self sends msgGWinGesture when one of the following occurs. (Each of these is described more detail below.)

- Case 1: A gWin receives msgGWinXList (typically because a translation has completed).
- Case 2: A gWin receives msgInputEvent with an event of press-hold or a tap-press-hold.
- Case 3: A gWin receives msgQuickHelpHelpShow from the QuickHelpManager.

If the response to msgGWinGesture is stsRequestDenied, the gesture is unrecognized and one of the following actions is taken:

- In Case 1, a translated gesture, msgGWinBadGesture is self sent.
- In Case 2, normal gesture processing continues. This is because a press-hold or a tap-press-hold gesture is sent in response to an input event while potentially in the process of collecting data for another gesture (see below).
- In Case 3, the "no help available" help is displayed via msgQuickHelpShow.

If the response to msgGWinGesture is stsRequestForward, msgGWinForwardGesture is self sent. If the response to msgGWinForwardGesture is stsRequestDenied or stsRequestForward, the same action is taken as if msgGWinGesture returned stsRequestDenied.

Case 1: How a GWin Receives Translated Gestures.

msgGWinGesture is self sent in response to msgGWinXList. msgGWinXList is self sent by gWin after an xGesture translator has completed its translation. This occurs as follows:

When msgPenStroke is received from the input system, the gWin adds strokes to a gesture translator. This is done via a self send of msgGWinStroke, which adds the stroke via sending msgScrAddStroke to the gesture translator.

\[
gWin \rightarrow msgScrAddStroke \rightarrow xGesture \text{ Translator}
\]

When an "out of proximity" event is received, gWin self sends msgGWinComplete. In response to the msgGWinComplete, gWin sends msgScrComplete to the gesture translator.

\[
gWin \rightarrow msgScrComplete \rightarrow xGesture \text{ Translator}
\]

The translator then sends msgXlateCompleted back to the gWin, indicating translation is complete. GWin retrieves translated results by sending msgXlateData to the gesture translator.
This returns an xlist containing the translated data (see xlist.h). GWin then self sends msgGWinXList to process the xlist. This extracts the appropriate information from the xlist (via XList2Gesture). GWin then performs the gesture processing and forwarding described below:

- Self send msgGWinGesture.
- If msgGWinGesture returns stsRequestDenied, gWin self sends msgGWinBadGesture.
- If msgGWinGesture returns stsRequestForward and style.gestureForward is set, gWin self sends msgGWinForwardGesture. Similar to the forwarding of keyboard events, the gWin packages up the gesture information and uses msgWinSend to forward the gesture. This results in parent windows potentially receiving msgGWinForwardedGesture (see msgGWinForwardedGesture).
- If msgGWinForwardGesture returns stsRequestForward and the gesture is the help gesture, gWin calls PenPoint's quick help with hlpQuickHelpNoHelp. This invokes quick help with the "No help available" text.
- If msgGWinForwardGesture returns stsRequestDenied or stsRequestForward, gWin self sends msgGWinBadGesture

Case 2: How a GWin Synthesizes Some Gestures.

If, when processing input events, gWin sees a press-hold or a tap-press-hold input event, gesture processing and forwarding takes place. If the gesture is unrecognized, then normal input processing continues. This means that if an end-user press-holds on an area where press-hold has no meaning, the window in question receives msgGWinGesture with xgsPressHold. The window returns stsRequestForward (as will all the windows that see msgGWinForwardGesture). Normal processing continues, and when the user lifts the pen the translation of the single tap occurs and the gesture processing mentioned above takes place. If the gesture is recognized, the gesture translation is aborted and input data is thrown away until (and including) the next Pen Up event. A description:

- gWin self sends msgGWinGesture with xgsPressHold or xgsTapHold.
- If the response to msgGWinGesture is stsRequestDenied, processing of the input continues.
- If the response to msgGWinGesture is stsOK, gesture processing is aborted.
- If the response to msgGWinGesture is stsRequestForward and style.gestureForward is set, gWin self sends msgGWinForwardGesture. This results in parent windows potentially receiving msgGWinForwardedGesture (see msgGWinForwardedGesture).
- If the response to msgGWinForwardGesture is stsRequestDenied or stsRequestForward, processing of the input continues.
- If the response to msgGWinForwardGesture is stsOK, gesture processing is aborted.

Case 3: How a GWin Responds to msgQuickHelpHelpShow.

The final case in which msgGWinGesture is sent is in response to msgQuickHelpHelpShow. This is sent from theQuickHelpManager when in help mode and the user taps on the screen. GWin responds by sending msgGWinGesture with the help gesture, and performing similar forwarding above. When msgGWinGesture returns stsRequestDenied, or msgGWinForwardGesture returns stsRequestDenied or stsRequestForward, gWin sends msgQuickHelpShow to display the No Help Available message.
**Style.\textit{gestureLocal} and Coordinate Transformations**

When using large windows (width or height near or above $2^{16}$), you should style.\textit{gestureLocal} to true. Doing so avoids some potential numeric overflow conditions that can make gesture recognition unreliable.

Setting style.\textit{gestureLocal} true changes the coordinate system used internally by gWin. It also changes the coordinate system used in some of gWin's more sophisticated self sent messages. If you don't use these more sophisticated messages, you can just set style.\textit{gestureLocal} true and never worry about it again, regardless of the size of your window. If you do use these messages, then you should read the rest of this section to understand what's different.

Here are the messages whose parameters are affected by style.\textit{gestureLocal}:

- msgGWinStroke
- msgGWinXList
- msgGWinTransformGesture
- msgGWinTransformXList

Normal gesture processing (style.\textit{gestureLocal} is false) is done using the following coordinate transformations:

- The stroke input event is delivered with pArgs->xy set to the local window coordinates and the pen data in root window pen coordinates.
- On the first stroke to the window, gWin remembers an offset of (0,0). This step is obviously trivial in this case but is important when style.\textit{gestureLocal} is true.
- This value is first converted to root window coordinates and then the resulting value is converted to pen units.
- This vector is subtracted from the origin of the pen stroke data. The pen stroke data is still in pen units but has been shifted so that its origin is relative to the local window origin.
- This shifted stroke is self sent using msgGWinStroke. THIS IS IMPORTANT. Any object intercepting this message gets pen data that has been shifted to appear in the local window. This is slightly different than the pen stroke which comes from the input system.
- In response to the first msgGWinStroke, gWin creates a translator and makes itself an observer of the translator. The stroke is then added to the translator.
- Normal input collection of strokes continues. Eventually the gesture is completed and translation occurs.
- In response to msgXlateComplete, gWin gets the XList data and converts it from pen units to window units. Remember that since the pen strokes were shifted by the origin of the window (in digitizer units), the window units give locations in the local window. gWin then self sends msgGWinXList.
- In response to msgGWinXList, gWin converts the xlist information and self sends msgGWinGesture.

If style.\textit{gestureLocal} is true, the same sequence of events occurs, but with the following change in coordinate systems:

- When the first stroke comes in to the gWin, the local window coordinates of the stroke are saved as the offset instead of 0,0. This value is converted to root window coordinates and then converted to
pen units and used to offset the stroke. This means that the stroke is no longer in local window space, but rather are in root window coordinate space.

- Subclasses which handle msgGWinStroke are getting data which is root window relative. If they need it in local window space then they have to transform it first.
- When the translation is complete, the offset that was remembered earlier is converted to root window coordinates and then to pen units. This offset is added to the points returned by the translator before converting back to screen units. The effect is that now the gesture is shifted back to its proper location in root window space.
- When converting the XList data to the GWin gesture, the important points are converted from root window coordinates back to local window coordinates before self sending msgGWinGesture.

```c
#ifndef GWIN_INCLUDED
#define GWIN_INCLUDED
#endif
#ifndef GO_INCLUDED
#include <go.h>
#endif
#ifndef CLSMGR_INCLUDED
#include <clsmgr.h>
#endif
#ifndef WIN_INCLUDED
#include <win.h>
#endif

// Next up: 25 Recycled: 3
```

**Common #defines and typedefs**

typedef OBJECT GWIN;

**Default Window Flags**

These are the default input flags set by a gWin at msgNew time if gestureEnable is set. Changing these flags after new time is possible, but extreme care needs to be taken as these define the pen events that get generated to the window.

```c
#define gWinInputFlags (inputStroke | inputOutProx | inputInk | inputTip | inputTimeout | inputAutoTerm | inputEnter | inputHoldTimeout)
```

**Style Structure**

typedef struct GWIN_STYLE {
    U16 gestureEnable: 1, // enables gesture translation
    gestureForward: 1, // enables forwarding of gestures
    gestureLocal: 1, // enables localized strokes for large gesture windows (>32K digitizer pts)
    keyboardForward: 1, // enables forwarding of key events
    privateData1: 2, // private
    grabDown: 1, // grab input on msgPenDown vs. msgPenStroke
    grabActive: 1, // private
    firstEnter: 1, // grab on msgPenEnter if no other grab
    tossingEvents: 1, // private
    askOtherWin: 1, // ask other gWin if it wants event
    otherWinSaysYes: 1, // answer yes if asked if you want event
    reserved: 4 // reserved for future use
} GWIN_STYLE, *P_GWIN_STYLE;
**Gesture Structure**

This data structure defines all information returned by a gesture translator in the form of a simple data structure. It is used as a parameter to many of the gesture methods defined in `gWin`.

```c
typedef struct GWIN_GESTURE {
    MESSAGE msg; // gesture Id
    RECT32 bounds; // bounding box in LWC
    XY32 hotPoint; // gesture hot point
    OBJECT uid; // object in which the gesture was generated
    U32 reserved; // reserved for future use
} GWIN_GESTURE, *P_GWIN_GESTURE;
```

**Messages**

**msgNew**

Creates and initializes a new instance.

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

**Arguments**

```c
typedef struct GWIN_NEW_ONLY {
    GWIN_STYLE style; // gWin style flags
    U32 helpId; // quick help id
    U32 reserved;
} GWIN_NEW_ONLY, *P_GWIN_NEW_ONLY;
```

```c
#define gWinNewFields

winNewFields
GWIN_NEW_ONLY gWin;
```

```c
typedef struct GWIN_NEW {
    gWinNewFields
} GWIN_NEW, *P_GWIN_NEW;
```

**Comments**

If `gWin.style.gestureEnable` is true, then `gWin` ORs in `gWinInputFlags` into `pArgs->win.flags.input` before passing the message to its ancestors. These `win.flags.input` bits can be changed after the `gWin` is created, but extreme care should be taken!

If setting a `helpId`, setting the `pNew->gWin.helpId` to the same as the `pNew->win.tag` helps minimize memory needed by the object. It is recommended that the `helpId` be the same as the window tag if possible. However, if the window tag changes when the help id is the same as the window tag, then the help tag will change too.

**msgNewDefaults**

Initializes the `GWIN_NEW` structure to default values.

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

**Arguments**

```c
typedef struct GWIN_NEW {
    gWinNewFields
} GWIN_NEW, *P_GWIN_NEW;
```

**Comments**

Zeros out `pNew->gWin` and sets:

```c
pArgs->gWin.style.gestureEnable = TRUE;
pArgs->gWin.style.gestureForward = TRUE;
pArgs->gWin.style.keyboardForward = TRUE;
pArgs->gWin.style.grabDown = TRUE;
win.input = gWinInputFlags;
```
msgGWinGetStyle
Returns the current style.
Takes P_GWIN_STYLE, returns STATUS.

#define msgGWinGetStyle
typedef struct GWIN_STYLE {
  U16 gestureEnable: 1, // enables gesture translation
  gestureForward: 1, // enables forwarding of gestures
  gestureLocal: 1, // enables localized strokes for large
                   // gesture windows (>32K digitizer pts)
  keyboardForward: 1, // enables forwarding of key events
  privateData1: 2, // private
  grabDown: 1, // grab input on msgPenDown vs.
               // msgPenStroke
  grabActive: 1, // private
  firstEnter: 1, // grab on msgPenEnter if no other grab
  tossingEvents: 1, // private
  askOtherWin: 1, // ask other gWin if it wants event
  otherWinSaysYes: 1, // answer yes if asked if you want event
  reserved: 4; // reserved for future use
} GWIN_STYLE, *P_GWIN_STYLE;

msgGWinSetStyle
Sets the style settings.
Takes P_GWIN_STYLE, returns STATUS.

#define msgGWinSetStyle
typedef struct GWIN_STYLE {
  U16 gestureEnable: 1, // enables gesture translation
  gestureForward: 1, // enables forwarding of gestures
  gestureLocal: 1, // enables localized strokes for large
                   // gesture windows (>32K digitizer pts)
  keyboardForward: 1, // enables forwarding of key events
  privateData1: 2, // private
  grabDown: 1, // grab input on msgPenDown vs.
               // msgPenStroke
  grabActive: 1, // private
  firstEnter: 1, // grab on msgPenEnter if no other grab
  tossingEvents: 1, // private
  askOtherWin: 1, // ask other gWin if it wants event
  otherWinSaysYes: 1, // answer yes if asked if you want event
  reserved: 4; // reserved for future use
} GWIN_STYLE, *P_GWIN_STYLE;

Comments
If gestureEnable is true, gWin ORs in the gWinInputFlags with the window flags. (See the comments
near msgNew in this file.) Setting gestureEnable to false does NOT clear these flags.

msgGWinSetHelpId
Sets the gWin's helpId for quick help.
Takes U32, returns STATUS.

#define msgGWinSetHelpId
MakeMsg(clsGWin, 16)

Comments
Setting the helpId to be identical to the gWin's win.tag helps minimize the amount of instance data
taken by a gWin.
**msgGWinGetHelpId**

Returns the gWin's helpId.

Takes P_U32, returns STATUS.

```c
#define msgGWinGetHelpId MakeMsg(clsGWin, 17)
```

**msgGWinGetTranslator**

Returns the gWin's translator object.

Takes P_OBJECT, returns STATUS.

```c
#define msgGWinGetTranslator MakeMsg(clsGWin, 7)
```

Comments

gWin's default response is to return the current translator object.

By default, gWin has a null current translator unless strokes have been added since msgNew or since the last msgGWinAbort or msgGWinComplete. (In other words, gWin does not have a translator unless it is currently collecting or translating strokes.)

See Also

msgGWinAbort

**msgGWinAbort**

**msgGWinSetTranslator**

Sets the translator object and returns the previous one.

Takes P_OBJECT, returns STATUS.

```c
#define msgGWinSetTranslator MakeMsg(clsGWin, 8)
```

Comments

This message has no affect if the gWin has not received a stroke from the input system since the last msgGWinComplete or msgGWinAbort.

Because of this limitation you probably should not use this message.

gWin's default response is to set its translator object to pArgs AND to set *pArgs to the uid of the previous translator.

See Also

msgGWinAbort

**msgGWinTransformGesture**

Transforms gesture information into local window coordinates.

Takes P_GWINGesture, returns STATUS.

```c
#define msgGWinTransformGesture MakeMsg(clsGWin, 14)
```

Comments

This message is useful for clients who handle msgGWinForwardedGesture.

Transforms the gesture bounds and hotPoint into the local window coordinate system.

This is only necessary if the gesture occurred in a window other then self.

gWin's default response modifies the bounds, hotPoint, and uid (set to self) fields.
**msgGWinTransformXList**

Transforms xlist information to local window coordinates.

Takes P_XLIST, returns STATUS.

```c
#define msgGWinTransformXList MakeMsg(clsGWin, 15)
```

Comments

This message is useful for clients who handle msgGWinXList.

This message is only necessary if the xlist was generated relative to a window other than self. This message transforms the gesture bounds and hotPoint to local window coordinates system.

---

**Gesture Processing**

**msgGWinStroke**

Self sent to process a pen stroke received from the input system.

Takes P_INPUT_EVENT, returns STATUS.

```c
#define msgGWinStroke MakeMsg(clsGWin, 5)
```

Comments

If style.gestureEnable is false, gWin's default response is to return stsOK.

If style.gestureEnable is true, gWin's default response is as follows. First, if the gWin has no translator, one is created by self sending msgGWinTranslator and gWin makes itself an observer of the translator. Next, it then sends msgScrAddedStroke to the translator to tell the translator that the gWin has received a new stroke.

Subclasses can handle this message and process individual strokes. If style.gestureLocal is false, stroke coordinates are self relative; if style.gestureLocal is true, stroke coordinates are root window relative.

**See Also**

msgGWinTranslator

**msgGWinTranslator**

Self sent to retrieve the translator used to gather and translate strokes.

Takes P_OBJECT, returns STATUS.

```c
#define msgGWinTranslator MakeMsg(clsGWin, 4)
```

Comments

gWin's default response is to create an instance of clsXGesture.

gWin self sends msgGWinTranslator whenever it needs a translator to gather and translate strokes. For instance, when gWin receives msgGWinStroke, and the stroke is the first stroke in a new gesture, gWin self sends msgGWinTranslator.

The translator will be destroyed during gWin's handling of msgGWinComplete or msgGWinAbort.

**See Also**

msgGWinComplete

**msgGWinComplete**

Self sent to complete a gesture.

Takes void, returns STATUS.

```c
#define msgGWinComplete MakeMsg(clsGWin, 6)
```

Comments

gWin self sends msgGWinComplete when "out of proximity" or a timeout occurs. Clients can send msgGWinComplete to cause gesture completion and translation.
gWin's default response to is cause translation as described in the introductory material at the beginning of this file. gWin then destroys its translator.

If the gWin has a grab (perhaps because it was collecting strokes when a client sends this message), the grab is NOT terminated in response to this message. But the gWin will remember that this message has been received and will terminate the grab in response to the next msgInputEvent it receives.

**msgGWinAbort**

Aborts a gesture.

Takes void, returns STATUS.

#define msgGWinAbort MakeMsg(clsGWin, 9)

gWin's default response is very similar to its response to msgGWinComplete, except that the translation is aborted instead completed. As with msgGWinComplete, the gWin destroys its translator and ceases collecting strokes.

A client can send msgGWinAbort to abort the gWin's processing of a gesture.

The grab behavior is identical to that described with msgGWinComplete.

Subclasses may field msgGWinAbort but must also allow their ancestor to see the message.

**msgGWinXList**

Self sent by gWin to process an xlist.

Takes P_XLIST, returns STATUS.

#define msgGWinXList MakeMsg(clsGWin, 1)

After a translation has been completed (in other words, after gWin has received msgXlateCompleted from its translator), gWin extracts the translation data (in the form of an xlist) from the translator, and then self sends msgGWinXList.

gWin's default response is to extract the gesture information from the xlist (using the xlist utility routine XList2Gesture) and then self sends msgGWinGesture.

**See Also**

msgGWinGesture

---

**Gesture Recognition and Forwarding Messages**

**msgGWinGesture**

Self-sent to process a gesture.

Takes P_GWIN_GESTURE, returns STATUS.

#define msgGWinGesture MakeMsg(clsGWin, 2)

typedef struct GWIN_GESTURE {
    MESSAGE msg;             // gesture Id
    RECT32 bounds;           // bounding box in LWC
    XY32 hotPoint;           // gesture hot point
    OBJECT uid;              // object in which the gesture was generated
    U32 reserved;            // reserved for future use
} GWIN_GESTURE, *P_GWIN_GESTURE;
Gesture Recognition and Forwarding Messages

The default response to `msgGWinGesture` is as follows:

For the help gesture(s), return the result of self sending `msgGWinHelp`. By default, `msgGWinHelp` returns `stsRequestForward` if the `helpId` is zero, or `stsOK` if there is a valid `helpId`

For all other gestures, return `stsRequestForward`.

Effectively, the default response of `gWin` to `msgGWinGesture` is to return `stsOK` if the gesture is a help gesture on a window and the window has a valid `helpId`. Otherwise the default behavior is to return `stsRequestForward`.

`GWin`'s default response to `msgGWinForwardedGesture` is the same as `msgGWinGesture`. This means that the help gesture(s) is forwarded up the window hierarchy until a `gWin` has a valid `helpId`, and then that `gWin` sends the appropriate message and quick help id to the `QuickHelpManager`.

Hence a window can have a common `helpId` (and corresponding help text) for all (or some) child windows, and the quick help text displayed will be the same regardless of the child window the gesture actually occurred in.

**Return Value**

- `stsRequestForward` The gesture was not processed and should be forwarded.
- `stsRequestDenied` The gesture was not processed and should not be forwarded.
- `stsOK` The gesture was processed and should not be forwarded.

**See Also**

`msgGWinXList`

---

**msgGWinForwardGesture**

Causes a gesture to be forwarded to parent windows.

Takes `P_GWIN_GESTURE`, returns `STATUS`.

```c
#define msgGWinForwardGesture MakeMsg(clsGWin, 20)
```

The default response of the `gWin` to `msgGWinForwardGesture` is as follows:

- **Arguments**
  - `typedef struct GWIN_GESTURE {
      MESSAGE msg;               // gesture Id
      RECT32 bounds;            // bounding box in LWC
      XY32 hotPoint;            // gesture hot point
      OBJECT uid;               // object in which the gesture was generated
      U32 reserved;             // reserved for future use
    } GWIN_GESTURE, *P_GWIN_GESTURE;
  ```

In response to this message, `gWin` initiates gesture forwarding. This results in each parent window within the same process receiving `msgGWinForwardedGesture`, from the immediate parent to the root.

If any window along the path returns `stsOK` from `msgGWinForwardedGesture`, or the window has style `gestureForward` off, `stsOK` is returned.

`gWin` performs this forwarding via `msgWinSend`. The status returned to the sender of `msgGWinForwardedGesture` is the status returned by this `msgWinSend`. See the comments for `msgGWinForwardedGesture` for return values and their interpretation.

The `msgWinSend` of `msgGWinForwardedGesture` is only delivered to windows in same process.

**Return Value**

- `stsRequestForward` The gesture was not processed by any of the ancestor windows. Further processing should occur if possible.
- `stsRequestDenied` The gesture was not processed by any of the ancestor windows, and was aborted at some level of the walk. No further processing should occur.
- `stsOK` The gesture was processed. No further processing should occur.

**See Also**

`msgGWinXList`
**msgGWinForwardedGesture**

Message received when a gesture is forwarded.

Takes P_GWIN_GESTURE, returns STATUS.

```c
#define msgGWinForwardedGesture MakeMsg(clsGWin, 11)
```

**Message Arguments**

```c
typedef struct GWIN_GESTURE {
    MESSAGE msg; // gesture Id
    RECT32 bounds; // bounding box in LWC
    XY32 hotPoint; // gesture hot point
    OBJECT uid; // object in which the gesture was generated
    U32 reserved; // reserved for future use
} GWIN_GESTURE, *P_GWIN_GESTURE;
```

**Comments**

See the comments describing msgGWinGesture.

msgGWinForwardedGesture is sent to a gWin when a gesture event has been forwarded from a child window. Subclasses wishing to process gestures forwarded from child windows should handle this message.

Do not send this message; it should only be self sent by clsGWin.

**Return Value**

- `stsRequestForward` The gesture was not processed and should be forwarded further.
- `stsRequestDenied` The gesture was not processed and should not be forwarded any further.
- `stsOK` The gesture was processed and should not be forwarded any further.

**See Also**

msgGWinHelp

---

**msgGWinBadGesture**

Displays feedback for unrecognized and unknown gestures.

Takes P_GWIN_GESTURE, returns STATUS.

```c
#define msgGWinBadGesture MakeMsg(clsGWin, 10)
```

**Message Arguments**

```c
typedef struct GWIN_GESTURE {
    MESSAGE msg; // gesture Id
    RECT32 bounds; // bounding box in LWC
    XY32 hotPoint; // gesture hot point
    OBJECT uid; // object in which the gesture was generated
    U32 reserved; // reserved for future use
} GWIN_GESTURE, *P_GWIN_GESTURE;
```

**Comments**

gWin’s response is to display the unrecognized gesture feedback (if `pArgs->msg == xgsNull`) or the unknown gesture feedback (for any other value of `pArgs->msg`).

gWin’s default response to msgGWinXList includes self-sending msgGWinBadGesture if the gesture is unrecognized by the recognition system (xgsNull) or if none of the recipients of msgGWinGesture and msgGWinForwardedGesture processed the gesture.

**See Also**

msgGWinXList

---

**msgGWinHelp**

The gWin displays quick help for itself.

Takes NULL, returns STATUS.

```c
#define msgGWinHelp MakeMsg(clsGWin, 22)
```
Keyboard Processing and Forwarding Messages

msgGWinKey
Self sent to process a key input event.
Takes P_INPUT_EVENT, returns STATUS.
#define msgGWinKey MakeMsg(clsGWin, 21)
Comments
As part of its default response to msgInputEvent, gWin self sends msgGWinKey if the input event is a key event.
gWin's default response to msgGWinKey is to return stsRequestForward.
Return Value
stsRequestForward The key event was not processed and should be forwarded further.
stsRequestDenied The key event was not processed and should not be forwarded any further.
stsOK The key event was processed and should not be forwarded.

msgGWinForwardKey
Initiates keyboard event forwarding.
Takes P_INPUT_EVENT, returns STATUS.
#define msgGWinForwardKey MakeMsg(clsGWin, 19)
Comments
Subclasses should not handle this message.
In response this message, gWin initiates keyboard event forwarding. This results in each parent window within the same process receiving msgGWinForwardedKey, from the immediate parent to the root.
If any window along the path returns stsOK from msgGWinForwardedGesture, or the window has style.keyboardForward off, stsOK is returned.
gWin performs this forwarding via msgWinSend. The status returned to the sender of msgGWinForwardedKey is the status returned by this msgWinSend. See the comments for msgGWinForwardedKey for return values and their interpretation.
The msgWinSend of msgGWinForwardedKey is only delivered to windows in same process.
Return Value
stsRequestForward The key event was not processed by any of the ancestor windows, and should be forwarded further if meaningful.
stsRequestDenied The key event was not processed by any of the ancestor windows, and was aborted at some level of the walk. No further processing should occur.
stsOK The key event was processed. No further processing should occur.
See Also
msgWinSend
### msgGWinForwardedKey
Message received when a keyboard event is forwarded to a gWin.
Takes P_INPUT_EVENT, returns STATUS.

```c
#define msgGWinForwardedKey MakeMsg(clsGWin, 18)
```

**Comments**

`msgGWinForwardedKey` is sent to a gWin when a keyboard event has been forwarded from a child window. Subclasses wishing to handle keyboard events forwarded from child windows should handle this message.

gWin’s default response is to return `stsRequestForward`.

Do not send this message; it should only be self sent by clsGWin.

**Return Value**

- **stsRequestForward** The key event was not processed and should be forwarded further.
- **stsRequestDenied** The key event was not processed and should not be forwarded any further.
- **stsOK** The key event was processed and should not be forwarded any further.

### msgGWinBadKey
Self sent to allow a subclass to handle bad keys.
Takes P_INPUT_EVENT, returns STATUS.

```c
#define msgGWinBadKey MakeMsg(clsGWin, 23)
```

**Comments**

gWin’s default response is to return `stsOK`.

gWin self sends `msgGWinBadKey` when (1) `msgGWinKey` returns `stsRequestDenied`, (2) `msgGWinKey` returns `stsRequestForward` and `style.keyboardForward` is not set, or (3) `msgGWinForwardedKey` returns `stsRequestDenied` or `stsRequestForward`.

### msgGWinIsComplete
Called to determine if a gesture was sent while processing input.
Takes P_GWIN_GESTURE, returns STATUS.

```c
#define msgGWinIsComplete MakeMsg(clsGWin, 24)
```

**Message Arguments**

```c
typedef struct GWIN_GESTURE {
    MESSAGE msg;       // gesture Id
    RECT32 bounds;     // bounding box in LWC
    XY32 hotPoint;     // gesture hot point
    OBJECT uid;        // object in which the gesture was generated
    U32 reserved;      // reserved for future use
} GWIN_GESTURE, *P_GWIN_GESTURE;
```

**Comments**

This message is used to determine if the gesture may have been sent other than when processing `msgGWinXList` or `msgQuickHelpHelpShow`. Put simply, this message returns `stsOK` for any gesture other then those sent while processing input where gesture processing may continue. Examples are press-hold and tap-press hold.

**Return Value**

- **stsRequestDenied** The gesture was sent while processing input
- **stsRequestDenied** The gesture was sent from `msgGWinXList` or `msgQuickHelpHelpShow`. 
msgGWinGestureDone

Sent to indicate the end of a gesture.


```c
#define msgGWinGestureDone MakeMsg(clgWin, 25)
```

```c
typedef struct GWIN_GESTURE {
    MESSAGE msg;   // gesture Id
    RECT32 bounds; // bounding box in LWC
    XY32 hotPoint; // gesture hot point
    OBJECT uid;   // object in which the gesture was generated
    U32 reserved; // reserved for future use
} GWIN_GESTURE, *P_GWIN_GESTURE;
```

As part of its default response to `msgGWinXList`, `gWin` self sends `msgGWinGestureDone`.
(`msgGWinXList` is self sent after the forwarding protocol has completed but before `msgQuickHelpShow` or `msgGWinBadGesture` is sent.)

It is intended for use by classes that modify their state in anticipation of receiving `msgGWinGesture` and fail to receive it. (For instance, a subclass could handle `msgGWinGesture` and not pass the message along to its ancestor). Such classes should watch for `msgGWinAbort` and `msgGWinGestureDone`. Either, but not both, could be sent for any one gesture.

Subclasses may field `msgGWinGestureDone` but must also allow their ancestor to see the message.

---

**Messages from Other Classes**

**msgFree**

Defined in `clsmgr.h`.

Takes `OBJ_KEY`, returns `STATUS`.

Comments

In response to `msgFree`, `gWin` removes itself as an observer of its translator and then destroys the translator. In addition, `gWin` frees any memory it has allocated.

**msgSave**

Defined in `clsmgr.h`.

Takes `P_OBJ_SAVE`, returns `STATUS`.

Comments

In response to `msgSave`, `gWin` saves state information. `gWin` files its `helpId` if the `helpId` is different then the window tag.

Note that the `gWin`'s translator is not saved or restored since the translator only exists while the `gWin` is actively collecting strokes.

**msgRestore**

Defined in `clsmgr.h`.

Takes `P_OBJ_RESTORE`, returns `STATUS`.

Comments

In response to `msgRestore`, `gWin` restores state information, including the `helpId`.

Note that the `gWin`'s translator is not saved or restored since the translator only exists while the `gWin` is actively collecting strokes.
**msgWinSend**

Defined in win.h.

Takes P_WIN_SEND, returns STATUS.

Comments

gWin handles msgWinSend if pArgs->msg is msgGWinForwardedGesture or msgGWinForwardedKey. For all other values of pArgs->msg, gWin simply passes the message to its ancestor.

If pArgs->msg is msgGWinForwardedGesture, gWin self sends msgGWinForwardedGesture. If this returns stsRequestForward and the gWin’s style.gestureForward is set, gWin passes the msgWinSend to its ancestor, allowing the forwarding to continue. Otherwise gWin returns the result of the self send of msgGWinForwardedGesture.

If pArgs->msg is msgGWinForwardedKey, gWin self sends msgGWinForwardedKey. The response to this message is handled similarly to the gesture case, except that style.keyboardForward is checked rather than style.gestureForward.

See Also

msgGWinForwardedKey

**msgInputEvent**

Defined in input.h.

Takes P_INPUT_EVENT, returns STATUS.

Comments

This is the main processing message for gWin.

For keyboard events, gWin self sends msgGwinKey, and performs the keyboard processing and forwarding as described earlier.

For pen events, gWin returns stsInputTerminate if gestureEnable is not set. Otherwise, gWin initiates a grab by returning stsInputGrabTerminate on msgPenDown if style.grabDown is set.

On msgPenStroke events gWin self sends msgGWinStroke and continues the grab by returning stsInputGrabTerminate.

On msgPenOutProxUp, msgPenOutProxDown, or msgPenTimeout gWin self sends msgGWinComplete and releases the grab by returning stsInputTerminate.

For other pen events, gWin returns stsInputTerminate or stsInputTerminate if it "grabbing" (has returned stsInputGrabTerminate due to a msgPenDown or msgPenStroke), and not "released-the-grab" (returned stsInputTerminate due to a msgPenOutProxDown, msgPenOutProxUp, or msgPenTimeout).

If gWin receives a msgPenTap, is not "grabbing", and has gestureEnable set, gWin synthesizes a tap gesture by self sending msgGWinXList. Thus, even though if inputStroke events are turned off in the window, gWin can still recognize tap gestures.

Return Value

stsInputGrabTerminate Temporarily grabbing input events

stsInputTerminate Not grabbing input events.

See Also

msgGWinStroke
**msgQuickHelpHelpShow**
Defined in qhelp.h.
Takes P_XY32, returns STATUS.

Comments
The theQuickHelpManager sends msgQuickHelpHelpShow to a gWin to ask the gWin to display the gWin's quick help. (This is the message that theQuickHelpManager sends when the user taps while in quick help mode.)

A gWin's default response is to self send msgGWinGesture; the gesture sent along with this msgGWinGesture is a synthesized help gesture.

If the response to the msgGWinGesture is stsRequestForward, gWin self sends msgGWinForwardGesture. If the response to the msgGWinForwardGesture is stsRequestForward, gWin self sends msgQuickHelpShow to theQuickHelpManager with a helpId of hlpQuickHelpNoHelp. (In response to this, theQuickHelpManager displays the "no help available" text to the user.)

**msgXlateCompleted**
Defined in xlate.h.
Takes nothing, returns STATUS.

Comments
A gWin's gesture translator sends msgXlateCompleted to the gWin when a gesture translation is complete. (The gWin has previously started the translation by sending msgScrComplete to the gesture translator.)

A gWin's default response is to extract the xlist from the translator and self send msgGWinXList.

See Also
msgGWinXList
This file contains definitions for clsHWCustomFrame. clsHWCustomFrame inherits from clsFrame.

This file contains the API definition for clsHWCustomFrame. Instances of clsHWCustomFrame are created by the Settings Notebook when the user taps the "Customize" button on the Installed Handwriting page. The Settings Notebook will pass in the handle of the prototype set to be customized. It is up to clsHWCustomFrame instances to carry out the customization and destroy themselves when finished.

**Debugging Flags**

clsHWCustomFrame uses the Handwriting debug flag set 'Z'. clsHWCustomFrame uses:

40000  Show all internal debugging messages

```c
#ifdef HWCUSTOM_INCLUDED
#define HWCUSTOM_INCLUDED
#endif

#ifdef FRAME_INCLUDED
#include <frame.h>
#endif

#ifdef FS_INCLUDED
#include <fs.h>
#endif
```

**Common #defines and typedefs**

```c
typedef struct HWCUSTOM_NEW_ONLY {
    OBJECT protoSetHandle; // handle of prototype set to customize
} HWCUSTOM_NEW_ONLY;
#define hwCustomNewFields
frameNewFields
    HWCUSTOM_NEWONLY hwcustom;
}
typedef struct HWCUSTOM_NEW {
    hwCustomNewFields
} HWCUSTOM_NEW, *P_HWCUSTOM_NEW;
```

**File System Attributes**

The values for the 32-bit attribute hwCustomAttrCustomizable are:

0: This profile is fully customizable.

1: This engine allows customization, but this profile is the original, generic profile for this engine; users must rename it before they customize it. This will to prevent users from inadvertently overwriting the original copy of the profile on the distribution media. If the attribute is 1, Handwriting Customization will pop up a dialog forcing the user to copy or rename the profile before customization.

2: This profile is not customizable.

Any other value: Same as 0; profile is fully customizable.
If the attribute is missing from the directory, Customization will assume the profile is fully customizable.

Here is the magic incantation to stamp the attribute on a profile:

```
STAMP /G "<profile>" /A 800278 <value>
```

For instance, the GOWrite profile gets stamped like this:

```
STAMP /G "GOWrite" /A 800278 1
```

This will stamp a value of 1 on the GOWrite directory for Admin 316 (clsHWCustomFrame), Index 1.

```c
#define hwCustomAttrCustomizable FSMakeFix32Attr(clsHWCustomFrame, 1)
```

### Messages

#### msgNewDefaults

Initializes the HWCUSTOM_NEW structure to default values. Default values are the same as for clsFrame, with a protoSetHandle of 0.

Takes P_HWCUSTOM_NEW, returns STATUS. Category: class message.

```c
typedef struct HWCUSTOM_NEW {
    hwCustomNewFields
} HWCUSTOM_NEW, *P_HWCUSTOM_NEW;
```

#### msgNew

Creates a handwriting customization frame window, acting on the handwriting prototype set in pArgs->hwcustom.protoSetHandle. If protoSetHandle==0, acts on the CurrentInstalledHWXProtos.

Takes P_HWCUSTOM_NEW, returns STATUS. Category: class message.

```c
typedef struct HWCUSTOM_NEW {
    hwCustomNewFields
} HWCUSTOM_NEW, *P_HWCUSTOM_NEW;
```

### Quick Help Tags

```c
#define hlpHWCustomIcon MakeTag(clsHWCustomFrame, 0)
#define hlpHWCustomNote MakeTag(clsHWCustomFrame, 1)
#define hlpHWCustomAlert MakeTag(clsHWCustomFrame, 2)
#define hlpHWCustomExitNote MakeTag(clsHWCustomFrame, 25)
#define hlpHomeWinICLabel MakeTag(clsHWCustomFrame, 5)
#define hlpHomeWinULabel MakeTag(clsHWCustomFrame, 6)
#define hlpHomeWinNumLabel MakeTag(clsHWCustomFrame, 7)
#define hlpHomeWinSymLabel MakeTag(clsHWCustomFrame, 8)
#define hlpHomeWinSCLabel MakeTag(clsHWCustomFrame, 9)
#define hlpHomeWinExitLabel MakeTag(clsHWCustomFrame, 10)
#define hlpHomeWinNextArrow MakeTag(clsHWCustomFrame, 4)
#define hlpHomeWinStatTitle MakeTag(clsHWCustomFrame, 21)
#define hlpHomeWinStatSets MakeTag(clsHWCustomFrame, 22)
#define hlpHomeWinStatRecRt MakeTag(clsHWCustomFrame, 23)
#define hlpHomeWinStatLearn MakeTag(clsHWCustomFrame, 24)
#define hlpHomeWinStatRecom MakeTag(clsHWCustomFrame, 26)
#define hlpHomeWinInstrs MakeTag(clsHWCustomFrame, 11)
#define hlpHomeWinBlankAreas MakeTag(clsHWCustomFrame, 3)
#define hlp26WinSetTitle MakeTag(clsHWCustomFrame, 12)
#define hlp26WinLearnBtn MakeTag(clsHWCustomFrame, 13)
#define hlp26WinClearBtn MakeTag(clsHWCustomFrame, 14)
#define hlp26WinNextBtn MakeTag(clsHWCustomFrame, 15)
#define hlp26WinDoneBtn MakeTag(clsHWCustomFrame, 16)
#define hlp26WinInstrs MakeTag(clsHWCustomFrame, 18)
#define hlp26WinInputLabel MakeTag(clsHWCustomFrame, 19)
#define hlp26WinInputBox MakeTag(clsHWCustomFrame, 20)
#define hlp26WinBlankAreas MakeTag(clsHWCustomFrame, 17)
```
This file contains definitions for clsHWLetterFrame.

clsHWLetterFrame inherits from clsFrame.

This file contains the API definition for clsHWLetterFrame. Instances of clsHWLetterFrame are created by the Settings Notebook when the user taps the "Practice" button on the Installed Handwriting page. The Settings Notebook will pass in the handle of the prototype set to practice. It is up to clsHWLetterFrame instances to carry out the practice session and destroy themselves when finished.

## Debugging Flags

clsHWLetterFrame uses the Handwriting debug flag set 'Z'. clsHWLetterFrame uses:

- 10000 Show all internal debugging messages

```c
#ifdef HWLETTER_INCLUDED
#define HWLETTER_INCLUDED
#endif
#include <frame.h>
#endif
```

## Common #defines and typedefs

```c
typedef struct HWLETTER_NEW ONLY {
    OBJECT protoSetHandle;   // handle of prototype set to practice
} HWLETTER_NEW ONLY;
#define hwLetterNewFields \
    frameNewFields \ 
    HWLETTER_NEW ONLY hwletter;
typedef struct HWLETTER_NEW {
    hwLetterNewFields
} HWLETTER_NEW, *P_HWLETTER_NEW;
```

## Messages

### msgNewDefaults

Initializes the HWLETTER_NEW structure to default values. Default values are the same as for clsFrame, with a protoSetHandle of 0.

Takes P_HWLETTER_NEW, returns STATUS. Category: class message.

```c
typedef struct HWLETTER_NEW {
    hwLetterNewFields
} HWLETTER_NEW, *P_HWLETTER_NEW;
```
**msgNew**

Creates a handwriting practice frame window, using the handwriting prototype set in `pArgs->hwletter.protoSetHandle`. If `protoSetHandle==0`, uses the current `InstalledHWXProtos`.

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

```c
typedef struct HWLETTER_NEW {
    hwLetterNewFields
} HWLETTER_NEW, *P_HWLETTER_NEW;
```

**Quick Help Tags**

- `#define hlpLetterPractice` MakeTag(clsHWLetterFrame, 1)
- `#define hlpLWInputSPaper` MakeTag(clsHWLetterFrame, 2)
- `#define hlpLWKeyboard` MakeTag(clsHWLetterFrame, 3)
- `#define hlpLWPrevScribble` MakeTag(clsHWLetterFrame, 4)
- `#define hlpLWXlateResult` MakeTag(clsHWLetterFrame, 5)
- `#define hlpLWAnimationWin` MakeTag(clsHWLetterFrame, 6)
This file contains the API definition for clsInput and PenPoint’s input system. clsInput inherits from clsObject. clsInput provides the object-oriented interface to PenPoint’s input system. The functions described in this file are contained in INPUT.LIB.

Introduction

PenPoint’s input system collects events generated by devices such as thePen and theKeyboard. It then distributes those events to other objects in the system. The input system is almost always single-threaded. Usually only one input event is being distributed through the system at any given time. The exception is when using msgInputModalStart and msgInputModalEnd.

Road Map

This file contains general information about PenPoint’s input system and input events. Information specific to pen events is in pen.h. Information specific to key events is in key.h.

Most PenPoint application programs do not need to use the PenPoint input system directly. PenPoint has several classes that manage input for clients. Check these classes to see if they meet your needs. Candidate classes include the following (and their subclasses). (This list is not exhaustive.)

- clsGWIn (gwin.h)
- clsSPaper (spaper.h)
- clsIP (insert.h)
- clsNotePaper (notepapr.h)
- all toolkit classes.

Any client handling input directly (rather than using PenPoint classes which handle input) needs to understand the following:

- How to set up window input flags so that desired input events are received. See the section "Input Flags."
- How to handle msgInputEvent in general, and how to handle the device-specific values for msgInputEvent’s pArgs->devCode. See pen.h and key.h.
- How to return appropriate status values in response to msgInputEvent. See "Return Values From msgInputEvent."

Any client that needs to grab input needs to understand:

- General grabbing information. See the section "Grabs and Grabbers."
• msgInputEvent return values that start a grab and keep a grab going. See "Return Values From msgInputEvent."

Any client that needs to be the input target (and therefore the recipient of keyboard events) needs to understand:
• InputSetTarget()

The other interfaces described in this file are typically used by sophisticated clients.

Overview

This diagram illustrates the flow of events into, through and out of the input system:

```
+-----------+      +-----------+      +-----------+
| Pen       |      | Keyboard  |      | Other Device |
+-----------+      +-----------+      +-----------+
       |             |             |
       +-----------+      +-----------+

+---------------+      +---------------+
| Input Queue   |      | Input Routing |
+---------------+      +---------------+

| Filters      |
+---------------+

<table>
<thead>
<tr>
<th>Grabber</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Pre-specified</th>
</tr>
</thead>
</table>

| XY           |
+---------------+

| Target       |
+---------------+
```

Each of these major pieces is described below:

• Devices such as the pen and keyboard generate low-level input events. (These "devices" are partially implemented in the MIL and partially implemented in PenPoint.) These low-level input events are converted into PenPoint input events and are sent to the Input Queue.

• The input system pulls events off the queue one at a time and decides where to send or "route" the event.

There the "event routing" process starts.

• First the event is run through the list of Filters. Filters have the opportunity to examine each input event. Filters are ordered by their priority. Filters return a status which indicates how processing of the event should continue.

• Next the event is sent to the current grab object, or grabber. (There might not be a current grabber, in which case this step is skipped.) The grabber returns a status which indicates how processing of
the event should continue and whether the grab should continue. The input system maintains a stack of grabbers to support nested modal behavior.

The next step in an event's routing depends in part on the event. (Only one of these alternatives is used.)

* If the event has a pre-specified destination, msgInputEvent is sent to that destination. If the event has a pre-specified destination, it is found in pArgs->listener for msgInputEvent. An event has a pre-specified destination only if the event has been programmatically inserted into the input system.

* If the event has a "valid" XY coordinate (which typically means it was generated by thePen), the event is routed to window objects. The top-most window (farthest from theRootWindow) which encloses the XY coordinate gets the first opportunity to process the event. Each window may terminate processing of the event or allow the input system to send the event to its (the window's) parent window.

* Otherwise the event is sent to the current input target, if the target is non-null. (This is how all keyboard events are routed.)

Filters

Filters are used to implement some types of modal behavior. Typically this modal behavior is relatively long-lived. For instance, PenPoint's Quick Help mode is implemented using filters.

It is extremely rare for PenPoint application programs to directly use or even be aware of filters.

<table>
<thead>
<tr>
<th>Object</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>qhelp win</td>
<td>16</td>
</tr>
<tr>
<td>vkey win</td>
<td>32</td>
</tr>
<tr>
<td>qhelp nb</td>
<td>32</td>
</tr>
<tr>
<td>vkey app</td>
<td>80</td>
</tr>
<tr>
<td>spell (proof)</td>
<td>96</td>
</tr>
<tr>
<td>insertion pad</td>
<td>96 (if modal)</td>
</tr>
<tr>
<td>menu</td>
<td>112</td>
</tr>
<tr>
<td>note</td>
<td>160</td>
</tr>
<tr>
<td>option</td>
<td>160</td>
</tr>
</tbody>
</table>

Grabs and Grabbers

Grabbers are used to implement light-weight modal behavior. These modes are typically pen controlled in that they start and end with some pen event, such as msgPenDown and msgOutProxUp. For instance resize handles are implemented using grabbers.

Many application programs never use grabbers directly but rather use PenPoint classes that use grabbers. As illustrated in the "Overview" section, the current grabber gets input messages after filters but before "normal" event distribution occurs. The grabber can "swallow" the event and stop any further distribution, or the grabber can allow distribution to continue.

There are two ways to start a grab.

* An object that is handling msgInputEvent can return a status value that tells the input system that it wants to be the grab object. See the section "Return Values from msgInputEvent."

* Any object can call InputSetGrab() passing in the object to become the grabber.

A grabber terminates a grab by returning from msgInputEvent a status value that does not have "Grab" turned on, or by setting the current grabber to objNull.
In order to keep the grab "alive," the grabber must always return a status from `msgInputEvent` that implies "Grab." If the input system gets a status returned that does not have "Grab" implied, it terminates the grab.

```c
#ifndef INPUT_INCLUDED
#define INPUT_INCLUDED
#endif

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

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

#ifndef GEO_INCLUDED
#include <geo.h>
#endif

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

## Common #defines and typedefs

### Miscellaneous

The following flags control event distribution to filters.

- `iflSendMyWindowOnly` tells the input system to not bother sending the message to the filter unless the event happened in the filter or in one of the filter's window children or window ancestors. It is strictly a performance enhancement.

```c
typedef U32 FILTER_FLAGS, *P_FILTER_FLAGS;
#define iflWindow flag0 // Private. Internal use only.
#define iflSendMyWindowOnly flag1
```

This is the number of bytes in the `INPUT_EVENT`'s `eventData` field. The data stored in this field depends on the `devCode` field. Handlers of `msgInputEvent` never need to use this value; all handlers will cast `pArgs->eventData` to an appropriate type.

```c
#define inptEDataSize 32 // no of bytes INPUT_EVENT's eventData field
```

## Return Values From msgInputEvent

### Overview

The status returned from `msgInputEvent` tells the input system how to continue processing the event. This section lists the `STATUS` values that recipients of `msgInputEvent` may return. Each of these statuses contains several "values." (Not all possible combinations of these are legal or supported.)

- Whether distribution for the event should continue or be terminated.
- Grab status. Whether to start or continue a grab for the recipient of `msgInputEvent`.
- Ancestor interest. Whether or not the ancestor class was interested in the event.
- Filter skip. For filters only, whether distribution of the event should skip certain filters.

The following table describes the relationship between the legal status codes and the values they "contain." For clarity, the "no" entries are left blank and the "Filter skip" information is not shown.
These status values can be returned by any handler of `msgInputEvent`:

- `stsInputContinue` Distribution of this event should continue.
- `stsInputTerminate` Distribution of this event should terminate.
- `stsInputGrabContinue` Distribution of this event should continue, and the grab should be continued (or started) for the recipient of `msgInputEvent`.
- `stsInputGrabTerminate` Distribution of this event should terminate, and the grab should be continued (or started) for the recipient of `msgInputEvent`.
- `stsInputGrab` Same as `stsInputGrabTerminate`.
- `stsInputIgnored` An ancestor class may return `stsInputIgnored` to inform a subclass that the ancestor was not interested in the event. The input system treats `stsInputIgnored` just like `stsInputContinue`.
- `stsInputGrabIgnored` An ancestor class may return `stsInputGrabIgnored` to inform a subclass that the ancestor was not interested in the event, but that the grab should be continued (or started) for the object the received `msgInputEvent`. The input system treats `stsInputGrabIgnored` just like `stsInputGrabContinue`.

These statuses should only be returned by Filters:

- `stsInputSkip` Distribution of this event should continue but all remaining filters should be skipped.
- `stsInputSkipTo2` Distribution of this event should continue but all remaining filters in Range 1 (priority less than 64) should be skipped.
- `stsInputSkipTo3` Distribution of this event should continue but all remaining filters in Ranges 1 and 2 (priority less than 128) should be skipped.
- `stsInputSkipTo4` Distribution of this event should continue but all remaining filters in Ranges 1, 2 and 3 (priority less than 192) should be skipped.
- `stsInputTerminateRemoveStroke` Distribution of this event should terminate, and any other events corresponding to the current stroke should not be sent at all.
- `stsInputGrabTerminateRemoveStroke` Distribution of this event should terminate, the grab should be continued (or started) for the recipient of `msgInputEvent`, and any other events corresponding to the current stroke should not be sent at all.

```c
#define stsInputContinue
#define stsInputContinue
#define stsInputSkip
#define stsInputSkipTo2
#define stsInputSkipTo3
#define stsInputSkipTo4
#define stsInputTerminate
```

```c
InputMakeSts (0)
InputMakeSts (evSkip)
InputMakeSts (evSkip | (1 << 4))
InputMakeSts (evSkip | (2 << 4))
InputMakeSts (evSkip | (3 << 4))
InputMakeSts (evTerminate)
```
```c
#define stsInputGrabContinue InputMakeSts(evGrab)
#define stsInputGrabTerminate InputMakeSts(evGrab | evTerminate)
#define stsInputGrab InputMakeSts(evGrab | evTerminater)
#define stsInputTerminateRemoveStroke InputMakeSts(evTerminate | evStroke)
#define stsInputGrabTerminateRemoveStroke InputMakeSts(evGrab | evTerminate | evStroke)
#define stsInputIgnored InputMakeSts(evIgnore)
#define stsInputGrabIgnored InputMakeSts(evIgnore)

#define stsInputQueueFull MakeStatus(clsInput, evOther | 2)
```

## Other Statuses

```c
define stsInputQueueFull MakeStatus(clsInput, evOther | 2)
```

## Input Flags

### Overview

Each window has a set of input flags that are stored in the window's win.flags.input field. These flags can be manipulated while handling msgNew and msgNewDefaults. They can also be manipulated with several other window messages; see win.h for more information.

InputSetGrab() and InputFilterAdd use these flags as one of their parameters.

```c
typedef U32 INPUT_FLAGS, *P_INPUT_FLAGSi
```

### "Interest" Flags

PenPoint's input system can generate many messages. Most clients are only interested in a subset of the messages that can be generated. So clients can provide hints to the input system about the input events the client is interested in. This reduces the message traffic and increases performance. For instance, if a client is not interested in pen movement events when the pen is up above the writing surface (but within proximity), the client can clear the inputMoveUp flag.

Typically, a flag enables or disables several input events. For instance, setting the inputTip flag enables both msgPenDown and msgPenUp (see pen.h).

You should treat these flags as a hint to the input system. You should not assume that a specific input event will not arrive because you have not enabled the corresponding bit in the input flags.

This table contains examples of the messages that are enabled by setting various flags. This table is only representative -- it is not complete!

<table>
<thead>
<tr>
<th>input flag</th>
<th>example of message(s) enabled</th>
<th>message defined in</th>
</tr>
</thead>
<tbody>
<tr>
<td>inputTip</td>
<td>msgPenUp, msgPenDown</td>
<td>pen.h</td>
</tr>
<tr>
<td>inputMoveUp</td>
<td>msgPenMoveUp</td>
<td>pen.h</td>
</tr>
<tr>
<td>inputMoveDown</td>
<td>msgPenMoveDown</td>
<td>pen.h</td>
</tr>
<tr>
<td>inputEnter</td>
<td>msgPenEnterUp, msgPenEnterDown</td>
<td>pen.h</td>
</tr>
<tr>
<td>inputExit</td>
<td>msgPenExitUp, msgPenExitDown</td>
<td>pen.h</td>
</tr>
<tr>
<td>inputInProx</td>
<td>msgPenInProxUp</td>
<td>pen.h</td>
</tr>
<tr>
<td>inputOutProx</td>
<td>msgPenOutProxUp</td>
<td>pen.h</td>
</tr>
<tr>
<td>inputStroke</td>
<td>msgPenStroke</td>
<td>pen.h</td>
</tr>
<tr>
<td>inputTap</td>
<td>msgPenTap</td>
<td>pen.h</td>
</tr>
<tr>
<td>inputHoldTimeout</td>
<td>msgPenHoldTimeout</td>
<td>pen.h</td>
</tr>
<tr>
<td>inputChar</td>
<td>msgKeyChar</td>
<td>key.h</td>
</tr>
<tr>
<td>inputMultiChar</td>
<td>msgKeyMulti</td>
<td>key.h</td>
</tr>
<tr>
<td>inputMakeBreak</td>
<td>msgKeyUp, msgKeyDown</td>
<td>key.h</td>
</tr>
</tbody>
</table>
```
Inking Flags

WARNING: Inking and the acetate are subject to major changes in future releases.

Miscellaneous Flags

inputNoBusy If cleared, then the input system automatically turns on PenPoint’s busy clock if the recipient of a message does not return before a certain timeout. If set, this default busy clock behavior is disabled.

inputResolution If set, msgPenMoveUp and msgPenMoveDown messages are sent each time the pen moves one digitizer unit. (In other words, the input system sends a move event for even the smallest detectable amount of movement. If cleared, move events are sent only when the pen has moved at least one display pixel’s size.

inputAutoTerm Should only be set by a grabber. Specifies that all events that the grabber is not interested in should be treated as if the grabber returned stsInputGrabTerminate.

inputGrabTracker Should only be set by a grabber. Specifies that the grabber does not need the input system to perform its normal hit detection. This is strictly a performance enhancement. (The name of this value is an anachronism. Originally trackers were the only grabbers that didn’t need hit detection.)
Event Distribution Flags

Input distribution flags give some additional information to the being sent the input event.

- **evfFilter** object is getting this event because it is a filter;
- **evfGrab** object is getting this event because it is a grabber;
- **evfListener** object is getting because it was specified in the input event listener field;
- **evfTarget** object is getting this event because it is the target;
- **evfXYLeafToRoot** object is getting this event as part of the XY distribution;
- **evfInSelf** event occurred in this window;
- **evfInChild** event occurred in a child of this window;

NOTE: **evfInSelf** and **evfInChild** will become obsolete in future releases.

- **evfGrabTracker** object had input grab tracker flag on.

### Messages

**msgInputEvent**

The **InputManager** uses this message to deliver input events.

Takes P_INPUT_EVENT, returns STATUS.

```c
typedef struct INPUT_EVENT {
    SIZEOF length;          // actual length of pArgs
    INPUT_DIST_FLAGS flags; // distribution information
    MESSAGE devCode;        // input event
    OS_MILLISECONDS timestamp; // time event was queued
    XY32 xy;                // location of event
    OBJECT listener;       // pre-specified destination
    // Normally objNull.
    OBJECT destination;    // originating device
    OBJECT originator;     // event specific data
    U8 eventData[inptEDataSize]; // event specific data
} INPUT_EVENT, *P_INPUT_EVENT;
#define msgInputEvent MakeMsg(clsInput, 0)
```
**Comments**

pArgs->devCode contains the "event" that is being delivered. These events are device-specific. See pen.h for a list of pen events and key.h for a list of key events.

The pArgs for msgInputEvent is best thought of as a union type. pArgs can always be cast to a P_INPUT_EVENT, but the content of pArgs->eventData depends on the value of pArgs->devCode. For some values of pArgs->devCode, the pArgs are actually larger than an INPUT_EVENT structure, so use the pArgs->length field to determine the length of the input event. For example, the msgPenStroke and msgKeyMulti events both have data which extends past the end of the INPUT_EVENT structure.

For events that have a valid XY, pArgs->destination is the top-most window with input enabled (FlagOff(inputDisable, ...)).

The recipient of this message must return one of the status values described in the section "Return Values from msgInputEvent."

---

**Notification Messages**

**msgInputGrabPushed**

Notifies a grabbing object that it is being pushed onto the grabber stack and the pArgs is the new grabber.

Takes OBJECT, returns STATUS.

*define msgInputGrabPushed MsgNoError (MakeMsg(clsInput, 0x83))

**msgInputGrabPopped**

Notifies a grabbing object that is being popped from the grabber stack and becoming the current grabber.

Takes OBJECT, returns STATUS.

*define msgInputGrabPopped MsgNoError (MakeMsg(clsInput, 0x84))

**msgInputTargetDeactivated**

Notifies the input target that some other object is become the input target.

Takes OBJECT, returns STATUS.

*define msgInputTargetDeactivated MsgNoError (MakeMsg(clsInput, 0x85))

**msgInputTargetActivated**

Notifies an object that it is becoming the input target.

Takes OBJECT, returns STATUS.

*define msgInputTargetActivated MsgNoError (MakeMsg(clsInput, 0x86))

---

**Functions**

**InputFilterAdd**

Adds a filter to the filter list.

Returns STATUS.
**InputFilterRemove**

Removes a filter from the filter list.

Returns STATUS.

**Function Prototype**

```c
STATUS EXPORTED InputFilterRemove(
    OBJECT listener    // filter to remove
);
```

**InputEventInsert**

Adds an event to the input event queue.

Returns STATUS.

**Function Prototype**

```c
STATUS EXPORTED InputEventInsert(
    P_INPUT_EVENT pEvent,
    BOOLEAN stamp
);
```

**Comments**

Most clients do not use this message.

If stamp is true, `pEvent->timestamp` is filled in with the current time and the event is added to the end of the queue. Otherwise, `pEvent->timestamp` is not modified and the event is placed at the head of the queue and the

**Return Value**

`stsInputQueueFull` the input system queue is full

**InputSetTarget**

Sets the input target object and flags.

Returns STATUS.

**Function Prototype**

```c
STATUS EXPORTED InputSetTarget(
    OBJECT target,    // new target object
    U32 flags          // new target flags
);
```

**Comments**

Clients use this message to set the input target. The input target is the object that receives `msgInputEvent` for all events that do not have an XY position — in particular, keyboard events.

PenPoint's UI guidelines state that the selection owner and input target should usually be the same object. PenPoint does not enforce this association in any way. See the UI documentation and sel.h for more information.

**See Also**

`msgInputTargetActivated.h`

**InputGetTarget**

Returns the current input target.

Returns OBJECT.

**Function Prototype**

```c
OBJECT EXPORTED InputGetTarget (void);
```

**See Also**

`InputSetTarget`
**InputSetGrab**

Sets the current grabber and its flags.

Returns STATUS.

Function Prototype: `STATUS EXPORTED InputSetGrab(
  OBJECT grabber, // new grabber
  U32 flags       // new grab input flags
);`

Comments:
The previous grabber is pushed onto the grabber stack.
If the flags parameter is 0, then `inputAllRealEventsFlags` is used.
If both parameters are null, the current grabber is removed and the grabber on the top of the grabber stack (if the stack isn't empty) becomes the current grabber.

**InputGetGrab**

Passes back the current grabber and its flags.

Returns void.

Function Prototype: `void EXPORTED InputGetGrab(
  P OBJECT pGrabber, // grabber
  P U32 pFlags      // current grab flags
);`

See Also: `InputSetGrab`

**msgInputModalStart**

Asks the InputManager to start recursive input.

Takes `P_INPUT_MODAL_DATA`, returns STATUS.

Arguments:
```
typedef struct INPUT_MODAL_DATA {
  U32   reserved;
  U32   clientData[2];
} INPUT_MODAL_DATA, *P_INPUT_MODAL_DATA;
```

Comments:
This message is used to implement a system-wide mode. Typical application programs should not send this message.
You must send this message to the input system using `ObjectSendUpdate()`. The sending task is blocked until the recursive task returns. The recursive task can pass data to the first task via `pArgs`.

See Also: `msgInputModalEnd`

**msgInputModalEnd**

Asks the InputManager to terminate recursive input.

Takes `P_INPUT_MODAL_DATA`, returns STATUS.

Arguments:
```
typedef struct INPUT_MODAL_DATA {
  U32   reserved;
  U32   clientData[2];
} INPUT_MODAL_DATA, *P_INPUT_MODAL_DATA;
```
This message terminates a system-wide mode. Typical application programs should not send this message.

This message must be paired with `msgInputModalStart`.

The sender of this message can pass information to the sender of `msgInputModalStart` by filling in `pArgs`.

This message may be sent with `ObjectCall()` or `ObjectSend()`.

**See Also**

`msgInputModalStart`

---

**msgInputActivityTimer**

Asks `theInputManager` to enable or disable the activity timer.

Takes `BOOLEAN`, returns `STATUS`.

```c
#define msgInputActivityTimer MakeMsg(clsInput, 5)
```

**Comments**

The input system maintains an "activity timer." Each time the input system has no events to process, the input system starts this timer. If no events are received before the timer expires, the input system puts PenPoint into Standby mode. This duration is typically several minutes long.

Long running background tasks should first send `msgInputActivityTimer` with `pArgs` of false to tell `theInputManager` to not turn off the machine. When the background operation is complete, the task should send the message again, but this time with a `pArgs` of true.

`theInputManager` keeps a nesting count which allows nested pairs of sends of this message.
This file contains the API definition for clsIP (Insertion Pads).
clsIP inherits from clsCustomLayout.

**Introduction**

 IPs provide a convenient and standard mechanism for getting handwritten input from a user. "IP" is an abbreviation for "Insertion Pad."

 IPs support several different visual styles -- character boxes, ruled lines, or blank writing areas and different optional behaviors. IPs use a translator to recognize handwriting if necessary.

**Typical Uses and Settings**

This section describes the most common uses and settings for the various types of IPs.

**Character Box IPs:**
- Their new.ip.style.displayType is ipsCharBox.
- Character Box IPs are typically used to edit or insert simple strings of text such as a person's name or a document name.

**Ruled Line IPs:**
- Their new.ip.style.displayType is ipsRuledLines.
- Ruled Line IPs are typically used when the handwriting preference is Ruled.
- When the preference is Ruled/Boxed, then the IP's style.ruledToBoxed and style.boxedToRuled fields are used to control the transmogrification between styles. It is the responsibility of the IP user to examine the preferences and determine if these fields should be set.

**Blank IPs:**
- Blank IPs are typically used to collect and display simple scribbles (perhaps a signature).
- Their new.ip.style.displayType is ipsBlank. Their new.ip.style.buttonType is typically ipsNoButton, as they never do translation.
- They do not display ruled lines in the sPaper created by default, nor do they allow scribble editing (see spaper.h).
- They turn off borders when printing, allowing them to be robustly embedded inside a document.

**Quick Start**

A typical IP client does the following:
- The client creates an IP in one of three styles described above.
- The client then adds itself as an observer of the IP and handles msgIPDataAvailable.
The msgIPDataAvailable handler uses msgIPGetXlateString to extract the string and then processes the string in some application specific manner.

The client should also handle either msgIPCancelled or msgFreePending so that the client can free any allocated data when the IP is destroyed.

### IP Components

An IP is constructed from several pieces. Most clients and subclasses don't need to know anything about these details, but advanced clients and subclasses might.

The main writing area of an IP is either a field or an sPaper. An ipsCharBox IP contains a field (an instance of clsField); ipsRuledLines IP's contain sPaper, as do ipsBlank/ipsSignature. IP's which have style.ruledToBoxed or style.boxedToRuled set switch between a field and an sPaper. The IP is an observer of the sPaper or field. The sPaper or field has an associated translator.

If style.buttonType is ipsBottomButtons or ipsTopButtons, then the IP also contains a command bar with three buttons. The IP is the client of all of the buttons in the command bar.

Technically clients and subclasses can modify these components directly, but this is not recommended. If these components are modified directly, extreme care must be taken -- current and future implementations of IP may make assumptions which can be violated by making some types of changes to the components.

### Client and Observers

There are two different paths for objects to receive "notification" messages from an IP.

If an IP's client is non-null, then the IP sends the following messages to the IP's client. If the client is null, then the IP sends the messages to the IP's observers. Self is the value of pArgs for all of these messages.

- msgIPCancelled
- msgIPClear
- msgIPDataAvailable
- msgIPCopied
- msgIPTransmogrified

### IPs and Translators

The sPaper or field component of an IP (whichever exists) has a translator which performs handwriting recognition.

The creator of the IP may specify this translator in two ways:

- A translator object may be passed to msgNew. Do this by setting new.ip.style.xlateType to ipXlateObject and new.ip.xlate.translator to the translator object's uid.
- An (optionally null) translation template may be passed to msgNew. Do this by setting new.ip.style.xlateType to ipXlateTemplate and new.ip.xlate.pTemplate to the address of the template. If the template is non-null, the IP compiles the template. Then the IP creates a translator (of dsXText; see xtext.h). This translator is created with the passed-in template if the template is non-null.
An IP with style.charOnly sets the translator to recognize single characters.
The translation information (the translator object and the digested template) are destroyed when the IP handles msgFree.
See msgIPSetTranslator for additional information.

IP Destruction

As a convenience, an IP will optionally self-destruct after providing its data or if the IP is cancelled. To get this behavior, set the IP's style.freeAfter to true.
The automatic destruction occurs during an IP's default response to the following messages:

- msgIPGetXlateData
- msgIPGetXlateString
- msgIPCancelled

Transmogrification

One of PenPoint's standard handwriting styles is called Ruled and Boxed.
When writing in this style, the following steps are taken: (1) the user writes into a ruled line (sPaper) IP and hits OK. (2) the handwriting is translated. (3) the ruled writing area is replaced by a combed field. (4) the user makes any corrections in the field and presses OK again. (5) the data is made available to the application and (6) either the IP is destroyed or the combed field is replaced with a ruled line sPaper ready for additional input.
The term "Transmogrification" describes the switching of writing area types and the moving of the data from the ruled lines to the field.
This transmogrification can happen in response to several messages, including msgIPClear, msgIPGetXlateData and msgIPXlateCompleted.
During transmogrification, the IP's style.displayType is changed. Also, the unnecessary components are destroyed and new ones created. The translator associated with the sPaper or field (whichever exists) is moved to the newly created sPaper or field (whichever didn't exist).
The ruledToBoxed and boxedToRuled fields in an IP's style determine when transmogrification happens:

ruledToBoxed

- If style.ruledToBoxed is true, then a ipsRuledLines IP transmogrifies into a ipsCharBox IP when translation occurs.
- Clients typically set style.ruledToBoxed to true if the prInputPadStyle preference is RuledAndBoxed.

boxedToRuled

- If style.boxedToRuled is true, then a ipsCharBox IPs transmogrifies into a ipsRuledLines IP when data is retrieved via msgIPGetXlateData or msgIPXlateString.
- Clients typically set style.boxedToRuled to true only if (1) the prInputPadStyle preference is RuledAndBoxed and (2) the IP is to be used multiple times before it is freed.
IPs and Preferences

This section describes the preferences that an IP considers and when it considers them. It also describes the preferences a client might consider when determining an IP's style. (See prefs.h for general information on preferences.)

When handling msgNew, msgIPSetStyle, and when transmogrifying, an IP uses the user's preferred value for Character Box Height, Character Box Width and Line Height. The IP does NOT observe these preferences so changes in their value won't affect an existing IP unless its style changes or the IP is transmogrified.

Clients may want to consider the following preferences when managing an IP and its translator. (A client may want to only check the preference when creating the IP. Alternatively, a client may want to observe the SystemPreferences and respond to changes.) Note that this is only one possibility—many clients will (correctly) chose to ignore the preferences or map from the preferences to IP characteristics differently.

prInputPadStyle:

- If this is prInputPadStyleRuledAndBoxed, the client would set an IP's style.displayType to ipsRuledLines and style.ruledToBoxed to true and possibly style.boxedToRuled to true. This causes an IP to transmogrify between ipsRuledLines and ipsCharBox display types. (See the section "Transmogrification" for details.)
- If this is prInputPadStyleRuled, the client would set an IP's style.displayType to ipsRuledLines and style.ruledToBoxed and style.boxedToRuled to false.
- If this is prInputPadStyleBoxed, the client would set an IP's style.displayType to ipsCharBox and style.boxedToRuled and style.ruledToBoxed to false.

prWritingStyle:

- Clients may want to let this preference affect the translation information they send with msgNew or the translator set with msgIPSetTranslator.

Single Character IPs

clsIP has specific support for single character IPs. Setting style.charOnly to true enables this support. Usually if charOnly is true, then style.buttonType is ipsProxButton, style.takeGrab is true, and the client floats the IP rather than embedding it.

Setting charOnly to true causes the IP to automatically set the number of rows and columns to 1. It also prepares the translator to expect only a single character.

Debugging Flags

IP's debugging flag set is 'h.' Defined flags are:

0001 Show general information about IP operations.
0002 Show information about IP translation operations.
0004 Show information about IP layout and size operations.

#ifndef INSERT_INCLUDED
#define INSERT_INCLUDED
#include <go.h>
#endif
Common #defines and typedefs

typedef OBJECT IP;

Display Types

Use one of these values in an IP's style.displayType. This field defines the type of the IP.

See the section "Typical Uses and Settings" for more information.

#define ipsRuledLines 0 // standard ruled lines; contains sPaper
#define ipsCharBox 1 // character box editing; contains field
#define ipsBlank 3 // signature pad; contains sPaper
#define ipsSignature 3 // same as ipsBlank
#define ipsCharBoxButtons 4 // Obsolete

ipsEditBox // Obsolete

Translator types

Use one of these values in an IP's style.xlateType. This field defines whether new.ip.xlate contains a
template or a translator object. See the section "IPs and Translators" for more information.

#define ipXlateObject 0 // pNew->xlate.translator is a translator
#define ipXlateTemplate 1 // pNew->xlate.pTemplate is an &XTEMPLATE

Space Collapse Rules

Use one of these values in an IP's style.spaceCollapse. For ipsCharBox IPs, this field defines how spaces
are treated in text strings retrieved from an IP via msgIPGetXlateData or msgIPGetXlateString.

ipsSpaceSpace causes multiple spaces at the end of a line to be replaced with a single space.
ipsSpaceNewLine causes an entire line's worth of spaces to be replaced with a single newline character.
ipsSpaceAsIs causes spaces to be returned literally.

#define ipsSpaceAsIs 0 // WYSIWYG
#define ipsSpaceSpace 1 // Multiple spaces at end of line become 1 space
#define ipsSpaceNewLine 2 // Single line of spaces becomes a newline

Button Types

Use one of these values in an IP's style.buttonType. This field defines the type of buttons an IP contains.

ipsNoButton is typically used with displayType of ipsBlank. ipsProxButton is valid only with
ipsRuledLines. This value cause translation to occur on out of proximity events. ipsBottomButtons and
ipsTopButtons create a command bar at the top or bottom containing an OK, Cancel, and Clear Button.

```c
#define ipsNoButton 0  // No button
#define ipsProxButton 3  // Proximity translation for ipsRuledLines
#define ipsBottomButtons 6  // Command bar buttons on bottom
#define ipsTopButtons 7  // Command bar buttons on top
```

**Modality Behavior**

Use one of these values in an IP's style.modal. When style.takeGrab is true, style.modal defines the result of a pen tap outside of the IP. The term takeGrab is somewhat misleading. The IP actually creates a modal filter to handle input.

```c
#define ipsNoMode 0  // Nothing happens on pen tap outside
#define ipsTranslateMode 1  // Translation happens on pen tap outside
#define ipsCancelMode 2  // Cancel happens if pen tap outside
```

**IP Style**

```c
typedef struct IP_STYLE {
    U16 displayType: 3,  // display type
    buttonType: 3,  // button type
    freeAfter: 1,  // See the section "IP Destruction."
    clientReplace: 1,  // Unused
    xlateType: 1,  // See the section "IPS and Translators."
    charOnly: 1,  // See the section "Single Character IPs."
    modal: 2,  // If style.takeGrab is true, describes modal
    // IP's behavior.
    takeGrab: 1,  // Makes the IP modal. Modal behavior is
    // defined by style.modal.
    reserved1: 1,  // Reserved
    takeFocus: 1,  // IP becomes the input target when created.
    delayed: 1,  // For ipsCharBox IPs, turns on the field
    // component's delayed behavior.
    U16 spaceCollapse: 3,  // Rule for collapsing spaces when
    // extracting information from ipsCharBox IP.
    embeddedLook: 1,  // Set to true to look good when embedded;
    // false to look good when floating. Affects
    // an IP's handling of msgNew and msgIPSetStyle.
    reserved2: 1,  // Reserved
    ruledToBoxed: 1,  // See the "Transmogrification" and "IPS and
    // Preferences" sections.
    boxedToRuled: 1,  // See the "Transmogrification" and "IPS and
    // Preferences" sections.
    clientIsThisApp: 1,  // Private
    reserved: 8,  // Reserved
} IP_STYLE, *P_IP_STYLE;
```

**Component Tags**

The components of an IP have the following window tags. See the section "IP Components" for more information.

```c
#define tagIPSPaper MakeTag(clsIP, 1)  // ipsRuledLines and ipsBlank
                      // IP's sPaper
#define tagIPField MakeTag(clsIP, 2)  // ipsCharBox IP's field
#define tagIPButton MakeTag(clsIP, 3)  // "OK" button
#define tagIPButtonClear MakeTag(clsIP, 4)  // "Clear" button
#define tagIPButtonCancel MakeTag(clsIP, 5)  // "Cancel" button
#define tagIPCommandBar MakeTag(clsIP, 6)  // command bar
Quick Help Tags

In most cases an IP component’s window tag and quick help are identical. But `tagIPSSignatureSPaper` is the quick help tag for `ipsBlank` IP’s `sPaper` and `tagIPSSingleChar` is the quick help tag of an IP with style.charOnly true.

```c
#define tagIPSSignatureSPaper MakeTag(clsIP, 7)
#define tagIPSSingleChar MakeTag(clsIP, 8)
```

Messages

```c
 ifndef NO_NEW
 ifndef ipNewFields

 msgNew

 Creates a new IP.
 Takes IP_NEW, returns STATUS.

 //
 // Translation information. Notice that this is a union type. See the
 // section "IPs and Translators" for more information.
 //

typedef union IP_XLATE {
 OBJECT translator;  // xlateType = 0, clsXlate object
 P_UNKNOWN pTemplate;  // xlateType = 1, P_XTEMPLATE
} IP_XLATE, *P_IP_XLATE;

typedef struct IP_NEW_ONLY {
 IP_STYLE style;  // IP style
 IP_XLATE xlate;  // See the section "IPs and Translators."
 // Translation information for the IP.
 U16 lineHeight;  // Unused
 OBJECT client;  // Client for notification messages.
 // See the section "Client and Observers."
 P_CHAR pString;  // Initial string for ipsCharBox IP’s field.
 U8 rows,cols;  // Number of rows and cols in IP. Can
 // be zero if shrinkWrap is on.
 U16 lines;  // Unused
 U16 xIndex;  // Unused
 U32 reserved1;
 U16 maxLen;  // Max string length IP can return.
 // 0 means no limit.
 U32 reserved;
} IP_NEW_ONLY, *P_IP_NEW_ONLY;

#define ipNewFields
 customLayoutNewFields
 IP_NEW_ONLY ip:

typedef struct IP_NEW {
 ipNewFields
} IP_NEW, *P_IP_NEW;
#endif // ipNewFields
#endif // NO_NEW
```

Comments

In response to msgNew, clsIP creates the necessary components for the IP. This may include an instance of clsField, clsSPaper, or clsCommandBar. The various components are initialized according to the new.ip.style settings.

The internal sPaper or field requires a translator. If xlateType is ipXlateObject, pNew->ip.xlate.translator is used as the translator object. If xlateType is ipXlateTemplate, then
pNew->ip.xlate.pTemplate is compiled and allocated at msgNew, and freed when the component is destroyed. See the section "IPs and Translators" for more information.

border.style.bottomMargin is always bsMarginMedium.

Finally, based on embeddedLook, msgNew changes the border style of the IP and the border and margin styles of the internal components to make the IP look good when embedded (embeddedLook true) or when floating (embeddedLook false).

Defaults changed if embeddedLook is false:

```c
border.style.borderInk = bsInkGray66;
border.style.leftMargin = bsMarginMedium;
border.style.rightMargin = bsMarginMedium;
border.style.topMargin = bsMarginMedium;
border.style.backgroundInk = bsInkGray33;
border.style.shadow = bsShadowThickGray;
win.flags.style |= wsSaveUnder;
```

Defaults changed if embeddedLook is true:

```c
border.style.borderInk = bsInkGray33;
border.style.leftMargin = bsMarginNone;
border.style.rightMargin = bsMarginNone;
border.style.topMargin = bsMarginNone;
border.style.backgroundInk = bsInkWhite;
border.style.shadow = bsShadowThickWhite;
win.flags.style &= ~wsSaveUnder;
```

---

**msgNewDefaults**

Initializes the IP_NEW structure to default values.

Takes P_IP_NEW, returns STATUS.

Message Arguments

typedef struct IP_NEW {
    ipNewFields
    } IP_NEW, *P_IP_NEW;

Comments

When handling msgNew, certain border.style values are changed depending on the value of ip.embeddedLook. See msgNew for details.

Zeros out pNew->ip and sets:

```c
ip.style.displayType = ipsRuledLines;
ip.style.buttonType = ipsBottomButtons;
ip.style.modal = ipsNoMode;
ip.style.delayed = true;
ip.maxLen = maxU16;

border.style.edge = bsEdgeAll;
border.style.resize = bsResizeCorner;
border.style.drag = bsDragDown;
border.style.top = bsTopDrag;

customLayout.style.limitToRootWin = true;
win.flags.input |=
    (inputTip | inputChar | inputMultiChar | inputAutoTerm | \inputInProx | inputEnter | inputMoveUp | inputMoveDelta);
win.flags.style |= wsSendGeometry;
embeddedWin.style.moveable = false;
embeddedWin.style.copyable = false;
```
### msgIPGetStyle

Passes back the style of the IP.

Takes P_IP_STYLE, returns STATUS.

```c
#define msgIPGetStyle MakeMsg(clsIP, 5)
```

```c
typedef struct IP_STYLE {
    U16 displayType: 3, // display type
    buttonType: 3, // button type
    freeAfter: 1, // See the section "IP Destruction."
    clientReplace: 1, // Unused
    xlateType: 1, // See the section "IPs and Translators."
    charOnly: 1, // See the section "Single Character IPs."
    modal: 2, // If style.takeGrab is true, describes modal
               // IP's behavior.
    takeGrab: 1, // Makes the IP modal. Modal behavior is
                // defined by style.modal.
    reserved1: 1, // Reserved
    takeFocus: 1, // IP becomes the input target when created.
    delayed: 1; // For ipsCharBox IPs, turns on the field
                // component's delayed behavior.
    U16 spaceCollapse: 3, // Rule for collapsing spaces when
                // extracting information from ipsCharBox IP.
    embeddedLook: 1, // Set to true to look good when embedded;
                    // false to look good when floating. Affects
                    // an IP's handling of msgNew and msgIPSetStyle.
    reserved2: 1, // Reserved
    ruledToBoxed: 1, // See the "Transmogrification" and "IPs and
                     // Preferences" sections.
    boxedToRuled: 1, // See the "Transmogrification" and "IPs and
                     // Preferences" sections.
    clientIsThisApp: 1; // Private
    reserved: 8; // Reserved
} IP_STYLE, *P_IP_STYLE;
```

### msgIPSetStyle

Changes the style of the IP.

Takes P_IP_STYLE, returns STATUS.

```c
#define msgIPSetStyle MakeMsg(clsIP, 6)
```

```c
typedef struct IP_STYLE {
    U16 displayType: 3, // display type
    buttonType: 3, // button type
    freeAfter: 1, // See the section "IP Destruction."
    clientReplace: 1, // Unused
    xlateType: 1, // See the section "IPs and Translators."
    charOnly: 1, // See the section "Single Character IPs."
    modal: 2, // If style.takeGrab is true, describes modal
               // IP's behavior.
    takeGrab: 1, // Makes the IP modal. Modal behavior is
                // defined by style.modal.
    reserved1: 1, // Reserved
    takeFocus: 1, // IP becomes the input target when created.
    delayed: 1; // For ipsCharBox IPs, turns on the field
                // component's delayed behavior.
    U16 spaceCollapse: 3, // Rule for collapsing spaces when
                // extracting information from ipsCharBox IP.
    embeddedLook: 1, // Set to true to look good when embedded;
```
Clients use this message to change the style settings of an IP. Also, an IP self sends this message to perform transmogrification.

In response to this message, an IP destroys obsolete components and creates new necessary ones. For example, changing from ipsCharBox to ipsRuledLines destroys the field component and creates an sPaper component.

If an sPaper is being destroyed and a field being created, or vice versa, the IP extracts the translator information from the component about to be destroyed and moves it into the newly created one.

This message dirties the layout the IP.

This method does not change the IP's embeddedLook, xlateType, takeGrab, or takeFocus.

msgIPGetTranslator

Passes back the translator for the IP.

Takes P_OBJECT, returns STATUS.

#define msgIPGetTranslator MakeMsg(clsIP, 7)

Comments

Passes back the translator for the IP, regardless of how it was created. An ipsBlank or ipsRuledLines IP passes back the translator used by the sPaper component. An ipsCharBox IP passes back the translator used by the field component.

See the section "IPs and Translators" for more information.

msgIPSetTranslator

Sets the translator for the IP.

Takes P_OBJECT, returns STATUS.

#define msgIPSetTranslator MakeMsg(clsIP, 20)

Comments

Use this message to set an IP's translator.

In response to this message, a ipsCharBox IP sets its field's translator. Other IPs sets their sPaper's translator. All IPs change their style.xlateType to ipXlateObject.

IMPORTANT: An IP does NOT free the current translation information in response to this message. The client must free this translation information. See the section "IPs and Translators" for more information.

msgIPGetClient

Passes back the IP's client object in *pArgs.

Takes P_OBJECT, returns STATUS.

#define msgIPGetClient MakeMsg(clsIP, 14)
msgIPSetClient
Makes pArgs the IP’s client.
Takes P_OBJECT, returns STATUS.
#define msgIPSetClient MakeMsg(clsIP, 15)

See the section "Client and Observers" for more information.

msgIPGetClient

msgIPSetString
Sets a ipsCharBox IP’s string.
Takes P_CHAR, returns STATUS.
#define msgIPSetString MakeMsg(clsIP, 10)

Use this message to initialize or change the contents of a ipsCharBox IP.
For ipsCharBox IPs, the default response to this message is to set the field component’s string and to re-layout the IP. For other types of IPs, the default response is to return stsOK.
See the section "IP Components" for more information.

msgIPTranslate
Translates scribbles in an IP.
Takes nothing, returns STATUS.
#define msgIPTranslate MakeMsg(clsIP, 3)

When pressed, the "OK" button of an IP’s command bar sends this message to the IP. Also, a client can send this message to cause an IP to translate any scribbles. An IP also self sends this message (1) in response to msgGWinForwardedKey and (2) when a modal IP terminates the mode (style.takeGrab is true, style.modal is ipsTranslateMode, and the pen taps outside of the IP).

The IP’s response to this message is as follows.

- ipsRuledLines and ipsBlank IPs send msgSPaperComplete to the IP’s sPaper component. (The sPaper in turn sends msgSPaperXlateCompleted back to the IP; see the comments on msgSPaperXlateCompleted for IP’s response.)
- ipsCharBox IPs with style.delayed false self send msgIPDataAvailable.
- ipsCharBox IPs with style.delayed true and untranslated scribbles in the field first translate the scribbles and then self send msgIPCopied.
- ipsCharBox IPs with style.delayed true and no untranslated scribbles in the field self send msgIPDataAvailable.

pArgs must be 0.

See Also
msgSPaperXlateCompleted
**msgIPCancelled**

Cancels an IP. Also sent to notify observer/client about the cancel.

Takes OBJECT, returns STATUS.

```c
#define msgIPCancelled MakeMsg(clsIP, 18)
```

**Comments**

When pressed, the "Cancel" button of an IP's command bar sends this message to the IP. A client can also send this message to cause an IP to cancel itself. Also, `msgIPCancelled` is sent to an IP's observers/client to notify them about the cancelling.

`msgIPCancelled` is also self sent if a modal IP has a style.modal value of `ipsCancelMode` and the modal IP is terminated (probably by a pen tap outside the IP).

The IP's response to `msgIPCancelled` is as follows. First the IP clears the component (field or `sPaper`) of any data it contains. Next, if the IP's style.freeAfter is true, the IP extracts itself from the window hierarchy and posts `msgDestroy` to itself. Finally, it sends `msgIPCancelled` to observers/client to inform them of the cancellation.

See the sections "IP Destruction" and "Client and Observers" for additional information.

**See Also**

`msgIPCancelled`

---

**msgIPClear**

Clears an IP's contents. Also sent to notify observers/client about the clearing.

Takes OBJECT, returns STATUS.

```c
#define msgIPClear MakeMsg(clsIP, 23)
```

**Comments**

When pressed, the "Clear" button of an IP's command bar sends this message to the IP. A client can also send this message to cause an IP to clear its contents. Also, `msgIPClear` is sent to an IP's observers/client to notify them about the clearing.

An IP's response to `msgIPClear` is as follows. If the IP has an `sPaper` component (ipsRuledLines or `ipsBlank` IP), then `msgSPaperClear` is sent to the `sPaper`. If the IP has a field component, and style.ruledToBoxed is false, then `msgFieldClear` is sent to the field. If the IP has a field and style.ruledToBoxed is true, then the IP transmogrifies itself to have an `sPaper`. Finally, `msgIPClear` is sent to the IP's observers/client.

See the sections "IP Components," "Client and Observers" and "Transmogrification" for additional information.

**See Also**

`msgIPCancelled` (spaper.h)(field.h)

---

**Observer/Client Messages**

**msgIPCopied**

Notifies observer/client that newly translated data has been copied into a `ipsCharBox` IP's field.

Takes OBJECT, returns STATUS.

```c
#define msgIPCopied MakeMsg(clsIP, 19)
```

**Comments**

See the section "Client and Observers" for additional information.
**msgIPDataAvailable**
Notifies observers/client that the IP has translated data available.
Takes OBJECT, returns STATUS.

```
#define msgIPDataAvailable MakeMsg(clsIP, 16)
```

Comments
Observers/clients can respond to this message by sending msgIPGetXlateData or msgIPGetXlateString to get the translated data.
See the section "Client and Observers" for additional information.
See Also
msgIPTranslate

---

**msgIPTransmogrified**
Notifies observers/client that the IP has been transmogrified.
Takes OBJECT, returns STATUS.

```
#define msgIPTransmogrified MakeMsg(clsIP, 24)
```

Comments
See the sections "Transmogrification" and "Client and Observers" for additional information.
See Also
msgIPTranslate

---

### Data Retrieval Messages

**msgIPGetXlateData**
Passes back translated data in xlist form.
Takes P_IP_XLATE_DATA, returns STATUS.

```
#define msgIPGetXlateData MakeMsg(clsIP, 4)
```

Arguments
typedef struct IP_XLATE_DATA {
  OS_HEAP_ID heap;  // In: heap for xlist allocation.
  P_UNKNOWN pXList;  // Out: pointer to resulting xlist.
} IP_XLATE_DATA, *P_IP_XLATE_DATA;

Comments
The default response to msgIPGetXlateData is as follows.
An xlist is created in pArgs->heap (or osProcessHeapId if pArgs->heap is null.) Then the xlist is filled in as follows.

- An ipsCharBox IP’s xlist contains an xtBounds followed by an xtText element. The IP’s field is cleared (using msgFieldClear; see field.h). (The bounds is artificially constructed.)
- An ipsRuledLines IP’s xlist contains the xlist returned by sending msgSPaperGetXlateData (see spaper.h) to the sPaper component of the IP.
- This message should not be sent to a ipsBlank IP because no translation is ever performed by this type of IP.

If the IP’s style.freeAfter is true, then the IP self destructs; see the section "IP Destruction" for details.
If self is a ipsCharBox IP and style.boxedToRuled is true, then the IP transmogrifies into a ipsRuledLines IP. See the "Transmogrification" section.
If self is a ipsCharBox IP then the space collapse rules defined in style.spaceCollapse are applied to the xtText element in the xlist.
IMPORTANT: The sender of msgIPGetXlateData must free the returned xlist and elements in the xlist. (See xlist.h in general and XListFree() in particular.)

See Also
msgIPTransmogrified.h.h

msgIPGetXlateString
Passes back translated data in string form.
Takes P_IP_STRING, returns STATUS.

typedef struct IP_STRING {
  U16 len;       // In-Out: length of buffer pointed to by pString
  P_CHAR pString;  // In-Out: buffer pointer
} IP_STRING, *P_IP_STRING;
#define msgIPGetXlateString MakeMsg(clsIP, 17)

Comments
In response to this message, an IP passes back its translated contents as a simple string form.
Clients should use this message rather msgIPGetXlateData if a simple string is needed. Clients should use msgIPGetXlateData if the additional information contained in an xlist is needed.

If pArgs->len is maxU16, the IP allocates the necessary string memory from the process heap. The sender of msgIPGetXlateString must free this memory.

The returned pArgs->pString is "clipped" to pArgs->maxLen. The actual number of characters returned is returned in pArgs->len.

Note: The handler of this message first self sends msgIPGetXlateData to get an xlist and then converts the data xlist to a string. See the comments regarding msgIPGetXlateData for information on the IP's self destruction and transmogrification.

See Also
msgIPGetXlateData

Messages from Other Classes

msgFree
Defined in clsmgr.h.
Takes P_OBJ_KEY, returns STATUS.

Comments
The IP sends msgFree to its components. It then frees any translation information passed into msgNew.
See the section "IPs and Translators" for more information.

msgSave
Defined in clsmgr.h.
Takes P_OBJ_SAVE, returns STATUS.

Comments
The IP saves all necessary state and uses the window hierarchy filing mechanism to save any components.
If the IP's client is OSThisApp(), this is remembered. See msgRestore for more information.
msgRestore
Defined in clsmgr.h.
Takes P_OBLRESTORE, returns STATUS.

Comments
clsIP restores self and uses the window hierarchy filing mechanism to restore any components. clsIP then re-establishes the necessary connections between self and each component.
If the IP's client was OSThisApp() when saved, then the IP's client becomes OSThisApp(); otherwise the client becomes to objNull.

See Also
win.h

msgSetOwner
Defined in clsmgr.h.
Takes P_OBJ_OWNER, returns STATUS.

Comments
An IP lets its superclasses respond to this message and then sends msgSetOwner to its components.
See the section "IP Components" for more information.

msgSPaperXlateCompleted
Defined in spaper.h.
Takes OBJECT, returns STATUS.

Comments
Only sophisticated subclasses might want to handle this message. An IP with an sPaper component (ipsRuledLines and ipsBlank) receives this message from the sPaper when the sPaper has completed translation.
If style.ruledToBoxed is false, this message simply self sends msgIPDataAvailable. Otherwise the IP tries to transmogrify itself, using the following steps:
  • The translated string is extracted from the sPaper component.
  • If the string is empty, the IP self sends msgIPDataAvailable and gives up the effort to transmogrify.
  • The IP transmogrifies itself.
In both cases, the sPaper component (if it still exists) is cleared.
See the "Transmogrification" section.

See Also
msgIPTransmute

msgWinStartPage
Defined in win.h.
Takes nothing, returns STATUS.

Comments
Only sophisticated subclasses might want to handle this message. This message informs an IP that it exists on a printer and that printing is about to commence.
If the IP is not ipsBlank, an IP's default response is to return stsOK. Otherwise, the IP turns off all of its own margins and all of the borders and ruling lines for the sPaper component. This causes the IP to print only the scribbles, which is particularly appropriate when an IP has been used to capture and hold a signature.
**msgGWinForwardedKey**

Defined in gwin.h.

Takes P_INPUT_EVENT, returns STATUS.

Comments

Only sophisticated subclasses might want to handle this message. A child window may send this message when the child receives a keyboard input event that it doesn’t want to handle.

If the key’s keyCode is uKeyReturn (see key.h), the IP self sends msgIPTranslate. Otherwise it returns stsRequestForward.

Sent when a component (field) forwards a key. An IP containing a field component that forwards the Return key causes msgIPTranslate to be self sent, as if the "OK" button was pressed.

See Also

msgIPTranslate.h.h

---

**msgInputTargetActivated**

Defined in input.h.

Takes OBJECT, returns STATUS.

Comments

Only sophisticated subclasses might want to handle this message. The input system sends this message to an object to inform an object that it is no the input target.

In response to this message, an IP remembers the previous input target. If the IP is a ipsCharBox IP, it makes the IP’s field the input target.

The IP restores the previous input target as part of its response to msgWinExtracted.

---

**msgTrackProvideMetrics**

Defined in track.h.

Takes P_TRACK_MTRICS, returns STATUS.

Comments

Only sophisticated subclasses might want to handle this message.

clsIP is a descendant of clsBorder. Unless turned off by a subclass, an IP is resizeable by the user. When clsBorder creates a resize object and its associated tracker, it first self sends msgTrackProvideMetrics to allow itself to modify the parameters of the tracker.

In response to this message, an IP does the following:

- If the tracker is not a resizing tracker, the IP simply returns stsOK.
- The IP remembers the original client of the tracker so that the IP can forward tracker-related messages onto that original client. It then makes itself be the client of the tracker.
- If the IP has a command bar (style.buttonType is ipsBottomButtons or ipsTopButtons), then pArgs->style.draw is set to tsDrawCmdBarRect and pArgs->cmdBarH is set appropriately.
- The tracker’s minimum size constraints are adjusted so that the IP can get no smaller than the scribbles that are in the IP’s field or sPaper. This prevents scribbles from being covered.
- The IP makes itself the client of the tracker so that the IP receives msgTrackUpdate and msgTrackDone.

See Also

msgTrackUpdate
**msgTrackUpdate**
Defined in track.h.
Takes P_TRACK_METRICS, returns STATUS.

Comments
Only sophisticated subclasses might want to handle this message.
The default response to this message is to forward the message to the tracker’s original client, as remembered in msgTrackProvideMetrics.

See Also
msgTrackProvideMetrics

**msgTrackDone**
Defined in track.h.
Takes P_TRACK_METRICS, returns STATUS.

Comments
Only sophisticated subclasses might want to handle this message.

*ipsBlank* IPs can be resized to any size. Otherwise the default response to this message is to force the new size of the IP to fit nicely around whole rows and columns (in *ipsCharBox* IPs) or lines (in *ipsRuledLines* IPs). Then the message is forwarded to the tracker’s original client, as remembered in msgTrackProvideMetrics.

See Also
msgTrackProvideMetrics

---

**Obsolete**

```c
#define stsIPNotSupported MakeStatus(clsIP, 1) // Obsolete
#define stsIPBadMode MakeStatus(clsIP, 2) // Obsolete
```
This file contains the API definition for the keyboard driver.

clsKey inherits from clsObject.

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

This file defines the data sent with keyboard event. Keyboard events are generated by both the real keyboard and the virtual keyboard.

**Keyboard Events**

When keyboard devices (physical or virtual) generate input events, the events are delivered via msgInputEvent. The following are the value of pArgs->devCode for msgInputEvent.

- msgKeyDown sent when a key is depressed.
- msgKeyUp sent when a key is released.
- msgKeyChar contains an individual character code and is sent when a key is depressed.
- msgKeyMulti contains multiple character codes that have accumulated since the last msgKeyMulti event was sent. This allows processing of multiple keys without the overhead of a separate message for each key. For all of these values, pArgs->eventData should be cast to P_KEY_DATA. (A msgKeyMulti event contains the same information as several msgKeyChar events.)

**Input Flags**

Keyboard events can be enabled or disabled using input flags. See input.h for more information. The relevant flags for keyboard events are:

<table>
<thead>
<tr>
<th>input flag</th>
<th>enables</th>
</tr>
</thead>
<tbody>
<tr>
<td>inputChar</td>
<td>msgKeyChar</td>
</tr>
<tr>
<td>inputMultiChar</td>
<td>msgKeyMulti</td>
</tr>
<tr>
<td>inputMakeBreak</td>
<td>msgKeyUp and msgKeyDown</td>
</tr>
</tbody>
</table>

Clients should still verify that the devCode is the particular message they are interested in.

**Sample Code**

You can verify that your msgInputEvent handler is handling a keyboard message by checking as follows:

```c
if (ClsNum(pArgs->devCode) == C1sNum(clsKey)) {
    ...
}
```

You should further verify that the devCode is the particular message that you are interested in processing.

Once you've decided that you're looking at a key event, you can cast pArgs->eventData as follows:

```c
P_KEY_DATA pKeyData;
pKeyData = (P_KEY_DATA) (pArgs->eventData);
```
This example shows how you might handle msgInputEvent with a devCode of msgKeyUp, msgKeyDown or msgKeyChar:

```c
for (i=0; i<pKeyData->repeatCount; i++) {
    HandleSingleKey(pKeyData->keyCode, pKeyData->shiftState);
}
```

This example shows how you might handle msgInputEvent with a devCode of msgKeyMulti:

```c
P_KEY.MULTI pMulti = pKeyData->multi;
for (i=0; i<pKeyData->repeatCount; i++) {
    for (j=0; j<pMulti[i].repeatCount; j++) {
        HandleSingleKey(pMulti[i].keyCode, pKeyMulti[i].shiftState);
    }
}
```

```c
#ifndef KEY_INCLUDED
#define KEY_INCLUDED
#endif

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

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

#ifndef OSHEAP_INCLUDED
#include <osheap.h>
#endif

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

### Keyboard Event Messages

- `#define msgKeyUp`  `MakeMsg(clsKey, 0)`
- `#define msgKeyDown`  `MakeMsg(clsKey, 1)`
- `#define msgKeyChar`  `MakeMsg(clsKey, 12)`
- `#define msgKeyMulti`  `MakeMsg(clsKey, 13)`

### Common #defines and typedefs

#### Shift Flags

These are used in the shiftState field of KEY_MULTI and KEY_DATA. They indicate which modifier keys were down when the event was generated.

```c
#define keyScrollLock flag0
#define keyNumLock flag1
#define keyCapsLock flag2
#define keyShift flag3
#define keyCtrl flag4
#define keyAlt flag5
#define keyLeftShift flag6
#define keyRightShift flag7
#define keyLeftCtrl flag8
#define keyRightCtrl flag9
#define keyLeftAlt flag10
#define keyRightAlt flag11
#define keyShiftLock flag12
#define keyCtrlLock flag13
#define keyAltLock flag14
```
Key Codes

Special ASCII characters

```c
#define uKeyBackSpace 0x08
#define uKeyTab 0x09
#define uKeyLineFeed 0x0a
#define uKeyReturn 0x0d
#define uKeyEscape 0x1b
```

Keys with no ASCII values; mapped into the user area of Unicode.

```c
#define uKeyF1 0xf001
#define uKeyF2 0xf002
#define uKeyF3 0xf003
#define uKeyF4 0xf004
#define uKeyF5 0xf005
#define uKeyF6 0xf006
#define uKeyF7 0xf007
#define uKeyF8 0xf008
#define uKeyF9 0xf009
#define uKeyF10 0xf00a
#define uKeyF11 0xf00b
#define uKeyF12 0xf00c
#define uKeyInsert 0xf020
#define uKeyDelete 0xf021
#define uKeyHome 0xf022
#define uKeyEnd 0xf023
#define uKeyPageUp 0xf024
#define uKeyPageDown 0xf025
#define uKeyUp 0xf026
#define uKeyDown 0xf027
#define uKeyLeft 0xf028
#define uKeyRight 0xf029
#define uKeyCenter 0xf02a
#define uKeyPressScreen 0xf02b
#define uKeyPause 0xf02c
#define uKeySysRq 0xf02d
#define uKeyBreak 0xf02e
#define uKeyBackTab 0xf02f
```

msgInputEvent Argument Types

`KEY_MULTI` holds the variable length data for `msgInputEvent` with a `devCode` of `msgKeyMulti`.

```c
typedef struct KEY_MULTI {
    U16 keyCode;   // ASCII value
    U16 scanCode;  // keyboard scan code
    U16 shiftState; // state of the shift, ctrl & alt keys
    U16 repeatCount;  // number of autorepeats to apply
    U8 reserved[4];   // reserved for future use
} KEY_MULTI, *P_KEY_MULTI;
```

`KEY_DATA` is the "true" type of `msgInputEvent`'s `pArgs->eventData` for all keyboard event messages.

If `msgInputEvent`'s `pArgs->devCode` is `msgKeyMulti`, the `keyCode`, `scanCode` and `shiftState` fields of this struct are undefined. Each of these fields is defined in a `KEY_MULTI` struct.

```c
typedef struct KEY_DATA {
    U16 keyCode;   // ASCII key translation
    U16 scanCode;  // keyboard scan code
    U16 shiftState; // state of the shift, ctrl & alt keys
    U16 repeatCount;  // if not msgKeyMulti, the no. of identical
                      // keycodes received. If msgKeyMulti, the
                      // number of KEY_MULTI structs in multi.
    U8 reserved[24];
    KEY_MULTI multi[1];  // if msgKeyMulti, an array of KEY_MULTIs
} KEY_DATA, *P_KEY_DATA;
```
KEYBOARD.H

Interface to the software keyboard class. Keyboards do NOT file.

clsKeyboard inherits from clsKeyCap.

Provides the standard keyboard look and interaction.

clsKeyboard inherits from clsKeyCap and provides keyboard-like
behavior. It directly supports the standard QWERTY keyboard and the PC 101 key keyboard layout
and display. Other forms of keyboards can be generated by overriding the keycap layout table.

The make/break interface is implemented through a call-back procedure. This routine is setup in the
new parameters and is called with the standard keyboard messages: msgKeyMake, msgKeyBreak,
msgKeyChar, and msgKeyMulti.

The scan code mapping table is generally reusable for most keyboard layouts.

WARNING: These API's are not currently in a suitable state for developers.

#ifndef KEYBOARD_INCLUDED
#define KEYBOARD_INCLUDED
#endif

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

#ifndef OSHEAP_INCLUDED
#include <osheap.h>
#endif

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

#ifndef KEY_INCLUDED
#include <key.h>
#endif

#ifndef KEYCAP_INCLUDED
#include <keycap.h>
#endif

#ifndef KEystate_INCLUDED
#include <keystate.h>
#endif

Quick Help Ids

This is the quick help Id for keyboard objects.

#define tagKeyboard MakeTag(clsKeyboard, 1)

Class Messages

msgNewDefaults

Initializes the default new arguments.

Takes P_KEYBOARD_NEW, returns STATUS.
typedef struct KEYBOARD_NEW_ONLY {
    P_U16 pMap; // scan code to key map
    P_KEYSTATE_PROC pProc; // proc for processing events
    P_UNKNOWN pUserData; // user data for the proc
} KEYBOARD_NEW_ONLY, *P_KEYBOARD_NEW_ONLY;
#define keyboardNewFields
    keyCapNewFields
    KEYBOARD_NEW_ONLY keyboard;
typedef struct KEYBOARD_NEW {
    keyboardNewFields
} KEYBOARD_NEW, *P_KEYBOARD_NEW;

The default settings are:
pArgs->keyboard.pMap = PC 101 keyboard mapping
pArgs->keyboard.pProc = pNull;
pArgs->keyboard.pUserData = NULL;

msgNew
Creates a new keyboard object.
Takes P_KEYBOARD_NEW, returns STATUS.

typedef struct KEYBOARD_NEW {
    keyboardNewFields
} KEYBOARD_NEW, *P_KEYBOARD_NEW;

define msgKeyboardReturn MakeMsg(clsKeyboard, 1)
This message is only needed by the virtual keyboard application.

Standard Keyboard Events

msgKeyMake
Self call & notification of make key.
Takes P_KEY_DATA, returns STATUS.

msgKeyBreak
Self call & notification of break key.
Takes P_KEY_DATA, returns STATUS.
msgKeyChar
Self call & notification of character event.
Takes P_KEY_DATA, returns STATUS.

msgKeyMulti
Self call & notification of multi-key event.
Takes P_KEY_DATA, returns STATUS.
KEYCAP.H

Interface for the KeyCap class.

clsKeyCap inherits from clsWin.

Provides an array of keycaps for keyboard emulations.

clsKeyCap inherits from clsWin which provides support for an array
of keyboard "caps" which can deliver a scan code and make/break events. Subclasses are expected to
added the glyph which is displayed on the cap when the key is painted. This is accomplished by
intercepting the self-call msgKeyCapPaintCap.

WARNING: These API's are not currently in a suitable state for developers.

#include <go.h>
#include <osheap.h>
#include <uid.h>
#include <win.h>

#define maxCaps 150

typedef struct KEYCAP_TABLE {
    U16 rows;
    U16 switchés;
    U16 capCodes[maxCaps];
} KEYCAP_TABLE, *P_KEYCAP_TABLE;

Cap Width Descriptors

A data table based mechanism is used to define the array of key caps. Each row is a fixed height
(determined by dividing the window by the number of rows). Each cap can have one of five widths,
small, medium, large, extra large and huge. A small cap is the basic unit of measure, all other cap sizes
are multiples of this size. This size is determined by dividing the window width by the number of switch
units. The cap sizes are: small = 1 unit, medium = 1.5 units, large = 2 units, extra large = 2.5 units, and
huge = 7 units.

#define kcEND   (0x0000)  // end of row marker
#define kcS   (0x1000)   // small cap
#define kcM   (0x2000)   // medium cap
#define kcL   (0x3000)   // large cap
#define kcX   (0x4000)   // extra large cap
#define kcH   (0x5000)   // huge cap

#define kcEND   (0x0000)  // end of row marker
#define kcS   (0x1000)   // small cap
#define kcM   (0x2000)   // medium cap
#define kcL   (0x3000)   // large cap
#define kcX   (0x4000)   // extra large cap
#define kcH   (0x5000)   // huge cap

typedef struct KEYCAP_TABLE {
    U16 rows;
    U16 switchés;
    U16 capCodes[maxCaps];
} KEYCAP_TABLE, *P_KEYCAP_TABLE;
Class Messages

msgNewDefaults
Initializes the new arguments.
Takes P_KEYCAP_NEW, returns STATUS.

Arguments
typedef struct KEYCAP_NEW_ONLY {
    P_KEYCAP_TABLE pTable;  // pointer to the keycap table
} KEYCAP_NEW_ONLY, *P_KEYCAP_NEW_ONLY;
define keyCapNewFields
    OBJECT_NEWONLY object;  
    WIN_NEWONLY win;  
} KEYCAP_NEW_ONLY keyCap;
typedef struct KEYCAP_NEW {
    keyCapNewFields  
} KEYCAP_NEW, *P_KEYCAP_NEW;

Comments
The pTable field is initialized to pNull by default.

msgNew
Creates a new instance of the keycap object.
Takes P_KEYCAP_NEW, returns STATUS.

Message
Arguments
typedef struct KEYCAP_NEW {
    keyCapNewFields  
} KEYCAP_NEW, *P_KEYCAP_NEW;

Comments
If the pTable pointer is NULL, the standard QWERTY layout is used by default.

msgKeyCapPaintCap
Hook to allow painting on top of keycap.
Takes P_KEYCAP_INFO, returns STATUS.

Arguments
typedef struct KEYCAP_INFO {
    XY32 xy;  // coordinates for search
    U16 scanCode;  // scan code for the keycap
    RECT32 rect;  // location of the keycap rect, LWC
    BOOLEAN hilite;  // TRUE for highlighted display
    OBJECT dc;  // Drawing context
} KEYCAP_INFO, *pKEYCAP_INFO;
define msgKeyCapPaintCap MakeMsg(clsKeyCap, 1)

Comments
This is the self-call hook which allows subclasses the ability to paint the glyph on the keycap. This call is made during the response to msgWinRepaint and is therefore already bracketed by msgWinBeginRepaint, msgWinEndRepaint.

msgKeyCapScan
Locates the keycap under a given x,y.
Takes P_KEYCAP_SCAN, returns STATUS.

Arguments
typedef struct KEYCAP_SCAN {
    XY32 xy;  // coordinates for search
    U16 scanCode;  // Out: scan code of keycap
    RECT32 rect;  // Out: keycap rect in LWC
} KEYCAP_SCAN, *P_KEYCAP_SCAN;
define msgKeyCapScan MakeMsg(clsKeyCap, 2)

Comments
This function returns the keycap which is under the Local Window Coordinates (LWC) xy.
msgKeyCapGetDc
Returns the DC used by the keycap.
Takes P_KEYCAP_GET_DC, returns STATUS.

typedef struct KEYCAP_GET_DC {
    OBJECT dc;  // Out: keycap dc object
} KEYCAP_GET_DC, *P_KEYCAP_GET_DC;
#define msgKeyCapGetDc MakeMsg(clsKeyCap, 3)

msgKeyCapRedisplay
Forces the display to be regenerated.
Takes nothing, returns STATUS.
#define msgKeyCapRedisplay MakeMsg(clsKeyCap, 5)

msgKeyCapHilite
Hilites the key with the given scan code.
Takes P_KEYCAP_HILITE, returns STATUS.

typedef struct KEYCAP_HILITE {
    U16 scan;  // In: scan code to hilite
    BOOLEAN on;  // In: true to display as hilite
} KEYCAP_HILITE, *P_KEYCAP_HILITE;
#define msgKeyCapHilite MakeMsg(clsKeyCap, 6)

Comments
The key cap object tracks which keys (by scan code) are highlighted at any given time.

msgKeyCapMake
Subclass hook to indicate button press of keycap.
Takes P_KEYCAP_INFO, returns STATUS.

#define msgKeyCapMake MakeMsg(clsKeyCap, 0x80)

typedef struct KEYCAP_INFO {
    U16 scanCode;  // scan code for the keycap
    RECT32 rect;  // location of the keycap rect, LWC
    BOOLEAN hilite;  // TRUE for highlighted display
    OBJECT dc;  // Drawing context
} KEYCAP_INFO, *P_KEYCAP_INFO;

Comments
This message is a self-call when the pen touches down on a keycap. Note that only one make/break event pair is generated for each penDown, penUp combination. Sliding off a key onto another is NOT considered a press on the new key.

msgKeyCapBreak
Subclass hook to indicate button release of keycap.
Takes P_KEYCAP_INFO, returns STATUS.

#define msgKeyCapBreak MakeMsg(clsKeyCap, 0x81)

typedef struct KEYCAP_INFO {
    U16 scanCode;  // scan code for the keycap
    RECT32 rect;  // location of the keycap rect, LWC
    BOOLEAN hilite;  // TRUE for highlighted display
    OBJECT dc;  // Drawing context
} KEYCAP_INFO, *P_KEYCAP_INFO;

Comments
This message is a self-call when the pen is lifted up from the previous make event.
KEYSTATE.H

Interface for the hardware independent keyboard code interpreter

WARNING: These API calls are not currently in a state suitable for developer use.

```c
#ifndef KEYSTATE_INCLUDED
#define KEYSTATE_INCLUDED
#ifndef Go_INCLUDED
#include <go.h>
#endif
#ifndef OSHEAP_INCLUDED
#include <osheap.h>
#endif
#ifndef KEY_INCLUDED
#include <key.h>
#endif
#define keyMultiMax 16 // max # multi-key buffered events
typedef void (PASCAL *P_KEYSTATE_PROC) (P_UNKNOWN, MESSAGE, P_KEY_DATA);
typedef struct KEYSTATE {
    U16 state; // keyboard decode state
    U16 lastScanned; // long-term state flags
    U16 lastSent; // last scan code processed
    U16 count; // last scan code sent
    U16 onHold; // count of repeated codes while on Hold
    S16 multi; // number of character events to be processed
    S16 multiIndex; // number of multi-char events
    P_U16 pMap; // pointer to the scan-to-char map
    S16 multiIndex; // index into the multi-key array
    P_KEY_MULTI pBuffer; // buffer for multi-key recording
    P_KEYSTATE_PROC pKeyEvent; // proc. pointer for reporting keystate changes
    P_UNKNOWN pUserData; // data for use by the user proc.
} KEYSTATE, *P_KEYSTATE;
```

KeyStateSetup

Initializes a state structure to quiescent values.

Returns nothing.

Function Prototype: `void PASCAL KeyStateSetup(
P_KEYSTATE pState
);`

KeyStateProcess

Converts the scan code into the appropriate action for shift keys and standard keys.

Returns nothing.

Function Prototype: `void PASCAL KeyStateProcess(
P_KEYSTATE pState,
U16 scanCode // pointer to the keyboard state structure
);`
**KeyStateConvert**

Converts a scan code to the appropriate character code, or sets up the appropriate shift state.

Returns nothing.

**Function Prototype**

```pascal
void PASCAL KeyStateConvert(
    P_KEYSTATE pState,    // pointer to the keyboard state structure
    U16 scanCode,          // scan code to convert
    P_U16 pChar,           // character code
    P_U16 pDisplay)        // display character code
```

**KeyStateReturn**

Process completion of the key event.

Returns nothing.

**Function Prototype**

```pascal
void PASCAL KeyStateReturn(
    P_KEYSTATE pState,    // pointer to the keyboard state structure
    MESSAGE msg,           // message to return
    P_KEY_DATA pKey)       // key data
```

**KeyStateFindScan**

Returns the scan code for a shift state flag.

Returns nothing.

**Arguments**

```c
typedef struct KEYSTATE_SCANS {
    U16 count;    // In: max count, Out: actual count
    U16 scanCodes[4];    // can be variable number of entries
} KEYSTATE_SCANS, *P_KEYSTATE_SCANS;
```

**Function Prototype**

```pascal
void PASCAL KeyStateFindScan(
    P_KEYSTATE pState,    // pointer to the keyboard state structure
    U16 state,            // state flag for search
    P_KEYSTATE_SCANS pScanCode // Out: scan code
```

**KeyStateDisplay**

Returns the set of display codes for the scan code.

Returns nothing.

**Arguments**

```c
typedef struct KEYSTATE_CODES {
    U16 count;    // In: max count, Out: actual count
    struct {
        U16 shift;
        U16 charCode;
    } data[4];    // can be variable number of entries
} KEYSTATE_CODES, *P_KEYSTATE_CODES;
```

**Function Prototype**

```pascal
void PASCAL KeyStateDisplay(
    P_KEYSTATE pState,    // pointer to the keyboard state structure
    U16 scanCode,          // scan code to be converted
    P_KEYSTATE_CODES pCodes // Out: scan code
```


PEN.H

This file contains the API definition for the pen driver.
c1sPen inherits from c1s0bject.
The functions described in this file are contained in INPUT.LIB.
This file contains information about pen-generated input events. See input.h for general information on
PenPoint's input system and input events. You should probably read at least the "Road Map" section of
input.h before trying to understand this file in detail.

Pen Events

When the pen generates input events, the events are delivered via msgInputEvent. The following values
are the value of pArgs->devCode for msgInputEvent.
msgPenUp sent when the pen tip is lifted from the screen.
msgPenDown sent when the pen tip touches the screen.
msgPenMoveUp sent as the pen moves while above the screen and in proximity.
msgPenMoveDown sent as the pen moves while touching the screen.
msgPenEnterUp sent when the pen enters a window while above the screen and in proximity.
msgPenEnterDown sent when the pen enters into a window while touching the screen.
msgPenExitUp sent when the pen exits a window while above the screen and in proximity.
msgPenExitDown sent when the pen exits a window while touching the screen.
msgPenInProxUp sent when the pen comes into proximity. This message is also sent when certain
timeouts occur; see the section "Proximity Timeout Events" for more information.
msgPenOutProxUp sent when the pen leaves proximity. This message is also sent when certain
timeouts occur; see the section "Proximity Timeout Events" for more information.
msgPenStroke sent with the collected stroke data. See the "Stroke Events" section.
msgPenTap sent when taps are recognized by the driver. See the "Tap Events" section. The taps field of
PEN_DATA contains the number of taps.
msgPenHoldTimeout sent after pen down and hold timeout. See the "Hold Timeout Events" section.
The taps field of PEN_DATA contains the number of taps that occurred before the Hold.

[Terminology Note: the msgPenInProxUp and msgPenOutProxUp events can be thought of as
msgPenInProx and msgPenOutProx since the pen tip is always up when these events are sent. The
trailing "Up" is present for historical reasons only.]
Input Flags

Pen events can be screened out using input flags. See input.h for more information. The relevant flags for pen are:

<table>
<thead>
<tr>
<th>input flag</th>
<th>enables</th>
<th>see section</th>
</tr>
</thead>
<tbody>
<tr>
<td>inputTip</td>
<td>msgPenUp</td>
<td></td>
</tr>
<tr>
<td>inputMoveUp</td>
<td>msgPenMoveUp</td>
<td></td>
</tr>
<tr>
<td>inputMoveDown</td>
<td>msgPenMoveDown</td>
<td></td>
</tr>
<tr>
<td>inputEnter</td>
<td>msgPenEnterUp</td>
<td></td>
</tr>
<tr>
<td>inputExit</td>
<td>msgPenExitUp</td>
<td></td>
</tr>
<tr>
<td>inputInProx</td>
<td>msgPenInProxUp</td>
<td></td>
</tr>
<tr>
<td>inputOutProx</td>
<td>msgPenOutProxUp</td>
<td></td>
</tr>
<tr>
<td>inputStroke</td>
<td>msgPenStroke</td>
<td></td>
</tr>
<tr>
<td>inputTap</td>
<td>msgPenTap</td>
<td></td>
</tr>
<tr>
<td>inputHoldTimeout</td>
<td>msgPenHoldTimeout</td>
<td>&quot;Hold Timeout Events&quot;</td>
</tr>
</tbody>
</table>

Enter Exit Window Events

msgPenEnterUp, msgPenDown, msgPenEnter, and msgPenExitDown are generated when the pen transits a window boundary. The window that the pen was previously in will receive msgPenExit or msgPenEnterDown (if its input flags request them). The window that the pen is now in will receive msgPenEnterUp or msgPenEnterDown (if its input flags request them). Note that if the pen leaves proximity, the window will receive a msgPenOutProxUp and not msgPenExitUp. Similarly, if the pen enters proximity, the window will receive msgPenInProxUp and not msgPenEnterUp.

The timestamp, strokeSeqNum and penXY field of the PEN_DATA structure will be valid. All other fields will be 0.

Hold Timeout Events

msgPenHoldTimeout events are generated when the user puts the pen on the display and leaves it there for the "Hold" timeout duration. This message is also generated if the user taps 1 or more times before holding the pen down.

For example, msgPenHoldTimeout is the event that triggers PenPoint's move and copy, and is also used by some applications to trigger wipe-through area selection.

msgPenHoldTimeout events are sent if the window's input flags have the inputHoldTimeout flag set.

The strokeSeqNum field of the PEN_DATA structure will be the sequence number of the most recent pen down. The penXY field of the PEN_DATA structure will be the pen device coordinates of the first pen down.

will be valid. All other fields will be 0.

Proximity Timeout Events

The input system also has a proximity-related timeout. These are used if the user lifts the pen off the display but leaves the pen in proximity.

This timer is typically used to trigger translation because some users don't lift their pen tips out of proximity but still want translation to occur.
If this timer expires with the pen still in proximity, the input system sends `msgPenOutProxUp`, followed by `msgPenInProxUp`. In other words, the input system generates events to make it look like the user temporarily lifted the pen out of proximity.

[Note: the obsolete messages `msgPenTimeout` and `msgPenHWTimeout` are not sent.]

### Stroke Events

Each time the pen goes down, moves, and comes up, the input system generates `msgInputEvent` with a `pArgs->devCode` of `msgPenStroke`. The `pArgs` also includes a compressed representation of the stroke.

One way to think about a stroke event is as a "summary" or "reprise" of `msgPenDown`, zero or more `msgPenMoveDowns`, and a `msgPenUp`.

Stroke events are delivered to the window in which the stroke started (if that window has the input flag `inputStroke` flag set), even if the stroke crosses that window's boundaries.

Stroke events are generated in addition to the other, lower level, messages that occur as the stroke event is being accumulated. Typical clients either handle `msgPenStroke` or the lower-level messages (`msgPenDown`, `msgPenMoveDown`, `msgPenUp`), but NOT both.

The input system assigns a sequence number to each stroke. Each pen event contains the stroke number that the event is a part of. This number is found in the "strokeSeqNum" field of `PEN_DATA`.

See the "Sample Code" section for an example of how to extract stroke information from the `pArgs` of a stroke event.

### Tap Events

A `msgPenTap` is generated if there is a `msgPenDown` followed by a `msgPenUp` and (1) any pen motion between the Down and Up is below some threshold and (2) the Down and Up happen within a certain time interval and (3) the Down and Up occur over the same window and (4) no other input event (excepting an optional Out of Proximity) event happens within a certain time threshold of the Up.

`msgPenTap` is sent if the input flag `inputTap` is set.

If the pen is "tapped" repeatedly, a single `msgPenTap` is sent and the taps field of `PEN_DATA` contains the number of pen taps.

The `strokeSeqNum` field of the `PEN_DATA` structure will be the sequence number of the most recent pen down. The `penXY` field of the `PEN_DATA` structure will be the pen device coordinates of the first pen down.

### Typical Sequences of Events

This sections illustrates some typical sequences of pen events. It does not include tap, timeout and stroke events. It also does not show forwarding up the window parent chain.

This table contains the flow of events if the pen comes down, moves around, and comes back up, all within a single window.

<table>
<thead>
<tr>
<th>quantity</th>
<th>event</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>msgPenInProxUp</code></td>
</tr>
<tr>
<td>0 or more</td>
<td><code>msgPenMoveUp</code></td>
</tr>
<tr>
<td>1</td>
<td><code>msgPenDown</code></td>
</tr>
<tr>
<td>0 or more</td>
<td><code>msgPenMoveDown</code></td>
</tr>
<tr>
<td>1</td>
<td><code>msgPenUp</code></td>
</tr>
<tr>
<td>0 or more</td>
<td><code>msgPenMoveUp</code></td>
</tr>
<tr>
<td>1</td>
<td><code>msgPenOutProxUp</code></td>
</tr>
</tbody>
</table>
This sequence is complicated if the pen crosses a window boundary while moving. When this happens, the input system generates Enter and Exit events to notify the windows being entered and exited. In the following example, assume that the window hierarchy isn't changing and that the pen crosses a window boundary between windows A and B while the pen is down.

<table>
<thead>
<tr>
<th>quantity</th>
<th>events seen by A</th>
<th>events seen by B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>msgPenInProxUp</td>
<td></td>
</tr>
<tr>
<td>0 or more</td>
<td>msgPenMoveUp</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>msgPenDown</td>
<td></td>
</tr>
<tr>
<td>0 or more</td>
<td>msgPenMoveDown</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>msgPenExitDown</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>msgPenEnterDown</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>msgPenUp</td>
<td></td>
</tr>
<tr>
<td>0 or more</td>
<td>msgPenMoveUp</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>msgPenOutProxUp</td>
<td></td>
</tr>
</tbody>
</table>

If the pen goes down in window A and in response window A "pops up" a window B right where the pen is, and the user lifts the pen, the following sequence occurs:

<table>
<thead>
<tr>
<th>quantity</th>
<th>events seen by A</th>
<th>events seen by B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>msgPenInProxUp</td>
<td></td>
</tr>
<tr>
<td>0 or more</td>
<td>msgPenMoveUp</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>msgPenDown</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>msgPenExitDown</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>msgPenEnterDown</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>msgPenUp</td>
<td></td>
</tr>
<tr>
<td>0 or more</td>
<td>msgPenMoveUp</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>msgPenOutProxUp</td>
<td></td>
</tr>
</tbody>
</table>

**Sample Code**

You can verify that your `msgInputEvent` handler is handling a pen message by checking as follows:

```c
if (ClsNum(pArgs->devCode) == ClsNum(clsPen)) {
    // Code to handle pen events...
}
```

Once you've decided that you're looking at a pen event, you can cast `pArgs->eventData` as follows:

```c
P_PEN_DATA pPenData;
pPenData = (P_PEN_DATA) (pArgs->eventData);
```

If `pArgs->devCode` is `msgPenStroke`, you can get a pointer to the stroke data as follows:

```c
P_PEN_STROKE pStroke;
pStroke = (P_PEN_STROKE)((P_PEN_DATA)(pArgs->eventData))->data;
```

```c
#ifndef PEN_INCLUDED
#define PEN_INCLUDED
#endif

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

#ifndef GEO_INCLUDED
#include <geo.h>
#endif

#ifndef INPUT_INCLUDED
#include <input.h>
#endif
```
Pen Event Messages

```c
#define rnsgPenUp
#define rnsgPenDown
#define rnsgPenMoveUp
#define rnsgPenMoveDown
#define rnsgPenEnterUp
#define rnsgPenEnterDown
#define rnsgPenExitUp
#define rnsgPenExitDown
#define rnsgPenInProxUp
#define rnsgPenOutProxUp
#define rnsgPenStroke
#define rnsgPenTap
#define rnsgPenHoldTimeout
```

MakeMsg(clsPen, eventTipUp)
MakeMsg(clsPen, eventTipDown)
MakeMsg(clsPen, eventMoveUp)
MakeMsg(clsPen, eventMoveDown)
MakeMsg(clsPen, eventEnterUp)
MakeMsg(clsPen, eventEnterDown)
MakeMsg(clsPen, eventExitUp)
MakeMsg(clsPen, eventExitDown)
MakeMsg(clsPen, eventInProxUp)
MakeMsg(clsPen, eventOutProxUp)
MakeMsg(clsPen, eventStroke)
MakeMsg(clsPen, eventTap)

Common #defines and typedefs

All pen events report coordinates in standard pen resolution, which units of 0.1 mm.

```c
#define penStdResolution 10000 // lines per meter
```

Possible states of the pen tip.

```c
typedef U16 PEN_TIP_STATE_FLAGS, *P_PEN_TIP_STATE_FLAGS;
#define penOutOfProximity 0
#define penInProximity flag0
#define penTipDown flag1
```

Possible states of the pen tip.

```c
Enum16(PEN_TIP_STATE_TYPE) {
    penTipOutOfProxState = 0,
    penTipInProxState = 1,
    penTipDownState = 2
};
```

```c
typedef U16 PEN_SUPPORTS_FLAGS, *P_PEN_SUPPORTS_FLAGS;
#define penSupportsProximity flag0
#define penSupportsPressure flag1 // For future use.
#define penSupportsZPosition flag2 // For future use.
#define penSupportsZAngle flag3 // For future use.
#define penSupportsXYAngle flag4 // For future use.
#define penSupportsPenId flag5 // For future use.
#define penSeparateFromScreen flag6 // digitizer is not integrated with display.
#define penDataIsStroke flag7 // For future use.
#define penSupportsInk flag12 // For future use.
#define penSupportsStrokes flag13 // For future use.
```

msgInputEvent Argument Types

PEN_DATA is the "true" type of msgInputEvent's pArgs->eventData for all pen event messages.

timeStamp  time the event occurred, as defined by the pen driver. This may differ from pArgs->timeStamp, which is time the event was enqueued by the input system.

strokeSeqNum  Number of the stroke that this event is in. See the section "Stroke Events" for more information.

taps  if pArgs->devCode is msgPenHoldTimeout this field contains the number of taps that occurred before the hold. If pArgs->devCode is msgPenTap, this field contains the tap count.
data Variable length data. Contents depends on the `msgInputEvent.pArgs->devCode`. For instance, if `pArgs->devCode` is `msgPenStroke`, then this field is the beginning of the event's stroke information.

typedef struct PEN_DATA {
    U32 timestamp;
    P_UNKNOWN reservedPointer;
    U32 strokeSeqNum;
    XY16 penXY; // in 0.1 mm pen units
    PEN_SUPPORTS_FLAGS penSupportsFlags;
    S16 pressure; // For future use.
    S16 zPosition; // For future use.
    U16 penId; // For future use.
    S16 xyAngle; // For future use.
    S16 zAngle; // For future use.
    U16 reserved[1];
    U8 tipState; // one of PEN_TIP_STATE_FLAGS
    U8 taps;
    U8 data[1]; // start of variable length data
} PEN_DATA, *P_PEN_DATA;

`PEN_STROKE` holds the variable length data for `msgInputEvent` with a `devCode` of `msgPenStroke`. See the section "Stroke Events" for more information. It holds the stroke data. It consists of header information followed by the compressed XY data.

The stroke data can be converted into other useful forms using the functions described in the section "Stroke Manipulation Functions."

typedef struct PEN_STROKE {
    RECT16 bounds; // bounds in pen coordinates
    U16 count; // number of points
    U16 id; // stroke id when added to scribble
    struct {
        U16 len:15, // # bytes in the data field
        selected:1; // used by scribble object
    } info;
    U8 data[1]; // byte array of compressed points
} PEN_STROKE, *P_PEN_STROKE;

**Other Types**

typedef struct CURRENT_STD_PEN_DATA {
    S16 xPosition; // in 0.1 mm pen units
    S16 yPosition; // in 0.1 mm pen units
    PEN_TIP_STATE_TYPE penTipState;
    U16 zPosition; // For future use.
    U16 pressure; // For future use.
    U16 penId; // For future use.
    U16 xyAngle; // For future use.
    U16 zAngle; // For future use.
    XY32 positionAcetate;
} CURRENT_STD_PEN_DATA, *P_CURRENT_STD_PEN_DATA;

typedef struct PEN_METRICS {
    U32 minResolution; // lines per meter
    U32 maxResolution; // lines per meter
    U32 currentResolution; // lines per meter
    U32 maxXPosition; // using pen resolution
    U32 maxYPosition; // using pen resolution
    U32 xPosition; // using pen resolution
    U32 yPosition; // using pen resolution
    PEN_TIP_STATE_FLAGS penTipState;
    PEN_SUPPORTS_FLAGS penSupportsFlags;
} PEN_METRICS, *P_PEN_METRICS;
# PEN.H

**Stroke Manipulation Functions**

## Messages

### msgPenMetrics

Sent to thePen. thePen passes back the current pen device metrics.

Takes P_PEN_METRICS, returns STATUS.

```c
#define msgPenMetrics MakeMsg(clsPen, 0xFE)

typedef struct PEN_METRICS {
    U32 minResolution; // lines per meter
    U32 maxResolution; // lines per meter
    U32 currentResolution; // lines per meter
    U32 maxXPosition; // using pen resolution
    U32 maxYPosition; // using pen resolution
    U32 xPos; // using pen resolution
    U32 yPos; // using pen resolution
    U32 deviceFlags;
    U32 reservedU32[2];
    PEN_TIP_STATE_FLAGS penTipState;
    PEN_SUPPORTED_FLAGS penSupportFlags;
    U16 lowSampleRate;
    U16 medSampleRate;
    U16 highSampleRate;
    U16 currentSampleRate;
    U16 reportingThreshold; // using pen resolution
    U16 deviceID;
    U16 reservedU16[4];
} PEN_METRICS, *P_PEN_METRICS;
```

## Stroke Manipulation Functions

### PenExpander

Decompresses a point from the compressed stroke structure.

Returns S16.

**Function Prototype**

```c
S16 PASCAL PenExpander(P_U8 pData, P_S16 pX, P_S16 pY);
```

**Comments**

pX and pY must point to the previous point value. (They must be set to the bounding box origin for the first point). The new point is passed back using the same pointers.

The return value is the number of bytes to advance pData to get to the next point.

### PenStrokeRetrace

Iterates the points in a compressed stroke structure.

Returns S16.

**Function Prototype**

```c
S16 PASCAL PenStrokeRetrace(}
```
PenStrokeUnpack16
Expands a compressed stroke to an array of XY16.
Returns STATUS.

Function Prototype
STATUS PASCAL PenStrokeUnpack16(
  P_PEN_STROKE pStroke,       // compressed stroke
  P_XY32 pBase,               // stroke window offset
  P_XY16 pBuffer,             // point buffer
  BOOLEAN toLWC               // true to transform points to LWC
);

PenStrokeUnpack32
Expands a compressed stroke to an array of XY32.
Returns STATUS.

Function Prototype
STATUS PASCAL PenStrokeUnpack32(
  P_PEN_STROKE pStroke,       // compressed stroke
  P_XY32 pBase,               // stroke window offset
  P_XY32 pBuffer,             // point buffer
  BOOLEAN toLWC               // true to transform points to LWC
);

XY16ToPenStroke
Converts an array of XY16 values to into a PEN_STROKE.
Returns STATUS.

Function Prototype
STATUS EXPORTED XY16ToPenStroke(
  XY16 pPointData[],         // In: XY point data
  U16 numPoints,             // In: number of points in pPointData
  OS_HEAP_ID heapId,         // In: heap to allocate stroke from
  P_PEN_STROKE *ppNewPenStroke  // Out: pointer to new pen stroke
);

Comments
The function allocates memory for the PEN_STROKE from heapId. If pPointData is null or numPoints is 0 then only the PEN_STROKE data structure is allocated.

PenCurrentStandardData
Fills in the most recent pen data in standard units.
Returns nothing.

Function Prototype
void PASCAL PenCurrentStandardData(P_CURRENT_STD_PEN_DATA pPenStdData);


**SCRIBBLE.H**

This file contains the API definition for clsScribble.

clsScribble inherits from clsObject.

Instances of clsScribble (or scribbles for short) store pen strokes. Scribbles also interact with translators.

---

**Introduction**

An scribble is a collection of pen strokes. Clients can add strokes to (and remove strokes from) a scribble. Clients can use msgScrRender to render a scribble in a given drawing context.

A client typically adds strokes to a scribble during the client's response to msgPenStroke type input events.

clsScribble is a relatively low-level piece of PenPoint. Many clients can and should use clsGWin (gwin.h) or clsSPaper (spaper.h) rather than clsScribble.

---

**Coordinates and the Base**

A scribble's coordinates are in Pen Units. (See msgDcUnitsPen in sysgraf.h.)

Each scribble has a "base." By default, a scribble's base is the origin of the first stroke added to the scribble (via msgScrAddStroke). Whenever a stroke is added to a scribble, the scribble's base is subtracted from the origin of the stroke before the stroke is made part of the scribble. In other words, all strokes are stored relative to the scribble's base. This allows repositioning the entire scribble by adjusting the base. For instance, the client might do this in response to a window resize operation to keep the scribble in the "same" position relative to the upper left corner of a window.

This base is not transparent when extracting a stroke from a scribble. When using msgScrStrokePtr to get a pointer to a stroke in a scribble, it is necessary to add the scribble base back to the stroke origin in order to get the original stroke origin.

---

**Adding Strokes to Scribbles**

This example shows how strokes are added to a scribble by a window that is handling msgInputEvent when pArgs->devCode is msgPenStroke. Note that pArgs->base is set to the origin of the scribble.

```c
// msgPenStroke's stroke data arrives in root window-relative device
// units. Convert the stroke data to be self-relative. Steps:
// (1) Compute the origin of self relative to the root window,
// (2) Convert that origin to pen units.
wm.parent = theRootWindow;
wm.child = NULL;
w.bounds.origin.x = 0;
w.bounds.origin.y = 0;
w.bounds.size.h = 0;
w.bounds.size.w = 0;
ObjectCall(msgWinTransformBounds, self, &wm);
ConvertOriginToPenCoordinates(&wm.bounds.origin);
```

Now add the scribble to the stroke. Note that the \texttt{scrAdd.base} is the base (i.e., origin) of the \texttt{STROKE}, not the scribble.

\begin{verbatim}
scrAdd.base.x = pStroke->bounds.origin.x - wm.bounds.origin.x;
scrAdd.base.y = pStroke->bounds.origin.y - wm.bounds.origin.y;
ObjectCall(msgScrAddStroke, scribble, &scrAdd);
\end{verbatim}

This code gives a rough idea of how a scribble adds a stroke in response to \texttt{msgScrAddStroke}. This is provided so that you can see how the base is used. Basically, the base of the scribble is subtracted from \texttt{pArgs->base} and used as the origin of the stroke.

\begin{verbatim}
// Make a local copy of the stroke. Then convert the stroke's origin to be relative to the scribble's base.
pNewStroke = <Copy of pArgs->pStroke>;
pNewStroke->bounds.origin.x = pArgs->base.x - scribble.base.x;
pNewStroke->bounds.origin.y = pArgs->base.y - scribble.base.y;
<add stroke to internal data structures>
\end{verbatim}

\section*{Repositioning Scribbles}

To reposition a scribble, (1) compute the delta by which you want to move the scribble, (2) get the scribble's current base using \texttt{msgScrGetBase}, (3) add the delta to the current base, and (4) set the base using \texttt{msgScrSetBase}. Be sure to use \texttt{msgScrSetBase} since it will readjust the bounds of the scribble.

\section*{Debugging Flags}

\texttt{clsScribble} uses the debugging flag set 'h'. Defined flags are:

\begin{verbatim}
0100  General scribble debugging information
0800  Scribble save and restore debugging information
\end{verbatim}

\section*{Limitations}

Strokes in a scribble must be within \((2^{15}-1)\) units of each other.

Memory for deleted strokes is only recovered upon \texttt{msgScrClear}. No other messages recover deleted stroke memory. Deleted strokes are saved and restored.

\begin{verbatim}
#include <go.h>
#include <osheap.h>
#include <geo.h>
#include <clsmgr.h>
#include <pen.h>
#include <debug.h>
\end{verbatim}

// Next Up: 18 Recycled: 2, 7, 8,
Common #defines and typedefs

typedef OBJECT SCRIBBLE;

stsScrOutOfRange is returned from msgScrAddStroke if the coordinates for the base of the scribble are out of the allowable range for strokes.

#define stsScrOutOfRange MakeStatus(clsScribble,1)

Messages Defined by Other Classes

msgNew
Creates and initializes a new scribble.
Takes P_SCR_NEW, returns STATUS. Category: class message.

msgNewDefaults
Sets the default values for the new arguments.
Takes P_SCR_NEW, returns STATUS.

Arguments
typedef struct SCR_NEW ONLY {
    XY32 base;          // initial base value, default (0,0)
    U32 reserved;       // reserved for future use
} SCR_NEW ONLY;
#define scribbleNewFields \
    OBJECT_NEW ONLY object; \
    SCR_NEW ONLY scribble;
typedef struct SCR_NEW {
    scribbleNewFields
} SCR_NEW, *P_SCR_NEW;

Comments
Zeros out pNew->scribble.

msgFree
Defined in clsmgr.h
Takes P_OBJ_KEY, returns STATUS.

Comments
The scribble frees all of its strokes.

msgSave
Defined in clsmgr.h.
Takes P_OBJ_SAVE, returns STATUS.

Comments
Saves all strokes to pArgs->file.

msgRestore
Defined in clsmgr.h.
Takes P_OBJ_RESTORE, returns STATUS.

Comments
Restores all strokes from pArgs->file.
**msgPicSegPaintObject**

Defined in picseg.h.

Takes P_PIC_SEG_OBJECT, returns STATUS.

Comments

In response to this message, a scribble paints itself. Any object which responds to msgPicSegPaintObject can be placed into a PicSeg (and instance of clsPicSeg).

**Messages**

**msgScrSetBase**

Sets the scribble's base.

Takes P_XY32, returns STATUS.

#define msgScrSetBase MakeMsg(clsScribble, 11)

Comments

Recomputes the bounds of the scribble to reflect the new base.

See the section "Coordinates and the Base" for more information.

**msgScrGetBase**

Passes back the base for the scribble.

Takes P_XY32, returns STATUS.

#define msgScrGetBase MakeMsg(clsScribble, 10)

Comments

See the section "Coordinates and the Base" for more information.

**msgScrGetBounds**

Passes back the bounds of the scribble.

Takes P_RECT32, returns STATUS.

#define msgScrGetBounds MakeMsg(clsScribble, 12)

Comments

Passes back the bounding box that contains all the strokes in the scribble. The bounding box is in pen units.

**msgScrCount**

Passes back the number of strokes in the scribble.

Takes P_U16, returns STATUS.

#define msgScrCount MakeMsg(clsScribble, 1)

**msgScrAddStroke**

Adds a stroke to the scribble.

Takes P_SCR_ADD_STROKE, returns STATUS.

#define msgScrAddStroke MakeMsg(clsScribble, 3)

Arguments

typedef struct SCR_ADD_STROKE {
    XY32       base;  // In: origin of the stroke.
    P_PEN_STROKE  pStroke;  // In: pointer to stroke data
    U16        index;  // Out: index of the newly added stroke
} SCR_ADD_STROKE, * P_SCR_ADD_STROKE;
In response to this message, the scribble makes a copy of the stroke data and adds the stroke to itself. Observers are notified with msgScrAddedStroke. Note the SCR_ADD_STROKE base is the base (i.e., origin) of the STROKE. If this is the first stroke to be added to the scribble, the scribble's base is set to pArgs->base. Otherwise the base of the stroke is shifted by the scribble base as follows:

\[
\text{stroke.bounds.origin} = \text{pArgs->base} - \text{scribble.base};
\]

In response to this message, the scribble marks as deleted the stroke with the index value of pArgs. Observers receive msgScrRemovedStroke.

Note: this does not actually free any memory, the scribble is just marked as deleted.

The scribble uses msgScrHit and msgScrDeleteStroke to do the deletions. Observers receive one msgScrRemovedStroke for each intersecting stroke. If pArgs->window is non-null, the scribble dirties the appropriate rectangle in the window.
msgScrComplete
Sent to a scribble to indicate completion.
Takes void, returns STATUS.

```c
#define msgScrComplete MakeMsg(clsScribble, 5)
```

Comments
Clients send this message to a scribble to tell the scribble that it is "complete." The client is responsible for determining when a scribble is complete. (For instance, the client might decide that a scribble is complete when the client receives an out-of-proximity pen event, or when a certain amount of time has elapsed since the last input event.)

A scribble does nothing in response to this message except to send msgScrCompleted to all observers.

A translator is a typical observer of a scribble. See xlate.h for information about how a translator responds to msgScrCompleted.

See Also
msgScrCompleted

msgScrStrokePtr
Passes back the pointer to the stroke identified by pArgs->index.
Takes P_SCR_STROKE_PTR, returns STATUS.

```c
#define msgScrStrokePtr MakeMsg(clsScribble, 9)
```

Arguments
typedef struct SCR_STROKE_PTR {
   U16 index; // In: index to the stroke
   P_PEN_STROKE pStroke; // Out: pointer to the index’th stroke, or
   // pNull if no such stroke exists.
} SCR_STROKE_PTR, *P_SCR_STROKE_PTR;

Comments
Be Careful! pArgs->pStroke is a pointer to internal data, not a copy.

Strokes retrieved from scribbles are relative to the scribble’s base. The stroke’s origin is NOT the same as was passed to msgScrAddStroke -- you need to add the scribble’s base back. Note that this may still not be the same as the original stroke origin if the scribble base has been adjusted.

msgScrClear
Deletes all the strokes in the scribble.
Takes void, returns STATUS.

```c
#define msgScrClear MakeMsg(clsScribble, 15)
```

Comments
In response to this message, the scribble sets its stroke count to zero. scribble’s stroke count to 0. It also frees all allocated memory.
Important: Observers are not notified!

msgScrRender
Renders a scribble in a window through a DC.
Takes P_SCR_RENDER, returns STATUS.

```c
#define msgScrRender MakeMsg(clsScribble, 13)
```
typedef struct SCR_RENDER {
    U16     start;     // stroke start index (0 for first)
    U16     stop;      // stroke stop index (maxU16 for last)
    XY32    base;      // unused
    OBJECT  dc;       // dc for rendering the stroke
    RECT32  rect;     // dirty rect
} SCR_RENDER, *P_SCR_RENDER;

Draws the strokes in the scribble using pArgs->dc. The start and stop indices describe the inclusive range of strokes to be rendered. Only strokes intersecting pArgs->rect are rendered. pArgs->rect must be in window device coordinates. pArgs->dc must be set up to draw in pen coordinates (using msgDcUnitsPen as described in sysgraf.h).

msgScrHit

Passes back the next stroke which intersects pArgs->rect.

Takes P_SCR_HIT, returns STATUS.

#define msgScrHit MakeMsg(clsScribble, 14)

typedef struct SCR_HIT {
    RECT32 rect;     // In: rect to test against, in pen coordinates.
    U16     index;   // In: For the first send, should be 0. Do not disturb between sends. Out: if
                    // pArgs->hi is true, the index of the next
                    // stroke that intersects pArgs->rect.
    BOOLEAN hit;     // Out: true if another stroke intersect pArgs->rect; false when no more strokes
                    // intersect.
} SCR_HIT, *P_SCR_HIT;

This message is typically sent multiple times to find all strokes that intersect pArgs->rect. The hit-test is quite simple -- a stroke "intersects" if the stroke's bounding box intersects pArgs->rect.

Clients should set pArgs->index to 0 before first sending this message and then not disturb that field between sends.

If a hit is found, pArgs->hit is true and pArgs->index is the stroke index. Otherwise pArgs->hit is false.

Example:

P_SCR_HIT hit;
hit.rect = <rect to check>
hit.index = 0;
hit.hit = TRUE;
while (hit.hit) {
    ObjectCall(msgScrHit, scribble, &hit);
    if (hit.hit) {
        // hit.index now equals the first stroke that intersected
    }
}

Notifications Sent to Observers

msgScrCompleted

Notifies observers that scribble input has been completed.

Takes NULL, returns STATUS.

#define msgScrCompleted MakeMsg(clsScribble, 0x80)
This notification is sent as part of a scribble’s response to `msgScrComplete`.

Typical use: Translators that are observing the scribble may perform their translation in response to this message. (See xlate.h for more information.)

### msgScrAddedStroke

Notifies observers that a stroke has been added to the scribble.

Takes `P_SCR_ADDED_STROKE`, returns `STATUS`.

```c
#define msgScrAddedStroke MakeMsg(clsScribble, 0x81)
```

#### Arguments

- typedef struct SCR_ADDED_STROKE {
  - U16 index; // index of the added stroke
  - P_PEN_STROKE pStroke; // pointer to the stroke data structure
} SCR_ADDED_STROKE, *P_SCR_ADDED_STROKE;

#### Comments

This notification is sent as part of a scribble’s response to `msgScrAddStroke`.

### msgScrRemovedStroke

Notifies observers that a stroke has been removed from the scribble.

Takes `P_SCR_REMOVED_STROKE`, returns `STATUS`.

```c
#define msgScrRemovedStroke MakeMsg(clsScribble, 0x82)
```

#### Arguments

- typedef struct SCR_REMOVED_STROKE {
  - U16 index; // index of the removed stroke
  - U16 id; // stroke identifier
  - RECT32 bounds; // bounds of the removed stroke (in pen units)
} SCR_REMOVED_STROKE, *P_SCR_REMOVED_STROKE;

#### Comments

This notification is sent as part of a scribble’s response to `msgScrDeleteStroke`.
This file contains the API definition for clsSPaper.

clsSPaper inherits from clsView.

An instance of clsSPaper (or sPaper, for short) provides stroke redisplay, very simple stroke editing, and translation.

Road Map

A typical sPaper client simply creates an sPaper with self as the listener. The client than handles the msgSPaperXlateCompleted notification and uses msgSPaperGetXlateData to extract the resulting data.

Clients or subclasses who wish to get involved in the sPaper's management of strokes might use:
- msgSPaperClear
- msgSPaperAddStroke
- msgSPaperDeleteStrokes

Clients or subclasses who wish to be involved in controlling when translation is triggered might use:
- msgSPaperComplete
- msgSPaperAbort

If a client only needs translation, the client may not need to use sPaper at all. The client may be able to use translators (xlate.h) and scribbles (scribble.h) directly.

SPaper Components

An sPaper manages a translator and a scribble. The sPaper observes the the translator and the translator observes the scribble.

The sPaper has a listener which is specified when the sPaper is created. An sPaper makes the listener an observer of the sPaper. As a result, the listener receives sPaper notifications.

Typical Scenario

The usual scenario for an spaper follows. The spaper is created and inserted onto the screen. The spaper receives pen strokes which it passes on to its scribble which in turn passes them on to a translator. At some point, the spaper is "complete" either via an external notification or optionally when an out of proximity event is received. The spaper notifies the scribble and the scribble notifies the translator. When the translator is complete, it notifies the spaper which in turn notifies its listener. The listener then asks the spaper for the translated data and the spaper gets the data from the translator.

Here's a typical flow of messages between the sPaper, its scribble, its translator and the sPaper's listener. (This scenario uses messages defined in input.h, pen.h, xlate.h and scribble.h)
When the sPaper receives a msgInputEvent that contains a stroke (see pen.h) it self sends msgSPaperAddStroke, which sends msgScrAddStroke to the scribble.

```
input system --> msgInputEvent --> sPaper
sPaper --> msgScrAddStroke --> scribble
```

The scribble then sends msgScrAddStroke to its observers. One of the scribble's observers is the sPaper's translator.

```
scribble --> msgScrAddStroke --> translator
```

Eventually sPaper receives msgSPaperComplete. (A client may send msgSPaperComplete to the sPaper. Alternatively, depending on the sPaper's flags, the sPaper may self send msgSPaperComplete. For example, see the description of the spProx flag elsewhere in this file.) In response to msgSPaperComplete, the sPaper sends msgScrComplete to the scribble. In turn, the scribble notifies its observers (including the translator) with msgScrCompleted.

```
sPaper --> msgScrComplete --> scribble
scribble --> msgScrCompleted --> translator
```

The translator responds to msgScrCompleted by sending msgXlateCompleted to its observers, which include the sPaper. The sPaper responds to msgXlateCompleted by sending msgSPaperXlateCompleted to its observers, which include the listener.

```
translator --> msgXlateCompleted --> sPaper
sPaper --> msgSPaperXlateCompleted --> listener
```

The listener typically sends msgSPaperGetXlateData to the sPaper to retrieve the translated data. In response to msgSPaperGetXlateData, the sPaper sends msgXlateData to the translator.

```
listener --> msgSPaperGetXlateData --> sPaper
sPaper --> msgXlateData --> translator
```

### Debugging Flags

clsSPaper uses the debugging flag set 'h'. Defined flags are:

- 0010 General sPaper debugging information
- 0020 sPaper translation debugging information
- 0080 sPaper save and restore debugging information

### Relationship to clsGWin

Although sPaper is a descendent of clsGWin, it inherits little of clsGWin's behavior. All input and translation behavior is overridden.

```
#ifndef SPAPER_INCLUDED
#define SPAPER_INCLUDED
#else include <go.h>
#endif
#else ifdef UID_INCLUDED
#include <uid.h>
#endif
#else ifdef OSHEAP_INCLUDED
#include <osheap.h>
#endif
#else ifdef WIN_INCLUDED
#include <win.h>
#endif
```
Common #defines and typedefs

typedef OBJECT SPAPER;

Flags

These flags are set in pNew->sPaper.flags field, and can be manipulated using msgSPaperSetFlags and msgSPaperGetFlags.

- spCapture. If false, the sPaper destroys an existing scribble and creates a new one when the first stroke since the last translation is received. If true, the scribble is preserved between translations. See the "SPaper Components" section for more information.
- spProx. If true, the sPaper self sends msgSPaperComplete when msgPenOutProxUp is received. In effect, setting this flag causes the sPaper to spontaneously translate when an "out of proximity" event occurs.
- spFixedPos. If true, the sPaper keeps the top-left corner of its scribble fixed distance from the top-left corner of self during a resize operation.
- spPenCoords. If true, xlists returned by the sPaper have pen coordinate rather than LWC coordinates.
- spGrab. If true, the sPaper grabs input in response to msgPenDown and releases the grab in response to msgSPaperAbort or msgSPaperComplete.
- spScribbleEdit. If true (the default), allows the user to delete scribbles via scratch out. sPaper implements a VERY rudimentary "scratch out" gesture. If spScribbleEdit is true and the user draws just the right "scratch out" gesture the strokes under the gesture are deleted. This does NOT use PenPoint's general gesture translation facilities.
- spRedisplay. If true (the default), the sPaper redisplays its scribble's strokes whenever anything changes.
- spSuppressMarks. If true, the following flags are treated as false: spRuling, spVRuling, spGrid, spTick, and spBaseLine.
- spRuling. If true (the default), horizontal ruling lines are drawn.
- spVRuling. If true, vertical ruling lines are drawn.
- spGrid. If true, grid lines are drawn.
- spTick. If true, tick marks are drawn.
- spBaseLine. If true, and spRuling is also true, the horizontal ruling lines are drawn as a baseline.
- spDataMoveable. If true, then the sPaper's scribble is moveable.
- spDataCopyable. If true, then the sPaper's scribble is copyable.

#define spCapture flag0
#define spProx flag4
#define spFixedPos flag5
#define spPenCoords flag6
#define spGrab flag8
#define spScribbleEdit flag11
#define spRedisplay flag7
#define spSuppressMarks flag12
#define spRuling flag1
#define spVRuling flag13
#define spGrid flag9
#define spBaseline flag14
#define spTick flag10
#define spDataMoveable flag2
#define spDataCopyable flag3

Messages

msgNew

Creates an sPaper object.

Takes P_SPAPER_NEW, returns STATUS. Category: class message.

typedef struct SPAPER_NEW ONLY {
    U16 flags;
    U16 lineHeight;  // Cell height (in points)
    OBJECT translator;  // Translation object
    OBJECT listener;  // This object is made an observer of the
    // sPaper.
    U16 rows;  // Rows for shrink wrap layout
    U16 cols;  // Cols for shrink wrap layout
    U16 charWidth;  // Cell width (in points)
    U32 reserved;
} SPAPER_NEW ONLY;
#define sPaperNewFields
#define viewNewFields

typedef struct SPAPER_NEW {
    sPaperNewFields
} SPAPER_NEW, *P_SPAPER_NEW;

msgNewDefaults

Initializes the NEW structure to default values.

Takes P_SPAPER_NEW, returns STATUS. Category: class message.

typedef struct SPAPER_NEW {
    sPaperNewFields
} SPAPER_NEW, *P_SPAPER_NEW;

pArgs->win.flags.input
    = (inputOutProx | inputTip | inputStroke | inputInk | inputNoBusy | inputHWTimeout | inputAutoTerm | inputTimeout | inputHoldTimeout);
pArgs->win.flags.style
    |= wsSendGeometry;
pArgs->view.dataObject
    = NULL;
pArgs->view.createDataObject
    = TRUE;
pArgs->sPaper.flags
    = (spRuling | spRedisplay | spScribbleEdit);
pArgs->sPaper.translator
    = NULL;
pArgs->sPaper.rows
    = 0;
pArgs->sPaper.cols
    = 0;
pArgs->sPaper.reserved
    = 0;
pArgs->sPaper.listener
    = NULL;
pArgs->sPaper.lineHeight = 25; // In case read fails.
read.resId = tagPrLineHeight;
read.heap = 0;
read.pData = &pArgs->sPaper.lineHeight;
read.length = SizeOf(U16);
ObjCallRet(msgResReadData, theSystemPreferences, &read, s);

// convert line height from hundredths of inches to points.
pArgs->sPaper.lineHeight = (pArgs->sPaper.lineHeight * 72) / 100;
pArgs->sPaper.charWidth = pArgs->sPaper.lineHeight;

---

**msgSPaperGetFlags**

Passes back the sPaper's flags.

Takes P_U16, returns STATUS.

#define msgSPaperGetFlags MakeMsg(clsSPaper, 19)

---

**msgSPaperSetFlags**

Sets the sPaper's flags.

Takes P_U16, returns STATUS.

#define msgSPaperSetFlags MakeMsg(clsSPaper, 20)

**Comments**

In addition to setting the flags, the scribble self sends msgWinDirtyRect to force itself to redraw with the new flags.

---

**msgSPaperGetTranslator**

Passes back the sPaper's translator object (may be NULL).

Takes P_OBJECT, returns STATUS.

#define msgSPaperGetTranslator MakeMsg(clsSPaper, 16)

---

**msgSPaperSetTranslator**

Replaces the sPaper's translator passes back the old translator.

Takes P_OBJECT, returns STATUS.

#define msgSPaperSetTranslator MakeMsg(clsSPaper, 17)

**Comments**

Important: the old translator is not destroyed. The client is responsible for eventually destroying it.

In response to this message, the sPaper replaces it translator. (The old translator is passed back.) The sPaper adds itself as an observer of the new translator and adds the translator as the translator as an observer of the sPaper's scribble (if one exists).

---

**msgSPaperGetScribble**

Passes back the sPaper scribble object (may be NULL).

Takes P_OBJECT, returns STATUS.

#define msgSPaperGetScribble MakeMsg(clsSPaper, 14)

**Comments**

See the section "sPaper Components" for more information.
**msgSPaperSetScribble**

Replaces the sPaper's scribble and passes back the old scribble.

Takes P_OBJECT, returns STATUS.

```c
#define msgSPaperSetScribble MakeMsg(clsSPaper,15)
```

**Comments**

Important: the old scribble is not destroyed. The client is responsible for eventually destroying it.

In response to this message, the sPaper replaces its scribble. (The old scribble is passed back.) The sPaper makes its translator (if one exists) an observer of the new scribble. This causes all strokes in the new scribble to be sent to the existing translator.

**msgSPaperGetCellMetrics**

Passes back some of sPaper's metrics.

Takes P_SPAPER_CELL_METRICS, returns STATUS.

```c
#define msgSPaperGetCellMetrics MakeMsg(clsSPaper,11)
```

**Arguments**

```c
typedef struct SPAPER_CELL_METRICS {
    RECT32 cellRect;       // centered writing area of the sPaper
    SIZE32 cellSize;       // size of an individual cell based on
    U16 rows;              // number of rows displayed
    U16 cols;              // number of columns displayed
} SPAPER_CELL_METRICS, *P_SPAPER_CELL_METRICS;
```

**Comments**

In response, sPaper passes back pArgs->cellRect, pArgs->cellSize, pArgs->rows and pArgs->cols.

Note that pArgs->rows and pArgs->cols are not the values passed to msgNew. (The values passed to msgNew are used for shrink wrap layout.)

**See Also**

msgSPaperGetSizes

**msgSPaperSetCellMetrics**

Sets the sPaper's cell metrics. Resizes and lays out window.

Takes P_SPAPER_CELL_METRICS, returns STATUS.

```c
#define msgSPaperSetCellMetrics MakeMsg(clsSPaper,13)
```

**Arguments**

```c
typedef struct SPAPER_CELL_METRICS {
    RECT32 cellRect;       // centered writing area of the sPaper
    SIZE32 cellSize;       // size of an individual cell based on
    U16 rows;              // number of rows displayed
    U16 cols;              // number of columns displayed
} SPAPER_CELL_METRICS, *P_SPAPER_CELL_METRICS;
```

**Comments**

In response, sPaper uses the new values of pArgs->cellSize, pArgs->rows and pArgs->cols to compute its new window size. It then self sends msgWinLayout to resize and re-layout self. The new value of the sPaper's cellRect is passed back in pArgs->cellRect.

**msgSPaperGetSizes**

Passes back the sPaper's line height and character width sizes, in points.

Takes P_SIZE16, returns STATUS.

```c
#define msgSPaperGetSizes MakeMsg(clsSPaper,21)
```
See Also

**msgSPaperSetSizes**

Sets the sPaper's line height and character width sizes, in points.

Takes P_SIZE16, returns STATUS.

```
define msgSPaperSetSizes      MakeMsg(clsSPaper, 22)
```

Comments

In response, the sPaper sets its lineHeight and charWidth. It recomputes other sizes that depend on those values, and repaints itself if necessary.

See Also

msgSPaperSetCellMetrics

**msgSPaperClear**

Destroys the sPaper's scribble.

Takes NULL, returns STATUS.

```
define msgSPaperClear      MakeMsg(clsSPaper, 4)
```

Comments

In response, the sPaper destroys its scribble, if it has one.

---

**Stroke Processing Messages**

**msgSPaperAddStroke**

Adds a stroke to the sPaper's scribble.

Takes P_INPUT_EVENT, returns STATUS.

```
define msgSPaperAddStroke      MakeMsg(clsSPaper, 2)
```

Comments

In response to msgPenStroke, the sPaper self sends this message to add a stroke to its scribble. If the sPaper does not have a scribble, one is created. If the sPaper is not capturing input (spCapture flag is false), and this is the first stroke added since the last translation, then any existing scribble is destroyed and a new one is created.

The sPaper self sends msgSPaperLocate before adding the stroke to the scribble to allow subclasses to process the stroke.

**msgSPaperLocate**

Allows subclasses to process the stroke before it is added to the scribble.

Takes P_SPAPER_LOCATE, returns STATUS.

```
define msgSPaperLocate      MakeMsg(clsSPaper, 6)
```

Arguments

```
typedef struct SPAPER_LOCATE {
    XY32 start;      // origin of stroke
    P_UNKNOWN pStroke;      // new stroke
} SPAPER_LOCATE, *P_SPAPER_LOCATE;
```

Comments

An sPaper's default response to this message is to return stsOK.

See Also

msgSPaperAddStroke
msgSPaperDeleteStrokes
Deletes strokes in the sPaper's scribble that intersect *pArgs.
Takes P_RECT32, returns STATUS.
#define msgSPaperDeleteStrokes MakeMsg(clsSPaper,18)
Comments
In response to this message, the sPaper sends msgScrDeleteStrokeArea to its scribble (after the rectangle is converted to the appropriate coordinate system).
If the spRedisplay flag is true, then sPaper also dirties the specified rectangle in itself to cause repainting to occur.

msgSPaperComplete
Tells the sPaper that the current stroke is complete.
Takes nothing, returns STATUS.
#define msgSPaperComplete MakeMsg(clsSPaper,3)
Comments
See the "Typical Scenario" section for a description of why and when this message is sent.
sPaper responds as follows. If the sPaper has a scribble, it sends msgScrComplete to the scribble. If there is no scribble, the sPaper self sends msgSPaperXlateCompleted to "complete" the translation, even though the resulting translation will be empty.
If this message is received while the sPaper is handling msgInputEvent, the status returned from msgInputEvent will cause any grab to be released.
See Also
msgSPaperXlateCompleted

msgSPaperAbort
Tells the sPaper to abort the entry of the current stroke.
Takes nothing, returns STATUS.
#define msgSPaperAbort MakeMsg(clsSPaper,23)
Comments
In response to this message, sPaper sends msgSPaperClear to self.
If this message is received while the sPaper is handling msgInputEvent, the status returned from msgInputEvent will cause any grab to be released.

Data Notification and Retrieval Messages

msgSPaperXlateCompleted
Notifies observers that data is available from the sPaper.
Takes OBJECT, returns STATUS.
#define msgSPaperXlateCompleted MakeMsg(clsSPaper,128)
Comments
This message has two roles.
Role 1: This notification is sent to the sPaper's observers (including the listener) when the sPaper decides that translation is complete. Note that the resulting "translation" might be empty.
Role 2: sPaper self sends this message when msgSPaperComplete has been received and there is nothing to translate. In response to this message, sPaper sends the same message to its observers, as described in Role 1 above.

**msgSPaperGetXlateData**

Passes back translated data.

Takes P_XLATE_DATA, returns STATUS.

```c
#define msgSPaperGetXlateData MakeMsg(clsSPaper, 7)
```

Comments

The sPaper's observers typically send this message in response to the sPaper's msgSPaperCompleted notification. See the "Typical Scenario" section for more information.

If there is no translator, or no scribbles to be translated, the sPaper passes back an empty xlist.

Otherwise, the sPaper extracts the xlist from its translator. If the sPaper's spPenCoords flag is true, the sPaper converts the xlist's coordinates to pen coordinates; otherwise it converts the xlist's coordinates to local window coordinates. Finally, the sPaper passes back the xlist.

The client must free the passed back xlist.

See Also

msgSPaperGetXlateDataAndStrokes.h

**msgSPaperGetXlateDataAndStrokes**

Passes back translated data and its associated strokes.

Takes P_SPAPER_XDATA, returns STATUS.

```c
typedef struct SPAPER_XDATA {
    OS_HEAP_ID heap;            // In: Heap to allocate space for stroke data
    PUNKNOWN pXList;            // (Null means to use osProcessHeapId.)
    BOOLEAN toLWC;              // In: true to convert strokes to LWC coordinates
} SPAPER_XDATA, *P_SPAPER_XDATA;
```

```c
#define msgSPaperGetXlateDataAndStrokes MakeMsg(clsSPaper, 8)
```

Comments

The sPaper's observers typically send this message (or msgSPaperGetXlateData) in response to the sPaper's msgSPaperCompleted notification. See the "Typical Scenario" section for more information.

This message is very similar in function to msgSPaperGetXlateData. In fact the first two fields of pArgs for this message are the same as the fields of pArgs for msgSPaperGetXlateData.

The only difference between the two messages is that msgSPaperGetXlateDataAndStrokes also passes back the stroke information used to produce the translation. The strokes are appended to the xlist as elements of type xtStroke16.

If pArgs->toLWC is true, then the coordinate information in the strokes is converted to Local Window Coordinates (see win.h) before being passed back.

The client must free the passed back xlist.

See Also

msgSPaperGetXlateData
### Messages Defined by Other Classes

#### msgFree
- Defined in clsmgr.h
- Takes P_OBJ_KEY, returns STATUS.

**Comments**
- If the sPaper contains a scribble, it first removes the translator (if it exists) as an observer of the scribble. It then sends msgDestroy to the scribble.
- If the sPaper contains a translator, it first remove self as an observer of the translator and then send msgDestroy to the translator.

#### msgSave
- Defined in clsmgr.h.
- Takes P_OBJ_SAVE, returns STATUS.

**Comments**
- An sPaper responds by sending msgResPutObject to its scribble and translator. (If the scribble and/or translator is null, this effectively writes the "null object" id into the resource file.)

#### msgRestore
- Defined in clsmgr.h.
- Takes P_OBJ_RESTORE, returns STATUS.

**Comments**
- An sPaper responds by sending msgResGetObject to pArgs->file to restore its scribble and translator that were saved while handling msgSave.
- If the restored translator is non-null, the sPaper makes itself an observer of the of the translator. If both the translator and scribble are non-null, the sPaper makes the translator an observer of the scribble.

#### msgSetOwner
- Defined in clsmgr.h.
- Takes P_OBJ_OWNER, returns STATUS.

**Comments**
- In response, an sPaper sends this message to its translator and scribble (if they are non-null). The sPaper then lets its ancestor (clsObject) set the sPaper's ownership.

#### msgXlateCompleted
- Defined in xlate.h.
- Takes nothing, returns STATUS.

**Comments**
- An sPaper receives this message because it is observing its translator. The translator uses this message to indicate that translation has been complete and that data is available.
- In response to this message the sPaper self sends msgSPaperXlateCompleted, which results in msgSPaperXlateCompleted being sent to all the sPaper's observers.

**See Also**
- msgSPaperXlateCompleted
**msgWinRepaint**
Defined in win.h.

Takes nothing, returns STATUS.

Comments
An sPaper responds by (1) drawing any necessary grid lines in the window, and (2) if spRedisplay is true, sending msgScrRender to its scribble

**msgWinSized**
Defined in win.h.

Takes P_WIN_METRICS, returns STATUS.

Comments
If the window being resized is self, and a change in height has occurred, and the spFixedPos flag is true, then the sPaper's scribble's base is adjusted by the change in height. This causes the scribble to remain at a fixed position relative to the upper left corner of the window. As a result of handling this message, msgSPaperGetCellMetrics and msgSPaperGetSizes will return different values.

See Also
scribble.h

**msgWinLayoutSelf**
Defined in win.h.

Takes P_WIN_METRICS, returns STATUS.

Comments
If wsLayoutResize is on in pArgs->options, the sPaper picks a width of

\[(\text{cols} \times \text{cellWidth}) + \text{self's borderSize.w}\]

and a height of

\[(\text{rows} \times \text{lineHeight}) + \text{self's borderSize.h}\]

**msgInputEvent**
Defined in input.h.

Takes P_INPUT_EVENT, returns STATUS.

Comments
sPaper handles msgPenUp, msgPenDown, msgPenStroke and msgPenOutProxUp events.

An sPaper grabs input by returning stsInputGrabTerminate in response to msgPenDown.

If flags.spGrab is false, the sPaper relinquishes the grab by returning stsInputTerminate in response to msgPenUp.

If flags.spGrab is true, the sPaper releases the grab by returning stsInputTerminate in response to msgPenOutProxUp. msgPenOutProxUp also cause a self send of msgSPaperComplete if flags.spProx is set.

msgPenStroke causes a self send of msgSPaperAddStroke.

All other msgInputEvent events return stsInputGrabIgnored or stsInputIgnored depending on the grab state the sPaper is in.

Return Value
stsInputTerminate

See Also
msgSPaperComplete
msgSelDelete
Defined in sel.h.
Takes U32, returns STATUS.
Comments
In response to this message, the sPaper self sends msgSPaperClear.

msgSelMoveSelection
Defined in sel.h.
Takes P_XY32, returns STATUS.
Comments
In response to this message, the sPaper first checks to see if the selection owner can "speak" the xferScribbleObject data transfer type. If it cannot, then the sPaper lets its ancestor process the message. If it can, and the selection owner is not self, then the sPaper gets the scribble from the selection owner, positions it as specified in pArgs, self sends msgSPaperSetScribble, and finally sends msgSelDelete to the selection owner.

msgSelCopySelection
Defined in sel.h.
Takes P_XY32, returns STATUS.
Comments
An sPaper's response to this message is identical to its response to msgSelMoveSelection except that the sPaper does not send msgSelDelete to the selection owner.
See Also
msgSelMoveSelection

msgSelBeginMove
Defined in sel.h.
Takes P_XY32, returns STATUS.
Comments
In response to this message, the sPaper first verifies that it has a scribble and that flags.spDataMoveable is true. If either of these fail, the sPaper lets its ancestor process the message. Otherwise the sPaper computes the bounding box of the scribble and self sends msgEmbeddedWinBeginMove.

msgSelBeginCopy
Defined in sel.h.
Takes P_XY32, returns STATUS.
Comments
In response to this message, the sPaper first verifies that it has a scribble and that flags.spDataCopyable is true. If either of these fail, the sPaper lets its ancestor process the message. Otherwise the sPaper computes the bounding box of the scribble and self sends msgEmbeddedWinBeginCopy.
**msgXferGet**  
Defined in xfer.h.  
Takes P_UNKNOWN, returns STATUS.

**Comments**  
If pArgs->id is xferScribbleObject, the sPaper creates a copy of its scribble and returns the copy in pArgs->uid.

**msgXferList**  
Defined in xfer.h.  
Takes OBJECT, returns STATUS.

**Comments**  
In response to this message, the sPaper adds the data transfer type xferScribbleObject to the list of data transfer types.

**msgTrackProvideMetrics**  
Defined in track.h.  
Takes P_TRACK_METRICS, returns STATUS.

**Comments**  
If pArgs->tag is tagMoveCopyIconTrack, the sPaper snaps the pen to the center-left of the move/copy icon.
XGESTURE.H

Interface file for clsXGesture

clsXGesture inherits from clsXtract.

#ifndef XGESTURE INCLUDED
#define XGESTURE INCLUDED
#endif
#ifndef GO INCLUDED
#include <go.h>
#endif
#ifndef UID INCLUDED
#include <uid.h>
#endif
#ifndef GEO INCLUDED
#include <geo.h>
#endif

Common #defines and typedefs

Gesture Definitions

These tags define the codes returned for a recognized gesture. Wherever a "gesture id" is called for, one of these codes is expected.

Certain of these gesture codes are OBSOLETE. That is, the shapes that they denote were experimental and are no longer recognized by the gesture recognizer. All such obsolete codes are indicated by the comment "not generated" at the end of the definition.

#define xgsNull MakeTag(clsXGesture, 0xff) // 255
// selection
#define xgsLeftParen MakeTag(clsXGesture, '(') // 40
#define xgsRightParen MakeTag(clsXGesture, ')') // 41
#define xgsPlus MakeTag(clsXGesture, '+') // 43
#define xgs1Tap MakeTag(clsXGesture, '.') // 46
#define xgs2Tap MakeTag(clsXGesture, 0x80) // 128
#define xgs3Tap MakeTag(clsXGesture, 0x81) // 129
#define xgs4Tap MakeTag(clsXGesture, 0x82) // 130
// Removed xgsNTapDrag
#define xgsPlusTap MakeTag(clsXGesture, 0x87) // 135 not generated
#define xgsCheckTap MakeTag(clsXGesture, 0x88) // 136
#define xgsTapHold MakeTag(clsXGesture, 0x89) // 137
#define xgsPressHold MakeTag(clsXGesture, 0x8a) // 138
// deletion
#define xgsCross MakeTag(clsXGesture, 'X') // 88 == xgsXGesture
#define xgsPigtailHorz MakeTag(clsXGesture, 0x8b) // 139 not generated
#define xgsScratchOut MakeTag(clsXGesture, 0x8c) // 140
#define xgsPigtailVert MakeTag(clsXGesture, 0x8d) // 141
// insert/replace
#define xgsCircle MakeTag(clsXGesture, '0')  // 79 == xgsOGesture
#define xgsCircleTap MakeTag(clsXGesture, 0x8a)  // 142
#define xgsUpCarett MakeTag(clsXGesture, 0x8f)  // 143
#define xgsRightCarett MakeTag(clsXGesture, 0x90)  // 144 not generated
#define xgsCircleDblTap MakeTag(clsXGesture, 0x91)  // 145 not generated
#define xgsCircleLine MakeTag(clsXGesture, 0x92)  // 146
#define xgsCircleFlickUp MakeTag(clsXGesture, 0x93)  // 147
#define xgsCircleFlickDown MakeTag(clsXGesture, 0x94)  // 148
#define xgsUpCaretDot MakeTag(clsXGesture, 0x95)  // 149
#define xgsUpCaretDblDot MakeTag(clsXGesture, 0x96)  // 150 not generated
#define xgsDblArrow MakeTag(clsXGesture, 0x97)  // 151 not generated
#define xgsDblCircle MakeTag(clsXGesture, 0x98)  // 152
// move/copy
#define xgsUpArrow MakeTag(clsXGesture, 0x99)  // 153
#define xgsUp2Arrow MakeTag(clsXGesture, 0x9a)  // 154
#define xgsDownArrow MakeTag(clsXGesture, 0x9b)  // 155
#define xgsDown2Arrow MakeTag(clsXGesture, 0x9c)  // 156
#define xgsLeftArrow MakeTag(clsXGesture, 0x9d)  // 157
#define xgsLeft2Arrow MakeTag(clsXGesture, 0x9e)  // 158
#define xgsRightArrow MakeTag(clsXGesture, 0x9f)  // 159
#define xgsRight2Arrow MakeTag(clsXGesture, 0xa0)  // 160
#define xgsDblUpArrow MakeTag(clsXGesture, 0xa1)  // 161
#define xgsDblDownArrow MakeTag(clsXGesture, 0xa2)  // 162 not generated
#define xgsUpTriangle MakeTag(clsXGesture, 0xa3)  // 163 not generated
#define xgsDownTriangle MakeTag(clsXGesture, 0xa4)  // 164 not generated
#define xgsRightUp MakeTag(clsXGesture, 0xa5)  // 165
#define xgsRightUpFlick MakeTag(clsXGesture, 0xa6)  // 166
#define xgsRightDown MakeTag(clsXGesture, 0xa7)  // 167
// white space
#define xgsCaret MakeTag(clsXGesture, 'C')  // 67
#define xgsLCorner MakeTag(clsXGesture, 'L')  // 76 "DownRight", "LGesture"
#define xgsLLCornerFlick MakeTag(clsXGesture, 0xa8)  // 168 "DownRightFlick"
#define xgsLCornerFlick MakeTag(clsXGesture, 0xa9)  // 169 "DownLeft"
#define xgsLRCornerFlick MakeTag(clsXGesture, 0xaa)  // 170 "DownLeftFlick"
#define xgsParagraph MakeTag(clsXGesture, 0xab)  // 171
#define xgsLeftCaret MakeTag(clsXGesture, 0xac)  // 172 not generated
#define xgsULCorner MakeTag(clsXGesture, 0xad)  // 173 "UpRight"
// scroll
#define xgsFlickUp MakeTag(clsXGesture, 0xae)  // 174
#define xgsFlickDown MakeTag(clsXGesture, 0xaf)  // 175
#define xgsFlickLeft MakeTag(clsXGesture, 0xb0)  // 176
#define xgsFlickRight MakeTag(clsXGesture, 0xb1)  // 177
#define xgsDblFlickUp MakeTag(clsXGesture, 0xb2)  // 178
#define xgsDblFlickDown MakeTag(clsXGesture, 0xb3)  // 179
#define xgsDblFlickLeft MakeTag(clsXGesture, 0xb4)  // 180
#define xgsDblFlickRight MakeTag(clsXGesture, 0xb5)  // 181
#define xgsFlickTapUp MakeTag(clsXGesture, 0xb6)  // 182 not generated
#define xgsFlickTapDown MakeTag(clsXGesture, 0xb7)  // 183 not generated
#define xgsFlickTapLeft MakeTag(clsXGesture, 0xb8)  // 184 not generated
#define xgsFlickTapRight MakeTag(clsXGesture, 0xb9)  // 185 not generated
#define xgsTrplFlickUp MakeTag(clsXGesture, 0xba)  // 186
#define xgsTrplFlickDown MakeTag(clsXGesture, 0xbb)  // 187
#define xgsTrplFlickLeft MakeTag(clsXGesture, 0xbc)  // 188
#define xgsTrplFlickRight MakeTag(clsXGesture, 0xbd)  // 189
#define xgsQuadFlickUp MakeTag(clsXGesture, 0xbe)  // 190
#define xgsQuadFlickDown MakeTag(clsXGesture, 0xbf)  // 191
#define xgsQuadFlickLeft MakeTag(clsXGesture, 0xc0)  // 192
#define xgsQuadFlickRight MakeTag(clsXGesture, 0xc1)  // 193
// misc
#define xgsLineCaretRight MakeTag(clsXGesture, 0xc2)  // 194 not generated
#define xgsLineCaretLeft MakeTag(clsXGesture, 0xc3)  // 195 not generated
#define xgsLineDblCaret MakeTag(clsXGesture, 0xc4)  // 196 not generated
// User-defineable
#define xgsLeftDown MakeTag(clsXGesture, 0xc5) // 197
#define xgsLeftUp MakeTag(clsXGesture, 0xc6) // 198
#define xgsUpLeft MakeTag(clsXGesture, 0xc7) // 199

// Undo
#define xgsVertCounterFlick MakeTag(clsXGesture, 0xc8) // 200
#define xgsHorzCounterFlick MakeTag(clsXGesture, 0xc9) // 201
#define xgsInfinity MakeTag(clsXGesture, 0xca) // 202 not generated
#define xgsCircleCrossOut MakeTag(clsXGesture, 0xcb) // 203

// Borders On
#define xgsBordersOn MakeTag(clsXGesture, 0xcc) // 204
#define xgsAsterisk MakeTag(clsXGesture, '*') // not generated

// Capital letters gestures
#define xgsAGesture MakeTag(clsXGesture, 'A') // 65
#define xgsBGesture MakeTag(clsXGesture, 'B') // 66
#define xgsCGesture MakeTag(clsXGesture, 'C') // 67 for xgsCGesture see above
#define xgsDGesture MakeTag(clsXGesture, 'D') // 68
#define xgsEGesture MakeTag(clsXGesture, 'E') // 69
#define xgsFGesture MakeTag(clsXGesture, 'F') // 70
#define xgsGGesture MakeTag(clsXGesture, 'G') // 71
#define xgsHGesture MakeTag(clsXGesture, 'H') // 72
#define xgsIGesture MakeTag(clsXGesture, 'I') // 73
#define xgsJGesture MakeTag(clsXGesture, 'J') // 74
#define xgsKGesture MakeTag(clsXGesture, 'K') // 75
#define xgsLLCorner MakeTag(clsXGesture, 'L') // 76 for xgsLLCorner, above
#define xgsMGesture MakeTag(clsXGesture, 'M') // 77
#define xgsNGesture MakeTag(clsXGesture, 'N') // 78
#define xgsOGesture MakeTag(clsXGesture, 'O') // 79 xgsCircle
#define xgsPGesture MakeTag(clsXGesture, 'P') // 80
#define xgsQGesture MakeTag(clsX Gesture, 'Q') // 81
#define xgsRGesture MakeTag(clsX Gesture, 'R') // 82
#define xgsSGesture MakeTag(clsX Gesture, 'S') // 83
#define xgsTGesture MakeTag(clsX Gesture, 'T') // 84
#define xgsUGesture MakeTag(clsX Gesture, 'U') // 85
#define xgsVGesture MakeTag(clsX Gesture, 'V') // 86 == xgsVGesture
#define xgsWGesture MakeTag(clsX Gesture, 'W') // 87
#define xgsXGesture MakeTag(clsX Gesture, 'X') // 88 == xgsCross
#define xgsYGesture MakeTag(clsX Gesture, 'Y') // 89
#define xgsZGesture MakeTag(clsX Gesture, 'Z') // 90
#define xgsQuestion MakeTag(clsX Gesture, '?') // 91

// graphic gestures in geo.ptc - currently not implemented
#define xgsRect MakeTag(clsXGesture, 0xf0) // 240 not generated
#define xgsRoundRect MakeTag(clsXGesture, 0xf1) // 241 not generated
#define xgsSpline MakeTag(clsXGesture, 0xf2) // 242 not generated
#define xgsPolyline MakeTag(clsXGesture, 0xf3) // 243 not generated
#define xgs0TapHold xgsPressHold
#define xgs1TapHold xgsTapHold
#define xgs2TapHold MakeTag(clsXGesture, 0xf4) // 244
#define xgs3TapHold MakeTag(clsX Gesture, 0xf5) // 245
#define xgs4TapHold MakeTag(clsX Gesture, 0xf6) // 246

typedef struct XLATE_GDATA {
  U32 gType;  // gesture code (one of the 32-bit values defined above)
  XY32 hotPoint;  // target point in window coordinates
} XLATE_GDATA, *P_XLATE_GDATA;

Output Data Structure

Information returned in an xlist.
Messages

msgNewDefaults:
Sets default values in XLATE_NEW structure for a gesture recognizer
Takes P_XLATE_NEW, returns STATUS.

Comments
Sets

```
pArgs->xlate.metrics.charCount = 1;
pArgs->xlate.metrics.lineCount = 1;
```

and all other values to 0

msgNew:
Creates a new Gesture translation object.
Takes P_XLATE_NEW, returns STATUS.

Comments
Note: sets the XLATE_NEW.mode to xlGesture, regardless of the value passed in via pArgs.

Notification Messages

msgXGestureComplete:
Hook for subclasses to postprocess the results of gesture recognition.
Takes NULL, returns STATUS.

```
#define msgXGestureComplete
#endif
```

Comments
Not implemented.
This file contains part of the API definition for clsXtract. For the remainder see xtract.h.

clsXtract inherits from clsObject.

Implements basic translation functions for converting pen input, in the form of strokes, to gestures or text characters.

Translators are objects that use pattern recognition techniques to convert pen input to gestures or text characters. There are three stages to the translation process: initialization, control (stroke collection and recognition), and notification (data output).

Since the translation object may preprocess input data as it is received, initialization messages should be sent before any strokes are added to the object. Initialization messages establish the rules for translation.

Control messages are used by the client to communicate specific information regarding the state of the translation as it pertains to the input stroke stream.

Notification messages are used by the translation object to notify the client as to the current state of the translation process.

For historical reasons messages and data types relating to translation are defined in terms of two class names: clsXlate and clsXtract. Conceptually, clsXlate is an abstract class (a class with no default behavior, i.e. no methods) and clsXtract is a subclass of clsXlate which implements methods for a large number of messages. As implemented, however, there is no such class as clsXlate in PenPoint 1.0. When PenPoint boots, clsXlate is not installed in the class hierarchy, and clsXtract is installed as a subclass of clsObject.

The clsXtract/clsXlate does not implement enough behavior to be used directly as a translator. Rather translation objects should be created as instances of one of the following subclasses:

- clsXGesture for gestures
- clsXText for letters with minimal language support
- clsXWord for letters as part of normal American English
- clsXTeach for letters when the application is to train the recognition engine. (It is not possible to train gestures)

See Also

xtract.h, xgesture.h, xtext.h, xword.h, xteach.h
Common #defines and typedefs

Internal Constants

The following are used globally by the translation object.

#define xltCharWordTerminator ('\0') // standard string terminator
#define xltCharSpace (' ') // character code for space
#define xltCharDotlessI (0x80) // character code for dotless i (private)
#define xltCharDotlessJ (0x81) // character code for dotless j (private)
#define xltCharUnknownDefault (0x15) // default "meatball" for unrecognized char
#define xltMaxWordLength (32) // buffer size for word translations

typedef struct POINT {
    S16 x, y;
} POINT, *P_POINT; // internal representation of a digitizer point

Status Values

The translation object may return the following status values.

#define stsXlateBufferOverflow MakeStatus(clsXlate, 1)
#define stsXlateBadProtoFile MakeStatus(clsXlate, 2)
#define stsXlateBadTransFile MakeStatus(clsXlate, 3)
#define stsXlateBadTrigramFile MakeStatus(clsXlate, 4)
#define stsXlateInputTruncated MakeNonErr(clsXlate, 1)

Creation Messages

Characteristics of the insertion pad.

typedef struct XLATE_METRICS {
    U16 lineCount;  // number of lines (0 = indeterminate)
    U16 charCount;  // number of character columns (0 = indeterminate)
    SIZE32 charBox; // size of character box (height and width)
    S32 baselineOffset; // baseline offset to bottom of char box (if charCount != 0)
    XY32 origin;   // origin of insertion pad in digitizer coordinates
} XLATE_METRICS, *P_XLATE_METRICS;

When "case smarts" are turned on (i.e. xltSmartCaseDisable hwx flag is OFF), the translation object will ignore the case in which the user wrote the input and will instead figure out the correct capitalization based on the settings in XLATE_CASE_METRICS. XLATE_CASE_TYPE tells the type of capitalization rules which the translation string should be made to obey. "No rules" means make everything lower case.

typedef enum XLATE_CASE_TYPE {
    xcmNone,   // Don't capitalize anything, force it all to lower case
    xcmSentence, // Capitalize first letter of each sentence, etc
    xcmField    // Capitalize as per XLATE_CASE_METRICS.context.field
} XLATE_CASE_TYPE, *P_XLATE_CASE_TYPE;

If the writer is a mixed case writer, then he/she is presumed to write both upper case and lower case shapes. An AllCapsWriter, on the other hand, will only write upper case shapes, never lower case shapes. This knowledge can help the shape recognizer by limiting the number of alternatives it has to choose from. This does not mean, however, that the translation will be all upper case, for it is the job of "case smarts" to convert the translation to the correct case.

typedef enum XLATE_CASE_WRITER {
    xcmMixedCaseWriter,  // Writer writes both upper and lower case shapes
    xcmAllCapsWriter,   // Writer writes in all upper case shapes
} XLATE_CASE_WRITER, *P_XLATE_CASE_WRITER;
typedef enum XLATE_CASE_FIELD {
    xcmOneInitialCapField, // capitalize first letter in the field
    xcmAllInitialCapsField, // capitalize first letter in each 'word'
    xcmAllCapsField, // capitalize all letters in the field
} XLATE_CASE_FIELD, *P_XLATE_CASE_FIELD;

typedef struct XLATE_CASE_METRICS {
    XLATE_CASE_TYPE type; // type of rule to use
    XLATE_CASE_WRITER writer; // type of input to expect
    union {
        SPELL_CASE_CONTEXT sentence; // specific rules if type is xcmSentence
        XLATE_CASE_FIELD field; // specific rules if type is xcmField
    } context;
} XLATE_CASE_METRICS, *P_XLATE_CASE_METRICS;

typedef struct XLATE_NEW_ONLY {
    U32 hwxFlags; // xlate rules (see msgXlateSetFlags)
    U16 charConstraints; // constrained char set flags
    XLATE_METRICS metrics; // insertion pad parameters
    P_UNKNOWN pTemplate; // compiled XTemplate; pNull if none.
    XLATE_CASE_METRICS xlateCaseMetrics; // case post-processing controls.
} XLATE_NEW_ONLY, *P_XLATE_NEW_ONLY;

typedef struct XLATE_NEW {
    OBJECT_NEW_ONLY object;
    XLATE_NEW_ONLY xlate;
} XLATE_NEW, *P_XLATE_NEW;

msgNewDefaults:

Initializes the XLATE_NEW structure to default values.

Takes P_XLATE_NEW, returns STATUS. Category: class message.

Message Arguments
typedef struct XLATE_NEW {
    OBJECT_NEW_ONLY object;
    XLATE_NEW_ONLY xlate;
} XLATE_NEW, *P_XLATE_NEW;

Comments
The default values are 0 for everything.

This message should, of course, be sent to one of the subclasses of clsXtract, not to clsXlate, since clsXlate is a fiction, and not to clsXtract, since clsXtract does not implement the complete behavior needed to do translation.

msgNew:

Creates a new translation object.

Takes P_XLATE_NEW, returns STATUS. Category: class message.

Message Arguments
typedef struct XLATE_NEW {
    OBJECT_NEW_ONLY object;
    XLATE_NEW_ONLY xlate;
} XLATE_NEW, *P_XLATE_NEW;

Comments
This message should, of course, be sent to one of the subclasses of clsXtract, not to clsXlate, since clsXlate is a fiction, and not to clsXtract, since clsXtract does not implement the complete behavior needed to do translation.
msgFree:
Destroys a translation object.
Takes P_NULL, returns STATUS.
Comments
This message should be sent to the object you wish to destroy.

**Initialization Messages**

The following messages control various settings and modes which govern the way translation is carried out. These messages must all be received by the translator BEFORE any strokes are received by it, since translators are allowed to begin translating "in the background", (i.e. before the input is complete).

**msgXlateModeSet:**
Sets the mode (i.e. character/code type) of a translation object.
Takes XLIATE_MODE, returns STATUS.

```c
#define msgXlateModeSet     MakeMsg(clsXlate, 5)
```

Arguments
```
typedef enum {
    x1Character,     // obsolete
    x1Text,          // use default text rules (ASCII)
    x1Number,        // obsolete
    x1Gesture,       // use default gesture rules
    x1Geometric      // obsolete
} XLIATE_MODE, *P_XLIATE_MODE;
```

Comments
The translation object can be configured to processes a variety of character/code types. The mode flag determines the type of character set and default behavior for the object.

**msgXlateModeGet:**
Gets the mode of a translation object.
Takes P_XLIATE_MODE, returns STATUS.

```c
#define msgXlateModeGet     MakeMsg(clsXlate, 10)
```

Arguments
```
typedef enum {
    x1Character,     // obsolete
    x1Text,          // use default text rules (ASCII)
    x1Number,        // obsolete
    x1Gesture,       // use default gesture rules
    x1Geometric      // obsolete
} XLIATE_MODE, *P_XLIATE_MODE;
```

Comments
The mode was set either at msgNew time or by msgXlateModeSet.

**msgXlateMetricsSet:**
Tells translator the dimensions and layout of the writing area.
Takes P_XLIATE_METRICS, returns STATUS.

```c
#define msgXlateMetricsSet   MakeMsg(clsXlate, 8)
```

Arguments
```
typedef struct XLIATE_METRICS {
    U16 lineCount;        // number of lines (0 = indeterminate)
    U16 charCount;        // number of character columns (0 = indeterminate)
    SIZE32 charBox;       // size of character box (height and width)
    S32 baselineOffset;   // baseline offset to bottom of char box (if charCount != 0)
    XY32 origin;          // origin of insertion pad in digitizer coordinates
} XLIATE_METRICS, *P_XLIATE_METRICS;
```
In order to assist the writer and the recognition system, an insertion pad can display guidelines, or "character boxes", that direct the writer in targeting. When character boxes are used, the XLATE_METRICS are used to communicate the physical box information to the translation object. The translator can use this information (when available) to decide how to group the strokes into characters.

Most internal processes key off the charCount field. If charCount is 0, the translation object assumes that there are no boxes. In that case it will default to a heuristic algorithm that combines information from the shape matching and context processing to estimate the writing baseline and character spacing.

(As an aside, the translation object does not use baseline information when charCount is 0. I.e. lineCount is ignored in that case.)

If charCount > 0, the translation object uses lineCount and charCount to calculate the number of boxes in the insertion pad. A combination of the charBox height and width and the x and y coordinates of the origin are used to define the physical bounds of each box. The translation object then uses this to determine character segmentation.

### msgXlateMetricsGet:

Gets metrics of a translation object.

Takes P_XLATE_METRICS, returns STATUS.

```c
#define msgXlateMetricsGet MakeMsg(clsXlate, 16)
typedef struct XLATE_METRICS {
    U16 lineCount; // number of lines (0 = indeterminate)
    U16 charCount; // number of character columns (0 = indeterminate)
    SIZE32 charBox; // size of character box (height and width)
    S32 baselineOffset; // baseline offset to bottom of char box (if charCount != 0)
    XY32 origin; // origin of insertion pad in digitizer coordinates
} XLATE_METRICS, *P_XLATE_METRICS;
```

The metrics were set in response to either msgNew or msgXlateMetricsSet.

### msgXlateStringSet:

Sets the current textual context for a translation object.

Takes P_XLATE_STRING, returns STATUS.

```c
#define msgXlateStringSet MakeMsg(clsXlate, 12)
typedef struct XLATE_STRING {
    P_CHAR pCurrentText; // pointer to current text string
    U16 length; // string length
    S16 startIndex; // index of first editable character
    S16 endIndex; // index of last editable character
} XLATE_STRING, *P_XLATE_STRING;
```

The following structure is used to communicate currently displayed text in the insertion pad. It is only applicable when using boxed insertion pads. The existing textual information must be registered if the translation object is using any string-based knowledge source (such as the dictionary or a template) where positional information within the string is crucial for proper recognition.

It is possible to allow only a portion of the displayed string to be in the insertion pad (and hence, editable). To allow for this, startIndex represents the first editable character's position in the string, and endIndex represents the last editable characters's position in the string. If the entire string is editable, set startIndex = 0 and endIndex = string length.
msgXlateSetFlags:

Sets the translation flags.

Takes U32, returns STATUS.

```c
#define msgXlateSetFlags

// Built-in Rules
#define xltSegmentVeto
#define xltCaseEnable
#define xltAlphaNumericEnable
#define xltPunctuationEnable
#define xltVerticalEnable

// Knowledge Source Controls
#define xltSpaceDisable
#define xltConnectedEnable
#define xltSpellingEnable
#define xltSpellingVeto
#define xltSpellingPropose
#define xTemplateEnable
#define xTemplateVeto
#define xTemplatePropose

// Post-processing Rules
#define xltProofEnable
#define xltAbbrEnable
#define xltExpansionEnable
#define xltSmartCaseDisable

// Not currently implemented
#define hwxGeoPolylinelines
#define hwxGeoSingleLines
#define hwxGeoLinesAlways
```

The translation flags (hwXFlags) govern which of the various scoring rules will be applied in choosing the best translation. They include built-in language rules, choice of assisting knowledge sources (speller, templates), and postprocessing rules, such as sentence-level case correction.

**Built-in Rules:** The translation object can be directed to use various default language rules to assist recognition. When a flag is turned on, the translator will show a preference for translations which obey the rule associated with that flag. For example if xltCaseEnable is on, the translator will show a preference for words that are either all lower case, all upper case or all lower case except the first letter.

**Knowledge Source Controls:** The translation object can be directed to use spelling and/or template information in order to assist recognition. Each of these knowledge sources, when it is turned on, has a choice of four modes of operation:

Enable, Enable+Veto, Enable+Propose and Enable+Veto+Propose.

The Enable flag must be ON in all four cases. This enables the use of the knowledge source and causes the translator to show a preference for words which conform to the source (i.e. are in the dictionary or match the template). If the Veto flag is also on, then the translator will ONLY consider translations which conform to the source and will reject all translations which do not. If the Propose flag is also on, it allows the translator to change some letters if it will result in a translation which conforms to the knowledge source even if the raw shape matcher did not suggest those letters.

**Post-processing Rules:** The translation object can apply post-processing rules to assist error-checking and proofing (spell correction). The only processing that is currently implemented is the "smart case" capability. This capability calls for the translator to use linguistic rules to correct the capitalization of the translation. This correction is always applied unless it is disabled by turning the smartCaseDisable flag on.
**msgXlateGetFlags:**

Gets the translation flags of an object.

Takes P_U32, returns STATUS.

```c
#define msgXlateGetFlags MakeMsg(clsXlate, 17)
```

Comments
The translation flags are also called the *hwxFlags*.

**msgXlateFlagsClear:**

Clears the given set of translation flags.

Takes U32, returns STATUS.

```c
#define msgXlateFlagsClear MakeMsg(clsXlate, 15)
```

Comments
Performs the operation

```
hwxFlags ^= ~pArgs;
```

thus turning OFF all flags which are ON in *pArgs* and leaving unchanged those flags which are OFF in *pArgs*.

**msgXlateCharConstraintsSet:**

Sets the character constraints of a translation object.

Takes P_U16, returns STATUS.

```c
#define msgXlateCharConstraintsSet MakeMsg(clsXlate, 11)
#define xltDisableUpperCase flag0 // disallow A thru Z
#define xltDisableLowerCase flag1 // disallow a thru z
#define xltDisableNumerals flag2 // disallow 0 thru 9
#define xltDisableCommonPunct flag3 // disallow ,,.!?:;%$+-*
#define xltDisableOtherPunct flag4 // disallow all other punctuation
```

Comments
Character constraints impose limits on the shapes that the writer is allowed to write. Setting the flag when appropriate may improve translation accuracy or performance since the shape matcher will know that it does not need to consider certain shapes as possibilities.

For example, a numeric-only translator can be constructed by setting all of the disable flags except for *xltDisableNumerals*.

Note that character constraints do not restrict the case of the translation string if "case smarts" are on. For example, case smarts may force the translation to be all lower case letters even if the *xltDisableLowerCase charConstraint flag* is set.

**msgXlateCharConstraintsGet:**

Gets the character constraints of a translation object.

Takes P_U16, returns STATUS.

```c
#define msgXlateCharConstraintsGet MakeMsg(clsXlate, 18)
```

Comments
The *charConstraints* were set in response to either *msgNew* or *msgXlateCharConstraintsSet*
msgXlateTemplateGet:

Gets the template for a translation object.
Takes PP_UNKNOWN, returns STATUS.

#define msgXlateTemplateGet MakeMsg(clsXlate, 13)

Comments
Will return in *pArgs a pointer to the compiled template currently in effect for the translator.

msgXlateTemplateSet:

Sets the template for a translation object.
Takes P_UNKNOWN, returns STATUS.

#define msgXlateTemplateSet MakeMsg(clsXlate, 9)

Comments
The pArg should be a pointer to the "compiled" template created by calling the function
XTemplateCompile() defined in xtemplt.h

msgXlateCharMemorySet:

Sets the current Character memory for character box mode.
Takes P_CHARACTER_MEMORY, returns STATUS.

#define msgXlateCharMemorySet MakeMsg(clsXlate, 22)

Arguments
typedef struct CHARACTER_MEMORY {
    U16 position; // position in the string
    P_CHAR usedCharacters; // list of characters already used
} CHARACTER_MEMORY, *P_CHARACTER_MEMORY;

Comments
In "boxed" mode (which typically is used when editing a short string), the translation object can accept
a list of characters already attempted in this position. This is used to allow ambiguous character shapes
to be translated differently on overwrite.

For example, a writer attempting to enter a lower case "L" may want to avoid repeatedly entering a
straight vertical stroke and receiving a numeral "1" as the translation. The character memory feature
allows a client that keeps track of previously overwritten text to pass this information to the translation
object. The translation object will then disallow any character in the "already tried" string.

This feature is implemented only for single character entries. The Position field refers to the position of
the character in the XLANE_STRING pCurrentText string. Setting character memory for more than one
position for a single translation will result in the character memory being ignored in all positions.

msgXlateCharMemoryGet:

Gets the current Character memory for character box mode.
Takes P_CHARACTER_MEMORY, returns STATUS.

#define msgXlateCharMemoryGet MakeMsg(clsXlate, 27)

Message
typedef struct CHARACTER_MEMORY {
    U16 position; // position in the string
    P_CHAR usedCharacters; // list of characters already used
} CHARACTER_MEMORY, *P_CHARACTER_MEMORY;

Comments
This message is intended for use by subclasses.
msgXlateSetXlateCaseMetrics:
Sets the "smart case" metrics.

Takes P_XLATE_CASE_METRICS, returns STATUS.

```
#define msgXlateSetXlateCaseMetrics MakeMsg(clsXlate, 26)
```

```
typedef struct XLATE_CASE_METRICS {
    XLATE_CASE_TYPE type;     // type of rule to use
    XLATE_CASE_WRITER writer; // type of input to expect
    union {
        SPELL_CASE_CONTEXT sentence; // specific rules if type is xcmSentence
        XLATE_CASE_FIELD field;     // specific rules if type is xcmField
    } context;
} XLATE_CASE_METRICS, *P_XLATE_CASE_METRICS;
```

Comments
The translation object can be directed to use Case (capitalization) heuristics above and beyond the basic `xltCaseEnable` heuristics set in the xlate flags. These rules are communicated via the `XLATE_CASE_METRICS` structure. They are applied in a post-processing pass by the translator, whereas the `hwxFlags` are applied during the initial search for a good translation.

These rules set expectations for input (writer style) as well as output format. The writer (CASE_WRITER) field prepares the system for the type of input, allowing either mixed case or all upper case input. The type (CASE_TYPE) field sets the style of heuristics. The context field sets the specific rules to implement.

See `spell.h` for definitions for `SPELL_CASE_CONTEXT`.

msgXlateGetXlateCaseMetrics:
Gets the "smart case" metrics.

Takes P_XLATE_CASE_METRICS, returns STATUS.

```
#define msgXlateGetXlateCaseMetrics MakeMsg(clsXlate, 25)
```

```
typedef struct XLATE_CASE_METRICS {
    XLATE_CASE_TYPE type;     // type of rule to use
    XLATE_CASE_WRITER writer; // type of input to expect
    union {
        SPELL_CASE_CONTEXT sentence; // specific rules if type is xcmSentence
        XLATE_CASE_FIELD field;     // specific rules if type is xcmField
    } context;
} XLATE_CASE_METRICS, *P_XLATE_CASE_METRICS;
```

Returns the values that were set either at `msgNew` time or by `msgXlateSetXlateCaseMetrics`.

msgXlateGetHistoryTemplate:
Gets the current alternate Translation Template.

Takes PP_UNKNOWN, returns STATUS.

```
#define msgXlateGetHistoryTemplate MakeMsg(clsXlate, 23)
```

```
There is no behavior of class xlate associated with the history template other than to respond to the Set and Get messages. It may used by the client to implement a "history" or cache mechanism, allowing the system to "remember" things previously translated.
```
**msgXlateSetHistoryTemplate:**
Sets the current alternate Translation Template.
Takes P_UNKNOWN, returns STATUS.

```c
#define msgXlateSetHistoryTemplate MakeMsg(clsXlate, 24)
```

### Control Messages

**msgXlateComplete:**
Initiates completion of translation after input is complete.
Takes NULL, returns STATUS.

```c
#define msgXlateComplete MakeMsg(clsXlate, 3)
```

**Comments**
Obsolete. See `msgXtractComplete` in xtract.h.
Not to be confused with `msgXlateCompleted` (see below).

Other control messages are defined in xtract.h. In general, the client does not need to play an active role in sending or receiving control messages.

### Notification Messages

**msgXlateData:**
Allows a client to read the results from a translation object.
Takes P_XLATE_DATA, returns STATUS.

```c
#define msgXlateData MakeMsg(clsXlate, 2)
```

**Arguments**

```c
typedef struct XLATE_DATA {
    OS_HEAP_ID heap; // In: heap to allocate structures
    struct XLIST *pXList; // Out: pointer to return info
} XLATE_DATA, *P_XLATE_DATA;

typedef struct XLATE_BDATA { // bounding information
    RECT32 box; // bounding information
    S32 baseline; // baseline offset
} XLATE_BDATA, *P_XLATE_BDATA;

typedef struct WORD_ENTRY { // structure for a word
    S16 score; // confidence factor
    CHAR string[xltMaxWordLength]; // word
} WORD_ENTRY, *P_WORD_ENTRY;

typedef struct WORD_LIST { // structure for a list of words
    RECT32 bound; // bounding information
    U16 count; // number of words in list
    WORD_ENTRY word[1]; // variable length array of words
} WORD_LIST, *P_WORD_LIST;
```

**Comments**
The client reads the translation results from the translation object via this message.

The translation object fills in the clients xlist data with the output data. The specific xlist type is dependent upon the specific translation class. Please refer to xlist.h for the information on each translation class.
The output data is only available upon completion of the translation process. Partial data cannot be read before the client has received the completion notification message (msgXlateCompleted) from the translation object (see below).

The output data is a read-once function. That is, you cannot send msgXlateData twice to the same translator. All translation object internal resources pertaining to the translated data are freed during the reading process.

This message must be sent to an instance of one of the subclasses of clsXtract, such as clsXText or clsXGesture. The clsXtract itself does not implement any behavior for this message.

**msgXlateCompleted:**

Notification to client that the translation has been completed.

Takes OBJECT, returns STATUS.

```c
#define msgXlateCompleted MakeMsg(clsXlate, 128)
```

This notification is sent by the translation object to its observers to inform them that translation is completed. Upon receiving this message the client should send msgXlateData (see above) back to the translator to read the output.

The pArgs is the id of the translator.
This file contains the API definition for some of the xlist filters. xlist filters provide a mechanism to alter the contents of an xlist.

Xlists are a dynamic list of dynamic items. Their API is defined in the file xlist.h. This file simply defines a filter function to operate on the xlist. This function should have probably been included in the file xlist.h.

See Also

xlist.h

#ifndef XLFILTER_INCLUDED
#define XLFILTER_INCLUDED
#ifndef XLIST_INCLUDED
#include <xlist.h>
#endif

XList2Text

Converts a translator xlist to lines of xtText & xtBounds.

Returns STATUS.

Function Prototype

STATUS EXPORTED XList2Text ( 
    P_XLIST pXList);

Comments

Converts xlist of the form:

[xtBounds xtTextWord [xtTextWord]] xtTextListEnd

into:

[xtBounds xtText]

where xtText is the space delimited xtTextWords.

Sets the xlfXList2Text flag in the xlist to indicate that the filter has been executed on this list. A subsequent invocation of XList2Text with this flag set will return stsOK without processing any data. Turning this flag off will cause another pass over the data. This will have no side affects.

See Also

xlist.h
This file contains the API definition for xlist. Xlists provide a set of dynamic list routines used by translators.

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

An xlist is a set of routines for manipulating a list of items of data type P_XLIST_ELEMENT. These items are allocated from a heap passed into the xlist when it is created. Elements have some flag settings, a data type, and a pointer. The pointer points to data defined by the data type, whose allocation is dependent on the flag settings.

Elements in the list are indexed from 0 to entries-1. A series of functions are provide to create and destroy lists, traverse lists, access and set list elements, insert new elements, and delete elements.

In addition, functions are provided to "filter" data from the xlist. These filters either extract useful data from the xlist in the form of a data structure, or actually "mutates" the xlist into an xlist of a different format. These filters are defined in this file and in xlfilter.h.

Xlists of various types are used throughout the system. Primarily, they are used to pass translation information between the hwx system and the client. See xlate.h for example uses in the hwx engine; and gwin.h, spaper.h, or insert.h for example uses inside the UI toolkit.

Typical users create xlists (XListNew), add and delete items (XListInsert, XListDelete), access the value of items (via filters or XListGet), traverse (XListTraverse) and free them (XListFree). Other functions, while useful, are rarely used.

Xlists have associated with them a heap with which use to allocate the memory needed to store the elements (P_XLIST_ELEMENT). They can also use this heap to allocate space for the data pointer field of an element, when the corresponding elements flag setting is xfHeapAlloc. In this situation, the element data pointer will be freed when the xlist is freed, or when XListFreeData is called. Allocating other memory off the xlist heap, although not recommended, is possible. It would be the clients responsibility to free this data. However, typically the user of an xlist will allocate space for the data pointer off of the heap using XListAlloc, insert an element into the xlist with the data, and allow the xlist to manage and free the memory.

#ifndef XLIST_INCLUDED
#define XLIST_INCLUDED
#ifndef GO_INCLUDED
#include <go.h>
#endif
#ifndef OSHEAP_INCLUDED
#include <osheap.h>
#endif
#ifndef CLSMGR_INCLUDED
#include <clsmgr.h>
#endif
#ifndef GEO_INCLUDED
#include <geo.h>
#endif
#endif
Common `#`defines and typedefs

# Xlist Data Structure

A pointer to an xlist is a pointer to a private data structure. This pointer is passed to the xlist function to create, destroy, and manipulate xlists.

```c
typedef P_UNKNOWN P_XLIST;
```

# Xlist Flags

These flags are stored in the xlist. They are useful to store xlist specific data. flag0 through flag15 are reserved for GO internal use, while flag16 through flag31 is for client use. The only flag currently used indicates that XList2Text has been run on the xlist. This optimizes successive calls to this xlist filter, allowing it to return without running the filter. Running the filter a second time, because the flag is clear, is harmless.

```c
#define xflXList2Text flag0
```

# Element Types

These are the data types for elements of an xlist. An element contains a type, a data pointer and flags. For each data type, the data pointer varies.

```c
Enum16 (XTYPE) {
   xtNull,          // pData = null = 0
   xtBounds,        // pData = P_BDATA (clsXGesture, clsXText)
   xtGesture,       // pData = P_GDATA (clsXGesture)
   xtText,          // pData = P_STRING (clsXText, XList2Text)
   xtObject,        // pData = OBJECT
   xtBoundsX,       // pData = P_BDATA (screen relative)
   xtCharAttrs,     // pData = P_XLIST_CHAR_ATTRS (txtxlist.h)
   xtParaAttrs,     // pData = P_XLIST_PARA_ATTRS (txtxlist.h)
   xtTabs,          // pData = P_XLIST_TABS (txtxlist.h)
   xtCharPos,       // pData = TEXT INDEX
   xtTextList,      // pData = P_WORD_LIST (hwx)
   xtSpare1,        // pData =
   xtSpare2,        // pData =
   xtSpare3,        // pData =
   xtSpare4,        // pData =
   xtGeometric,     // pData = P_XGEO_DATA unused
   xtTextListEnd,   // pData = NULL (sPaper)
   xtTextWord,      // pData = P_XTEXT_WORD (xtext) (clsXtext, sPaper)
   xtStroke16,      // pData = P_SPAPER_STROKE_DATA (spaper)
   xtSpace,         // pData = U32 unused
   xtTeachData,     // pData = P_XTEACH_DATA (xteach)
   xtUID,           // pData = UID of the gesture object
   xtEmbedObject,   // pData = P_TEXT_EMBED_OBJECT (txtdata.h)
   xtExtended,      // pData = UID, client data
   xLastEntry       // last entry in the xtList
};
```

# Xlist Element

This data structure defines an element in an xlist. An xlist element contains some flags, a data type, and a pointer to some data. The allocation and type of data depend on both the flags and the data type of the element.

```c
typedef struct XLIST_ELEMENT {
   UI16 flags;            // Element flags. Mostly allocation information.
   XTYPE type;            // Type of data in pData
   P_UNKNOWN pData;       // Pointer to data of element
} XLIST_ELEMENT, *P_XLIST_ELEMENT;
```
Element Flags

These flags are stored in the XLIST_ELEMENT flags, and indicate information about the elements. They can be changed dynamically simply by accessing the xlist element. Other flags not used are reserved for future use.

Allocation flags

These flags indicate how to treat memory for the element. They indicate how the element will be freed when the xlist is freed, and how to allocate space for the element when duplicated via XListDup (cannot duplicate xfObject). Setting more than one of these flags will have unpredictable results, as these are mutually exclusive flags.

#define xfHeapAlloc flag0 // Allocated from the xlist heap
#define xfObject flag1 // Element is an object. Cannot duplicate
#define xfXList flag14 // Element is a P_XLIST

This flag indicates that the elements data is used elsewhere, and should not be freed when freeing the xlist. It will be the clients responsibility to free the data if he sets this flag.

#define xfExtracted flag15 // Set if the data is used elsewhere

Traversal function

This callback function is used to as a function template called on elements of the xlist when traversing the xlist. See XListTraverse for more details. This function takes an xlist, an xlist element, and a user defined data pointer.

Function Prototype typedef STATUS FunctionPtr (P_XPROC) (P_XLIST pXlist, P_XLIST_ELEMENT pElement, P_UNKNOWN pUserData);

Public Functions

XListNew

Creates a new xlist.

Returns STATUS.

Function Prototype STATUS PASCAL XListNew (OS_HEAP_ID heap, P_XLIST *ppXList); // In: heap to allocate the xlist // Out: Pointer to the P_XLIST

Comments Creates and allocates an xlist from the specified heap, using the heap to allocate space for the P_XLIST_ELEMENT entries in the list, and for XListAlloc.

XListFree

Frees an xlist and all its data.

Returns STATUS.

Function Prototype STATUS PASCAL XListFree (P_XLIST pXlist); // In: xlist to free

Comments Traverses the xlist elements and frees the data (unless the element has xfExtracted set). For each element, frees the memory appropriately by traversing the xlist with function XListFreeData.

See Also XListFreeData
XListGetFlags

Passes back the XList flags for the xlist.

Returns STATUS.

Function Prototype

```pascal
STATUS PASCAL XListGetFlags(
    P_XLIST pXList,       // In: xlist to get the flags from
    P_U32 pFlags);       // Out: pointer to the flags
```

Comments

flag0 through flag15 are reserved for GO internal use. flag16 through flag31 are for client use.

XListSetFlags

Sets the XList Flags.

Returns STATUS.

Function Prototype

```pascal
STATUS PASCAL XListSetFlags(
    P_XLIST pXList,       // In: xlist to set the flags from
    U32 flags);           // In: new flags to set
```

Comments

Sets the flags associated with the xlist. flag0 through flag15 are reserved for GO internal use. flag16 through flag31 are for client use.

XListMetrics

Passes back the number of entries and heap Id.

Returns STATUS.

Arguments

```c
typedef struct XLIST_METRICS {
    OS_HEAP_ID heap;
    U16 entries;
} XLIST_METRICS, *P_XLIST_METRICS;
```

Function Prototype

```pascal
STATUS PASCAL XListMetrics(
    P_XLIST pXList,       // In: xlist to get the metrics from
    P_XLIST_METRICS pMetrics); // Out: metrics of the xlist
```

Comments

Passes back the number of entries in the xlist, and the heap used to allocate xlist memory. Note that there is no corresponding 'set' metrics function, as dynamically changing the heap or count would have drastic side affects.

XListInsert

Creates a new element at the index'th location.

Returns STATUS.

Function Prototype

```pascal
STATUS PASCAL XListInsert(
    P_XLIST pXList,       // In: xlist to insert item into
    U16 index,            // In: index of location to insert at
    P_XLIST_ELEMENT pElem); // In: element to insert
```

Comments

Allocates space for and creates a P_XLIST_ELEMENT in the xlist at the specified location. If index >= entries, the element is appended to the end of the list. The element data pointer allocation and storage depends on the type of the element. The following example shows a client inserting a 7 character string into an xlist. The element type is xtText and the insertion at the beginning of the xlist:

```c
XLIST_ELEMENT elem;
    elem.type = xtText;
    elem.flags = xfHeapAlloc;
```
XListDelete
Delete the element at the index'th location.
Returns STATUS.

Function Prototype: STATUS PASCAL XListDelete(
P_XLIST pXList,       // In: xlist to delete item from
U16 index);            // In: index of item to delete

Comments: Delete the element at the specified location. This calls XListFreeData to free any memory taken by the element data pointer. Frees memory associated with storing the P_XLIST_ELEMENT in the xlist.

See Also: XListFreeData

XListTraverse
Iterates across the list of elements.
Returns STATUS.

Function Prototype: STATUS PASCAL XListTraverse(
P_XLIST pXList,       // In: xlist to traverse
P_XPROC pProc,         // In: call back function to call for each element
P_UNKNOWN pUserData);  // In/Out: User defined data pointer

Comments: Iterates across the elements in the xlist. A callback function (pProc) is handed to this function, and is called for each element passing in the element and a client pointer as defined in P_XPROC. If any call to pProc returns anything but stsOK, the traversal is terminated and the status code returned. Nested traversals are allowed and supported.

See Also: XListIndex

XListIndex
Passes back the current traversal index.
Returns STATUS.

Function Prototype: STATUS PASCAL XListIndex(
P_XLIST pXList,       // In: pointer to xlist
U16 pIndex);          // Out: current index

Comments: Passes back the index for the current traversal. If no traversal is taking place, returns 0. Note that if nested traversals are taking place, the index of the current traversal will be returned. Once the sub-traversal is completed, the parent traversals index is restored and returned appropriately via calls to XListIndex.

See Also: XListTraverse

XListSet
Stores the copy of the index'th element.
Returns STATUS.

Function Prototype: STATUS PASCAL XListSet(
P_XLIST pXList,       // In: xlist pointer
U16 index,            // In: index of element
P_XLIST_ELEMENT pPtr); // In: new element to store at location
Stores the passed in element as the element in the specified location. If index is > number of entries, will
store in the last item in the list. Care should be taken, as the old item stored in that location is not freed
and is the clients responsibility. Useful only if changing an entire item in the xlist. Rarely used.

XListGet

Passes back a copy of the index' th element.

Returns STATUS.

Function Prototype

STATUS PASCAL XListGet(
P_XLIST pXList,    // In: xlist pointer
UI6 index,         // In: index of element
P_XLIST_ELEMENT pPtr);    // Out: Copy of element data.

Comments

Passes back a copy of the index' th element. The element, data type, and data pointer will be copied.
Hence the data pointer is a direct pointer to the data.

XListGetPtr

Passes back a pointer to the index' th element.

Returns STATUS.

Function Prototype

STATUS PASCAL XListGetPtr(
P_XLIST pXList,
UI6 index,
P_XLIST_ELEMENT *ppPtr);

Comments

Passes back a pointer to the index' th element in the xlist. Extreme care should be taken when accessing
this pointer, as it is the pointer stored in the xlist. Useful only if the client wishes to change some
information about an existing item in the xlist. Rarely used. Note that the data pointer field is the same
returned by XListGet.

See Also

XListGet

XListAlloc

Allocate some memory from the XList heap.

Returns STATUS.

Function Prototype

STATUS PASCAL XListAlloc(
P_XLIST pXList,    // In: xlist pointer
SIZEOF size,       // In: size of the requested allocation
P_UNKNOWN pMem);    // Out: pointer to the allocated memory

Comments

Allocates memory off of the xlist heap. Typically used to allocate space for the data pointer of an element
that has xfHeapAlloc set. Space for such an element data pointer will be freed in XListFreeData, called
when the xlist is freed via XListFree, or when the item is deleted via XListDelete. Other memory can be
allocated using this function, although it is the clients responsibility to ensure that it is freed.

See Also

XListFreeData

XListFreeData

Releases the data with the given entry.

Returns STATUS.
null
typedef struct X2GESTURE {
  U32 msg;       // gesture type
  RECT32 bounds; // gesture bounding box
  XY32 hotPoint; // gesture hot point
} X2GESTURE, *P_X2GESTURE;

FUNCTION Prototype
STATUS PASCAL XList2Gesture(
P_XLIST pXList,    // In: xlist to run filter on
P_X2GESTURE pData); // Out: converted data structure

Comments
Given an xlist containing xtBounds followed by xtGesture, (the xlist typically returned by the
clsXGesture translator after completed translation), this function extracts the useful information and
stores it in a standard c data structure. This function is used internally in gWin to convert the gesture
translator data structure into a more useful form.

See Also
gwin.h

XList2StringLength
Passes back the length of the string that XList2String will need.
Returns STATUS.

FUNCTION Prototype
STATUS PASCAL XList2StringLength(
P_XLIST pXList,
P_U16 pLength);

Comments
Computes the necessary length of a string that XList2String will need to copy a string. Includes space for
the terminating null character.

XList2String
Extracts the text information from an xlist.
Returns STATUS.

Arguments
typedef struct X2STRING {
  U16 count;  // In: buffer size
  P_CHAR pString; // In: pointer to the buffer
} X2STRING, *P_X2STRING;

FUNCTION Prototype
STATUS PASCAL XList2String(
P_XLIST pXList,
P_X2STRING pData); // In: xlist to process

Comments
Converts an xtBounds/xtText xlist into a string. Clips the returned string at the passed in count. This
string includes a null terminating character. The function takes an xlist of the form:
[xtBounds [xtText]]
and converts it into a string. As an example, suppose the xlist contains:
xtBounds1 xtText1 xtText2 xtText3 xtBounds2 xtText4 xtText5.
This is converted into:
xtText1xtText2xtText3xtText4xtText5

More typically, this function called on an xlist that has had adjacent xtText entries merged by
XList2Text. Typical usage is during processing of an xlist returned from msgXlateGetData. Here the
client simply wants to know the string returned, so he will call XList2Text, XList2StringLength (unless
he knows how big the string will be), and XList2String to get the string.

See Also
XList2Text.h
**Debugging Functions**

**XListDump**

Debugging interface for displaying an xlist in the debug log.

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED XListDump(
    P_XLIST pXList);        // In: array header
```

Comments

When called on an xlist, traverses the elements and displays useful information about the xlist in the debug log. It displays this information by calling a display routine that is dependent on the type of the element. A display routine can be registered for an element type using XListDumpSetup. If no display routine has been provided for an element type, it will display the generic information for the element consisting of the type, the flags, and the element data pointer.

See Also

XListDumpSetup

**XListDumpSetup**

Sets the xlist debug log display routine by type.

Returns STATUS.

Function Prototype

```c
STATUS EXPORTED XListDumpSetup(
    XTYPE type,         // In: xtype to bind this procedure to
    P_XPROC pProc,      // In: function to be called when dumping
    U32 data);          // In: type specific data passed to pProc in traversal
```

Comments

Called to register display routines for xlist element types with the xlist. This display routine will be called when the particular element type traversed when calling XListDump.

See Also

XListDump
This file contains the API for clsXShape, a skeletal class designed to be subclassed by particular shape recognition engines. In particular, the GOWrite shape recognizer, clsCTShape, is a subclass of clsXShape.

clsXShape inherits from clsOpenServiceObject.

ifndef XSHAPE_INCLUDED
#define XSHAPE_INCLUDED
ifndef GO_INCLUDED
#include <go.h>
#endif
ifndef CLSMGR_INCLUDED
#include <clsmgr.h>
#endif
ifndef OSHEAP_INCLUDED
#include <osheap.h>
#endif
ifndef OPENSERV_INCLUDED
#include <openserv.h>
#endif

Terminology change

NEW NAME (use these) OLD NAME (avoid using these, from uid.h)
#define theShapeEngines theHWXEngines
#define theInstalledShapeProfiles theInstalledHWXProtos
#define clsShapeEngineService clsHWXEngineService
#define clsShapeProfileInstallMgr clsHWXProtoInstallMgr
#define msgShapeSvcCurrentChanged msgHWXSvcCurrentChanged
#define SHAPE_svc_CURRENT_CHANGED HWX_svc_CURRENT_CHANGED
#define P_SHAPE_svc_CURRENT_CHANGED P_HWX_svc_CURRENT_CHANGED

Common #defines and typedefs

#define xsMaxCharList 20 // largest allowable matchArraySize for msgXShapeRecognize
#define xsMaxPath 4 // most strokes allowable to send to msgXShapeRecognize
#define xsMinMatchScore minS16 // worst possible score for translation
#define xsDigitizerResolution 254 // temporary hack. Eventually variable

The basic types of shape profile ("resource") data stored in files. This refers to the "alphabet" which the resource is able to recognize, not the use to which the recognized value will be put. In particular a text resource may be used as part of the process of recognizing gestures, since some gestures are upper case letters.

Enum16(XS_RESOURCE_TYPE) {
    xsResText = 0, // alphabetic (ascii)
    xsResReserved = 1, // reserved for use by GO
    xsResGesture = 2 // gestures
};
The types of data structure used to return information from msgXShapeRecognize.

```c
Enum16 (XS_MATCH_TYPE) {
    xsMatchAscii = 1, // uses XS_ASCII_MATCH data structure
    xsMatchGesture = 2, // uses XS_GESTURE_MATCH data structure
    xsMatchInternal = 3, // uses subclass-specific data structure
    xsMatchInternal2 = 4, // uses alternate subclass-specific data structure
};
```

Eight principal compass directions for straight lines.

```c
Enum16 (XS_DIRECTION) {
    xsRight = 0,
    xsUpRight = 1,
    xsUp = 2,
    xsUpLeft = 3,
    xsLeft = 4,
    xsDownLeft = 5,
    xsDown = 6,
    xsDownRight = 7,
    // Special indicators
    xsAllDirections = 8, // used internally
    xsDirEndMark = 9 // marks end of array of directions
};
```

```c
#define xsNumDirections (8)
#define XSNextDirectionCCW(d) (((d) + 1) & 7)
#define XSNextDirectionCW(d) (( (d) - 1) & 7)
#define XSOppositeDirection(d) ((d) ^ 4)
#define XSDeltaDirection (start, end) (((end) - (start)) & 7)
#define XSDeltaDirectionAdd(start, delta) (((start) + (delta)) & 7)
```

The following structures capture basic information about strokes (a stroke being a sequence of points passed through by the pen). See msgXShapeStrokePreview for further details.

```c
typedef struct XS_OCTAGON {
    S16 limit[xsNumDirections]; // max projection in each direction
} XS_OCTAGON, *P_XS_OCTAGON;
```

Data structure for returning information about recognition of an ascii character from msgXShapeRecognize.

```c
typedef struct XS_ASCII.MATCH {
    S16 score; // "penalty" for the match
    U8 character; // ascii code of proposed translation
    U8 segmentOffset; // reserved for GO. msgXShapeRecognize should set to 0
} XS_ASCII_MATCH, *P_XS_ASCII_MATCH;
```

Data structure for returning information about recognition of a gesture from msgXShapeRecognize.

```c
typedef struct XS_GESTURE_MATCH {
    S16 score; // "penalty" for the match
    U32 gestureId; // proposed translation (id codes defined in xgesture.h)
    POINT hotPoint; // coordinates of target point of the gesture
} XS_GESTURE_MATCH, *P_XS_GESTURE_MATCH;
```

Data structure for returning information about recognition of a straight line or a dot. Used by the GO context level processing to aid in segmentation. These scores are calculated by the GO context engine; they should not be calculated or used by 3rd party shape engine developers.

```c
typedef struct XS_LD_MATCH {
    S16 dotScore; // Score for a dot.
    S16 lineScore02; // Score for horiz/vert line
    S16 lineScore13; // Score for forw/backw slanted line
} XS_LD_MATCH, *P_XS_LD_MATCH;
```
The XS_STROKE record holds information pertinent to a single stroke. PenPoint computes all fields of this structure except pData and numData. The latter two are (optionally) computed by the shape matching engine. They are intended to hold whatever information the shape matcher wishes to extract from a single individual stroke.

define struct XS_STROKE {
    struct XS_STROKE *pNextStroke; /* pointer to next stroke */
    struct XS_STROKE *pPrevStroke; /* pointer to previous stroke */
    U16 strokeId; /* a unique identifier of this stroke */
    struct POINT *pPoint; /* arr of digitizer points (pendown to penup) */
    U16 numPoints; /* number of digitizer points (excl. end marker) */
    XS_OCTAGON bound; /* bounds of this stroke */
    P_UNKNOWN pData; /* subclass-specific data extracted from stroke */
    U16 numData; /* subclass-specific counter for pData */
    XS_ASCII MATCH asciiMatch[xsMaxCharList]; /* cached results of single stroke recog. */
    XS_LD_MATCH ldMatch; /* scores for line and dot matches */
} XS_STROKE, *P_XS_STROKE;

### Initialization Messages

**msgNewDefaults:**

Initializes the XSHAPE_NEW structure to default values.

Takes P_XSHAPE_NEW, returns STATUS. Category: class message.

**Comments**

Zeros out pArgs->xshape and sets

- pArgs->xshape.resType = xsResText;
- pArgs->xshape.resolution = xsDigitizerResolution;

**msgNew:**

Creates a new shape matching object.

Takes P_XSHAPE_NEW, returns STATUS. Category: class message.

**Arguments**

define struct XSHAPE_NEW_ONLY {
    P_UNKNOWN pProfile; /* ptr to data in subclass specific format */
    U16 numProfile; /* how many records (e.g. if pProfile pts to array) */
    XS_RESOURCE_TYPE resType; /* type of profile: xsResText, resGesture */
    OBJECT profDirHandle; /* handle to directory where profile resides */
    S16 resolution; /* digitizer granularity (dots per inch) */
    S16 charConstraints; /* flags to set restricted character sets */
    S16 reserved16; /* pad for now */
    U32 reserved[9]; /* may be used in future */
} XSHAPE_NEW_ONLY, *P_XSHAPE_NEW_ONLY;

define struct XSHAPE_NEW {
    openServiceObjectNewFields XSHAPE_NEW_ONLY xshape;
} XSHAPE_NEW, *P_XSHAPE_NEW;

**Comments**

This message is sent to the xshape subclass by the service manager when someone has requested a new shape matching engine. The service manager has filled in all of the xshape fields. In responding to this message it is merely necessary to copy the fields of xshape_new data into the new object's private instance data.
msgFree:
Destroys the object, releasing any memory associated with the translation.
Takes pNull, returns STATUS.

Comments
If any heaps were created in response to msgNew, this is the time to destroy them. NOTE: This is
NOT the place to free memory occupied by the data pointed to by pProfile. That memory was allocated
by your service class in response to msgXShapeSvcCurrentChanged and should only be free in response
to the next occurrence of the same message.

Control Messages

msgXShapeStrokePreview:
Computes and stores data relating to a single stroke
Takes P_XSHAPE_STROKE_PREVIEW, returns STATUS.

#define msgXShapeStrokePreview MakeMsg(clsXShape, 3)

typedef struct XSHAPE_STROKE_PREVIEW {
    P_XSHAPE pFirstStroke;  // IN: pointer to stroke record
} XSHAPE_STROKE_PREVIEW, *P_XSHAPE_STROKE_PREVIEW;

Arguments
This msg gives the class the opportunity to extract and store information that applies to an individual
stroke, not to the combined set of strokes that form a character. (The latter extraction should occur
totally within the method for msgXShapeRecognize.)

This message is sent by the input system as part of its background processing of strokes as they are
entered by the user. Background processing allows the system to produce the final translation more
quickly after the user taps the translate button.

Furthermore, a single stroke may be submitted more than once to the shape engine for recognition, as
the context engine tries out different combinations of strokes searching for the best segmentation. Thus
the stroke will be "previewed" only once, but may appear in several different combinations of strokes
submitted for recognition.

The subclass is responsible for defining the format and managing the memory that contains the
information extracted in the preview process. The pointer pData in the XS_STROKE record should be set
to point to this data. The field numData of the XS_STROKE record is available to record the number of
records pointed to by pData (if it's an array).

Memory for *pData should be allocated from a local heap whose heapId has been stored in the instance
data for the object. The heap should be created in response to msgNew and destroyed in response to
msgFree.

The method for msgXShapeStrokePreview may assume that the following fields of the XS_STROKE
record have already been calculated:

strokeId
bound
pPoint
numPoints

All other fields should be ignored.
The `strokeId` uniquely identifies the stroke (as far as this object is concerned).

The bound implicitly defines the bounding octagon for the stroke by recording for each of the 8 directions the maximum of the projections of all points in the stroke in that direction. Given a point P and a direction d, the projection of P in direction d is defined to be the x-coordinate of P in a coordinate system which is rotated d*45 degrees counterclockwise from the base coordinate system. Computationally this works out to:

\[
\begin{align*}
    x & \quad \text{if } d == 0 \quad (\text{xRight}) \\
    (x+y)/r & \quad \text{if } d == 1 \quad (\text{xUpRight}) \\
    y & \quad \text{if } d == 2 \quad (\text{xUp}) \\
    (-x+y)/r & \quad \text{if } d == 3 \quad (\text{xUpLeft}) \\
    -x & \quad \text{if } d == 4 \quad (\text{xLeft}) \\
    (-x-y)/r & \quad \text{if } d == 5 \quad (\text{xDownLeft}) \\
    -y & \quad \text{if } d == 6 \quad (\text{xDown}) \\
    (x-y)/r & \quad \text{if } d == 7 \quad (\text{xDownRight})
\end{align*}
\]

where r is \(\sqrt{2}\). Division by r is simulated in integer arithmetic as multiplication by 5 followed by (integer) division by 7.

From the bound the method can calculate other quantities as needed using the following formulas:

\[
\begin{align*}
    \text{baseline} & = -\text{bound.limit[xsDown]}; \\
    & \quad \text{// because } -\max{-y} = \min{y} \\
    \text{height} & = \text{bound.limit[xsUp]} + \text{bound.limit[xsDown]}; \\
    & \quad \text{// because } \max{y} + \max{-y} = \max{y} - \min{y} \\
    \text{width} & = \text{bound.limit[xsRight]} + \text{bound.limit[xsLeft]}; \\
    & \quad \text{// because } \max{x} + \max{-x} = \max{x} - \min{x}
\end{align*}
\]

`pPoints` points to an array of digitizer points, terminated with a record with coordinates (minS16, minS16). `numPoints` tells how many points are in the array, EXCLUDING the terminating record. (So `numPoints` can also be taken as the index of the terminating record.) The 0th record corresponds to `penDown`, the (numPoints-1)th record to `penUp`.

**msgXShapeRecognize:**

Provide possible translations for a set of strokes.

Takes `P_XSHAPE_RECOGNIZE` and returns `STATUS`.

```c
#define msgXShape_RECOGNIZE MakeMsg(clsXShape, 5)
```

**Typedef**

```c
typedef struct XSHAPE_RECOGNIZE {
    P_XS_STROKE pFirstStroke; // IN: linked list of (at most xsMaxPath) strokes
    XS_MATCH_TYPE matchType;  // IN: type of record in output array (matchAscii
    // for XS_ASCII_MATCH, xsMatchGesture for XS_GESTURE_MATCH)
    U16 matchArraySize;       // IN: number of records in output array (at
    // most xsMaxCharList)
    P_UNKNOWN pMatchResults; // IN: ptr to output array
} XSHAPE_RECOGNIZE, *P_XSHAPE_RECOGNIZE;
```

**Arguments**

- `P_XS_STROKE` `pFirstStroke`: Linked list of (at most `xsMaxPath`) strokes
- `XS_MATCH_TYPE` `matchType`: Type of record in output array (matchAscii, `XS_ASCII_MATCH`, `xsMatchGesture`, `XS_GESTURE_MATCH`)
- `U16` `matchArraySize`: Number of records in output array (at most `xsMaxCharList`)
- `P_UNKNOWN` `pMatchResults`: Pointer to output array

**Comments**

The set of strokes (given as a linked list) is a combination which the context level is testing to see if it represents a single character or gesture. The job of the shape engine is to return an array of the most likely translations (or "matches") together with a weight (or "score") for each of them. If the strokes do not match any of the forms which the shape engine is designed to recognize, it should return an empty array (i.e. the first record should be marked with score `xsMinMatchScore`).

Scores are 0 or negative, with 0 representing the best possible match. Scores below 0 represent progressively worse matches. The range is open ended below, but generally the scores for the most unlikely but still remotely possible translations should fall in the -80 to -120 range, or very occasionally below -120.
Different recognition technologies may have radically different approaches for arriving at scores and correspondingly different models of what the scores mean. One technology may assign scores as a measure of the amount of deviation from an ideal form, a kind of Euclidean distance function. Another technology may arrive at scores through a process of statistical tests, so the score would represent the amount of statistical evidence there is against a particular translation. Yet another technology may compute probabilities.

In order to deal uniformly with a variety of different shape recognition technologies, the context level processor requires that the scores reported by the shape engine be scaled or calibrated according to the following guidelines:

1. "Reasonable" scores should fall roughly in the range 0 to -100.
2. "SCORES SHOULD BE SCALED LOGARITHMICALLY," with every 10 point drop in score representing roughly a 50% reduction in confidence/probability/proximity etc. Thus for example a translation with a score of -50 is 1/8 as "good" (or 1/8 as "likely" or 8 times as "far" from being perfect) as a translation with a score of -20.
3. The score for each translation should reflect the confidence in that translation only. It should NOT be influenced by the confidence in any other translation. In particular, a high score for one translation does not preclude a high score for another translation. For example 'o' and 'O' may both score high (even perfect). In this way, scores need not behave like probabilities: they do not represent slices from a fixed pie.
4. Similarly, there is no requirement that the scores "add up" to a fixed total. For a particular sample, all of the scores may be poor, or the recognizer may even send back no translations. The context engine is depending on this fact in order to be able to use the shape engine to help it choose the correct character segmentation.
5. Scores should not be "tainted" by knowledge of character frequency in English or any other linguistic considerations. It is the job of the context level processing to take linguistic information into account. The shape engine must consider all characters as a priori equally likely, otherwise the bias for common characters in text will be duplicated at both levels, resulting in unwanted effects.

**msgXShapeShapeCompatible:**

Checks the possibility of translating the strokes as the char

Takes P_XSHAPE_COMPATIBLE, returns STATUS.

```c
#define msgXShapeShapeCompatible MakeMsg(clsXShape, 6)
```

**Arguments**

```
typedef struct XSHAPE_COMPATIBLE {
    P_XS_STROKE pFirstStroke; // IN: linked list of strokes
    US character;              // IN: desired translation for the strokes
    US strokeCount;           // IN: how many strokes in the linked list
    BOOLEAN compatible;       // OUT: is translation a priori possible
} XSHAPE_COMPATIBLE, *P_XSHAPE_COMPATIBLE;
```

**Comments**

Sees if there is anything about the strokes that absolutely rules out the letter as a translation. For example, some shape matchers may rule out certain translations based on the number of strokes in the list.

This message is sent by the context level only when it has been instructed to allow the dictionary (spelling) or a template to propose characters when the shape level is stuck. The context level makes this check just be sure that there is some remote possibility that the strokes do represent the proposed character before allowing the dictionary or template to propose it.
Training Messages

msgXShapeShapeEvaluate:
Checks how well the shape matcher translates the character.
Takes P_XTEACH_DATA, returns STATUS.
#define msgXShapeShapeEvaluate MakeMsg(clsXShape, 7)

Comments: Reports back how well the current engine translates the strokes, knowing what the correct translation is. Does NOT cause the engine to learn the new shape if it is translated poorly.

msgXShapeShapeLearn:
Forces shape matcher to learn new shape.
Takes P_XTEACH_DATA, returns STATUS.
#define msgXShapeShapeLearn MakeMsg(clsXShape, 8)

Comments: Usually invoked based on the results from msgXShapeShapeEvaluate.
#ifndef XTEACH_INCLUDED
#define XTEACH_INCLUDED
#endif

#include <go.h>
#include <uid.h>
#include <geo.h>

Common #defines and typedefs

typedef enum {
    xteachNoMatch,  // no matches
    xteachSingular,  // matches only the correct character
    xteachSuperior,  // matches the correct character best
    xteachEquivalent,  // matches the correct character and an
                      // incorrect character equally well
    xteachSecondary,  // matches an incorrect character best,
                      // but also matches the correct character
                      // same as secondary, except that the best
                      // match is marginal
    xteachInferior,  // matches only incorrect characters marginally
    xteachNotProposed,
    xteachMisRecognized,
    xteachEvaluateFailed,  // execute results
    xteachOK,
    xteachGeometricUpdated,
    xteachPrototypeAdded,
    xteachOutOfMem,
    xteachPrototypeRemoved,
    xteachPrototypeDowngraded,
    xteachAbort,
    xteachExecuteFailed
} TEACH_STATUS, *P_TEACH_STATUS;

#define xteachMaxConflict (64)
#define xteachMaxCharConflict (8)

typedef struct XTEACH_DATA {
    U32 id;  // character/symbol id
    TEACH_STATUS status;  // evaluation results
    U16 conflictCount;  // number of conflicting protos
    CHAR conflicts[xteachMaxConflict];  // conflicting characters
    U32 conflictId[8];  // indices of conflicting protos
    S16 conflictPenalty;  // penalty to assess
    P_UNKNOWN pFirstStroke;  // pointer to first stroke
    P_UNKNOWN pContext;  // pointer to HWX context
    XY32 target;  // coordinate of hot point target
    CHAR hotPointPath;
    CHAR hotPointExtrema;
} XTEACH_DATA, *P_XTEACH_DATA;
**Messages**

**msgNewDefaults:**
Sets default values for a new Teach translation object.
Takes P_XLATE_NEW, returns STATUS.

**msgNew:**
Creates a new Teach translation object.
Takes P_XLATE_NEW, returns STATUS.

**msgXlateData:**
Returns Teach results.
Takes P_XLATE_DATA, returns STATUS.

```c
typedef struct TEACH_DATA {
    TEACH_STATUS status;  // required action
    CHAR charConflicts[xteachMaxCharConflict]; // conflicting characters
} TEACH_DATA, *P_TEACH_DATA;
```

**msgXTeachSetId:**
Establishes expected translation results.
Takes P_CHAR, returns STATUS.

```c
#define msgXTeachSetId MakeMsg(clsXTeach, 0x01)
```

**msgXTeachExecute:**
Executes teaching per TEACH_STATUS.
Takes P_XLIST, returns STATUS.

```c
#define msgXTeachExecute MakeMsg(clsXTeach, 0x02)
```

**msgXTeachEvaluationGet:**
Reads evaluation data.
Takes P_XLATE_DATA, returns STATUS.

```c
#define msgXTeachEvaluationGet MakeMsg(clsXTeach, 0x03)
```

**msgXTeachSetTarget:**
Sets the target coordinates for the hot point.
Takes P_XY32, returns STATUS.

```c
#define msgXTeachSetTarget MakeMsg(clsXTeach, 0x05)
```
**Notification Messages**

**msgXTeachCompleted:**
Signals completion of training.
Takes P_LIST, returns STATUS.

```c
#define msgXTeachCompleted MakeMsg(clsXTeach, 0x04)
```

**Comments**
This message is sent to all observers of the translation object following successful completion of the method for msgXTeachExecute.
XTEMPLT.H

Translation Template Specifications for input fields

#ifndef XTEMPLT_INCLUDED
#define XTEMPLT_INCLUDED

/DS0010 Compilation: print ASCII input and hex-address of result.
/DS0020 Choices: print Hex address and ASCII list of choices plus count.

#ifndef GO_INCLUDED
#include <go.h>
#endif
#ifndef OS_INCLUDED
#include <os.h>
#endif
#ifndef XLATE_INCLUDED
#include <xlate.h>
#endif

Definitions

maxXTemplateXlateChoices is the number of different symbols that are in a CharList template.
maxXtmPictureLength is the longest template may be.

#define maxXTemplateXlateChoices 128 // Alphabet Size
#define maxXtmPictureLength 128 // Picture string length limit

Common Typedefs

Template Types

Templates are used to constrain handwritten input in order to translation accuracy. For example, if a field can only contain digits, constraining the input for that field to only digits that the letters '0', '1', and 'Z', are never seen for the '0', '1', and '2'. There are several different ways to input, each of which corresponds to a different template

Enum16(TEMPLATE_TYPE) {
    xtmTypeNone,           // no constraints
    xtmTypeGesture,        // limited to known gestures
    xtmTypeShape,          // limited to known shapes (NOT IMPLEMENTED)
    xtmTypeCharList,       // limited to a set of characters
    xtmTypeWordList,       // limited to a set of words
    xtmTypePicture,        // described by a picture language
    xtmTypeRegEx,          // described by a regular expression (NOT IMPLEMENTED)
    xtmTypeTrie,           // precompiled
};
Template Modes

A template may be interpreted in a variety of special modes. In general, modes describe circumstances under which incomplete input will be the same as complete input.

```c
Enum16(XTEMPLATE_MODE) {
    xtmModeDefault = 0,       // No special modes
    xtmModePrefixOK = flag0,  // input matching a prefix of the template is
    xtmModeLoopBackOK = flag1, // considered to match the template.
    xtmModeCoerced = flag2,   // the template is considered to repeat over
                             // and over
                             // Input should be coerced to match the
                             // template, even if it doesn't match exactly
                             // Only meaningful for xtmTypeWordList templates.
};
```

Template Header

Every template is a single allocated block of memory containing no pointers. The template header contains information about the template, including what's needed to file a template.

```c
typedef struct XTEMPLATE_TRIE_HEADER {
    U16 xtmTrieLength;
    U16 xtmTrieRevision;
    XTEMPLATE_TYPE xtmTrieType;
    XTEMPLATE_MODE xtmTrieMode;
} XTEMPLATE_TRIE_HEADER, *P_XTEMPLATE_TRIE_HEADER;
```

Template Metrics Structure

This structure is returned via the XTemplateGetMetrics subroutine, below. The major uses of this structure are to get to the template header in order to get the template length so can be filed, and to get access to the original template string.

```c
typedef struct XTEMPLATE_METRICS {
    P_XTEMPLATE_TRIE_HEADER pXtmHeader; // Template len, rev, etc.
    P_CHAR pXtmString;                  // Original string, NULL for word list or gesture.
    P_UNKNOWN pXtmTrieBase;             // Base of compressed TRIE structure
    U16 xtmTrieBaseLen;                 // Size of the compressed region
} XTEMPLATE_METRICS, *P_XTEMPLATE_METRICS;
```

Functions

**XTemplateCompile**

Given a type and an ASCII template representation, build a template structure.

Returns STATUS.

```c
typedef struct XTM_ARGS {
    XTEMPLATE_TYPE xtmType;       // What kind of template?
    XTEMPLATE_MODE xtmMode;       // What special modes?
    P_UNKNOWN pXtmData;           // ascii template
} XTM_ARGS, *P_XTM_ARGS;
```

Function Prototype

```
STATUS EXPORTED XTemplateCompile(
    P_XTM_ARGS pXtmArgs,          // Xtemplate Arguments
    OS_HEAP_ID heap,              // heap to use
    PP_UNKNOWN ppXtmDigested      // Out: compiled template
);
```
The currently implemented types have the following meanings:

**xtmTypeNone**  
pXtmData is unused. This is the same as having no template at all.

**xtmTypeGesture**  
pXtmData points to an XTEMPLATE_GESTURE_LIST.

**xtmTypeCharList**  
pXtmData contains a list of valid characters.

**xtmTypeWordList**  
pXtmData contains a list of all the different words that are legal in this field. This should be a PP_STRING pointing to a list of pointers to the words. Each word is a normal null-terminated string and the pointer list must be terminated with a Nil(P_STRING).

**xtmTypePicture**  
pXtmData contains a list of all the picture strings that are valid in this field. A picture string contains any of the following characters:

- 9: input must be a digit (0-9)
- a: input must be alphabetic
- A: input must be upper-case alphabetic
- n: input must be alphanumeric
- N: input must be upper-case alphanumeric
- x: input may be anything

- [: introduces a list of characters, Unix-style. [abc] is a single character position which must contain 'a', 'b', or 'c'. [a-m] matches any letter 'a' through 'm'. [a-m] matches any of 'a', '-', or 'm'.
- \: literal escape. Input must match next character. (Only needed to escape the above special characters).

For example, a modern California licence plate looks like this:

```
#AAA###
```

To include older forms of California plates, we might use:

```
#AAA###
###AAA
AAA###
```

either \n or tab separated. N.B. Multiple picture strings will not be supported in the first release.

A Social Security Number (with mandatory hyphens) would be coded like this:

```
###-##-####
```

Pictures currently can't be used for variable length data.

This special structure is used for xtmTypeGesture templates.

```c
typedef struct XTEMPLATE_GESTURE_LIST {
    U32 count;  // number of gestures in the list
    P MESSAGE pGestures;  // pointer to array of allowed gestures
} XTEMPLATE_GESTURE_LIST, * P_XTEMPLATE_GESTURE_LIST;
```

Space is allocated as required.
Basic XTemplate Arguments

XTemplateGetMetrics
Given a pointer to a translation template, extract various salient facts about it and return them.
Returns STATUS.

Function Prototype

```
STATUS EXPORTED XTemplateGetMetrics(
    P_UNKNOWN pXTemplate, // Template to extract the metrics of
    XTEMPLATE_METRICS pXtmMetrics // Out: metrics of the template
);
```

Can fail if the template version is too far out of date.

XTemplateSetMode
Change the mode in an already-created XTemplate.
Returns STATUS.

Function Prototype

```
STATUS EXPORTED XTemplateSetMode(
    P_UNKNOWN pXTemplate, // Compressed Template
    XTEMPLATE_MODE xtmMode // New mode
);
```

Changing xtmPrefixOK or xtmLoopBackOK may have no effect.

XTemplateFree
Free an existing Template.
Returns STATUS.

Function Prototype

```
STATUS EXPORTED XTemplateFree(
    P_UNKNOWN pXtmDigested // compiled template ptr.
);
```

Checks for pNull and just returns stsOK

XTemplateWordListSort
Given a pointer to a list of pointers to strings, sort the list of pointers so the strings appear in alphabatical order.
Returns void.

Function Prototype

```
void EXPORTED XTemplateWordListSort(
    PP_CHAR ppStringBase // compiled template
);
```

Last pointer in list must be Nil(P_STRING)

XTemplateCheckWord
Check if a word is in a template.
Returns BOOLEAN.

Function Prototype

```
BOOLEAN EXPORTED XTemplateCheckWord(
    P_UNKNOWN pXtmData, // compiled template
    P_CHAR pWord // Word to check
);
```
**XTemplateCheckGesture**

Check if a gesture is in a template.

Returns BOOLEAN.

Function Prototype: `BOOLEAN EXPORTED XTemplateCheckGesture(
P_UNKNOWN pXtmData, // compiled template
MESSAGE gesture // gesture to test
);`

**XTemplateAddWord**

Add a word to a wordlist template.

Returns STATUS.

Function Prototype: `STATUS EXPORTED XTemplateAddWord(
PP_UNKNOWN ppXtmData, // In/Out: compiled template
P_CHAR pWord, // Word to add
OS_HEAP_ID heap // heap to use
);`

**XTemplateDeleteWord**

Delete a word from a wordlist template.

Returns STATUS.

Function Prototype: `STATUS EXPORTED XTemplateDeleteWord(
PP_UNKNOWN ppXtmData, // In/Out: compiled template
P_CHAR pWord, // Word to add
OS_HEAP_ID heap // heap to use
);`

**XTemplInit**

DLL Initialization routine.

Returns STATUS.

`STATUS EXPORTED XTemplInit(void);`

Included for compatibility; not to be called by developers.
XTEXT.H

Defines the API for clsXText

clsXText inherits from clsXtract.

#ifndef XTEXT_INCLUDED
#define XTEXT_INCLUDED
#endif
#ifndef GO_INCLUDED
#include <go.h>
#endif
#ifndef UID_INCLUDED
#include <uid.h>
#endif
#ifndef XLATE_INCLUDED
#include <xlate.h>
#endif

Common #defines and typedefs

typedef struct XTEXT_WORD {
  RECT32 bounds; // bounding box
  CHAR str[1];   // text string, 0 terminated
} XTEXT_WORD, *P_XTEXT_WORD;

Messages

msgNewDefaults:
Fills in default values to XLATE_NEW structure.
Takes P_XLATE_NEW, returns STATUS..

Comments
All fields are set to 0 except the following hwxFlags which are turned ON:

  alphaNumericEnable
  punctuationEnable
  verticalEnable
  caseEnable
  smartCaseDisable

is also on (i.e. there will be NO "smart case" postprocessing to correct the capitalization of letters). The exception is that if the writer is an all caps writer (as determined by the global preference setting) then the default setting is OFF (i.e. there WILL be smart case postprocessing).
**msgNew:**

Creates a new Text translation object.
Takes P_XLATE_NEW, returns STATUS.

**msgXTextGetXList:**

Convert data to XList.
Takes P_XLATE_DATA, returns STATUS.

```
define msgXTextGetXList MakeMsg(clsXText, 0x01)
```

**msgXTextWordList:**

subclass opportunity to alter word list/disambig Called during the disambiguation extraction pass.
Takes P_WORD_LIST, returns STATUS.

```
define msgXTextWordList MakeMsg(clsXText, 0x02)
```

**msgXTextComplete:**

Hook for subclasses to postprocess translation results from clsXText
Takes P_XLIST, returns STATUS.

```
define msgXTextComplete MakeMsg(clsXText, 0x81)
```

**msgXTextNewLine:**

Indicates the start of a new line to subclasses.
Takes P_UNKNOWN, returns STATUS.

```
define msgXTextNewLine MakeMsg(clsXText, 0x82)
```

**msgXTextModLine:**

Indicates a modification of a line to subclasses.
Takes P_UNKNOWN, returns STATUS.

```
define msgXTextModLine MakeMsg(clsXText, 0x83)
```
This file contains part of the API definition for clsXtract. For the remainder see xlate.h.

clsXtract inherits from clsObject.

```c
#ifndef XTRACT_INCLUDED
#define XTRACT_INCLUDED
#endif

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

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

### Messages

**msgSave:**

Saves an extraction object.

Takes P_OBJ_SAVE, returns STATUS.

Comments

Writes the instance data for this object out to a file.

**msgRestore:**

Restores an extraction object.

Takes P_OBJ_RESTORE, returns STATUS.

Comments

Reads back in from a file the instance data for an extraction object and recreates the object.

### Initialization Messages

**msgAdded:**

Attachment to a scribble object

Takes OBJECT, returns STATUS.

Comments

The extraction object receives this message when it has been made an observer of a scribble object. When it receives this message, the extractor queries the scribble for all strokes which have been added to the scribble up to this time. Henceforth the scribble object will send msgScrAddedStroke to the extraction object every time a new stroke is added to the scribble. Thus one way or another the extractor has access to all the strokes associated with the scribble.

An extractor cannot be an observer of more than one scribble object at a time.
**msgRemoved:**
Detachment from a scribble object
Takes OBJECT, returns STATUS.

Comments
The extraction object receives this message when it is no longer an observer of the scribble object.

**msgXtractGetScribble:**
Reads the id of the attached scribble object.
Takes P_OBJECT, returns STATUS.

```c
#define msgXtractGetScribble MakeMsg(clsXtract, 1)
```

Comments
This message is used to obtain the id of the scribble object that this extraction object is attached to. It can be used by the client or by a subclass if it is interested in modifying and/or reading the scribble information directly.

**Control Messages**

**msgScrAddedStroke:**
Add a stroke to the extraction object.
Takes P_SCR_ADDED_STROKE, returns STATUS.

Comments
This message is received by the extractor from the scribble object each time a new stroke is added to the scribble.

**msgScrRemovedStroke:**
Remove a stroke from the extraction object.
Takes P_SCR_REMOVED_STROKE, returns STATUS.

Comments
This message is received by the extractor from the scribble object each time an existing stroke is deleted from the scribble.

**msgXtractStrokesClear:**
Clears out all strokes previously sent to translation object by scribble
Takes NULL, returns STATUS.

```c
#define msgXtractStrokesClear MakeMsg(clsXtract, 3)
```

Comments
Effectively restarts the translator as if it had just been attached to a fresh scribble.
As a side effect, the shape engine is released. A new shape engine will be attached as soon as new strokes get added by the scribble.
All the settings of the translator remain unchanged (e.g. hwxFlags, xlateCaseMetrics, xlateMetrics, etc).
This message was formerly called msgXtractContextClear. The new name more accurately reflects its functionality.
**msgScrCompleted:**
Notification that no more strokes will be added to scribble.
Takes NULL, returns STATUS.

Comments
This message is sent by the scribble object to the extraction object when the scribble has determined that no more strokes will be added. When it receives this message, the extractor will self-send the message `msgXtractComplete` (see below) to kick off the final stages of translation.

**msgXtractComplete:**
Hook for subclasses to complete the translation.
Takes NULL, returns STATUS.

```
#define msgXtractComplete           MakeMsg(clsXtract, Ox81)
```

Comments
The extraction object self-sends this message when it receives the message `msgScrCompleted`. This message will be processed by the appropriate subclass of `clsXtract` which will complete the translation.

Note that in certain instances (such as multiple line text pads), the translation object may have already translated a subset of the existing strokes as they were entered. This message tells the translation object to finish up (complete) the translation and not wait for any further strokes.
XWORD.H

This file contains the API definition for clsXWord.
clsXWord inherits from clsXText.

 ifndef XWORD_INCLUDED
#define XWORD_INCLUDED
 ifndef GO_INCLUDED
#include <go.h>
 endif
 ifndef UID_INCLUDED
#include <uid.h>
 endif

Common #defines and typedefs

#define xWordSpellCorrection flag0
#define xWordSpellOnly flag1
#define xWordProofEnable flag2
#define xWordAbbrEnable flag3

Messages

msgNewDefaults:
Fills in default values for a new Word translation object.
Takes P_XLATE_NEW, returns STATUS..
Comments
The default values are the same as for clsXText, except for the following hwxFlag setting:
xltSpellingEnable will be ON
xltSmartCaseDisable will be OFF
In addition,
pArgs->xlate.xlateCaseMetrics.type = xcmSentence;
pArgs->xlate.xlateCaseMetrics.writer = xcmMixedCaseWriter;
Capitalize first letter of each sentence, etc.

msgNew:
creates a new Word translation object.
Takes P_XLATE_NEW, returns STATUS..

msgXWordComplete:
Hook for subclasses to indicate completion.
Takes NULL, returns STATUS.
#define msgXWordComplete MakeMsg(clsXWord, 0x81)
Comments
Not implemented
BUTTON_STYLE,
BUTTON_NOTIFY, API1:348, API1:353
BUTTON_NEW_ONLY, API1:348
BUFFER_RETURN, API2:421
BROWSER_USER_COLUMN,
BROWSER_PATH,
BROWSER_NEW_ONLY, API2:183
BROWSER_GOTO,
BROWSER_GESTURE,
BROWSER_CREATE_DOC, API2:196
BROWSER_COLUMN, API2:191
BROWSER_JUSTIFY, API2:191
BROWSER_BOOKMARK, API2:196
BROWSER_COLUMN, API2:191
BROWSER_COLUMN_STATE, API2:192
BROWSER_CREATE_DOC, API2:196
BROWSER_DEF_COLUMN, API2:191
BROWSER_GESTURE, API2:197
BROWSER_GOTO, API2:194
BROWSER_METRICS, API2:191
BROWSER_NEW, API2:187
BROWSER_NEW_ONLY, API2:187
BROWSER_PATH, API2:195
BROWSER_USER_COLUMN, API2:192–193
BUFFER_RETURN, API2:421
BUTTON_NEW, API1:349
BUTTON_NEW ONLY, API1:348, API1:353
BUTTON_STYLE, API1:348, API1:350
BYTE_ARRAY, API2:199
ByteArrayCreate, API2:201
ByteArrayDelete, API2:202
ByteArrayDestroy, API2:201
ByteArrayFindByte, API2:200
ByteArrayFindIndex, API2:200
ByteArrayGapLength, API2:199
ByteArrayGetByte, API2:200
ByteArrayGetMany, API2:201
ByteArrayHeapMode, API2:202
ByteArrayInsert, API2:202
ByteArrayLength, API2:202
ByteArrayPrint, API2:200
ByteArrayRead, API2:203
ByteArrayReplace, API2:201
ByteArrayReserve, API2:202
ByteArrayWrite, API2:203
BYTEBUF_DATA, API2:205–206
BYTEBUF_NEW, API2:205–206
BYTEBUF_NEW ONLY, API2:205
CcittDecode31, API1:230
CcittEncode31, API1:230
CG_GET_OWNER, API2:589
CG_OWNER_NOTIFY, API2:591
CG_SET_OWNER, API2:590
CHARACTER_MEMORY, API1:744
CHOICE_MGR_NEW, API1:357
CHOICE_MGR_NEW ONLY, API1:357
CHOICE_NEW, API1:359–360
CHOICE_NEW ONLY, API1:359
CHOICE_STYLE, API1:359–360
CIM_ATTR_DEINSTALLABLE, API2:526
CIM_FIND_CLASS, API2:526
CIM_FIND_PROGRAM, API2:527
CIM_GET_CLASS, API2:526
CIM_LOAD, API2:527
CIM_TERMINATE, API2:527
CIM_TERMINATE_OK, API2:527
CIM_TERMINATE_VETOED, API2:526, API2:528
CIA.MULT, API1:366
CLASS_INFO, API1:36
CLASS_NEW, API1:6
CLASS_NEW ONLY, API1:6
CIChildEdge, API1:366
CIConstraint, API1:366
CILEExtend, API1:366
CLOSE_BOX_NEW, API1:371
CLOSE BOX_NEW ONLY, API1:371
CLOSE BOX_STYLE, API1:371–372
CIRelWinEdge, API1:366
CLS_SYM_MSG, API1:7
CLS_SYM_OBJ, API1:7
CLS_SYM_STS, API1:7
ClClearStatistics, API1:37
ClDumpStatistics, API1:37
ClMsgToString, API1:33
ClNum, API1:9
ClObjToString, API1:33
ClSetStatistics, API1:37
ClStatistics, API1:37
ClStringToMsg, API1:34
ClStringToObject, API1:34
ClsStringToObj, API1:34
ClsStringToSts, API1:34
ClsStringToTag, API1:34
ClsStsToString, API1:32
ClsSymbolsInit, API1:34
ClsTagToString, API1:33
COMMAND_BAR_NEW, API1:373
COMMAND_BAR_NEW ONLY, API1:373
COMMAND_BAR_STYLE, API1:373–374
CONNECTIONS_COMPARE, API2:372, API2:376
CONNECTIONS_CONNECT_STATE, API2:370
CONNECTIONS_ENUMERATE, API2:371–372
CONNECTIONS_ITEM, API2:370
CONNECTIONS_MENU_ITEM, API2:370
CONNECTIONS_NOTIFY, API2:376–377
CONNECTIONS_PASSWORDS, API2:370
CONNECTIONS_PERMISSIONS, API2:370
CONNECTIONS_REQUEST, API2:374–375
CONNECTIONS_SERVICE_INFO, API2:373
CONNECTIONS_STATE, API2:370–371
CONNECTIONS_TAG, API2:372
CONNECTIONS_TAG_ITEM, API2:373
CONNECTIONS_WARNINGS, API2:370
CONTROL_ENABLE, API1:379
CONTROL_NEW, API1:376
CONTROL_NEW ONLY, API1:376
CONTROL_PROVIDE_ENABLE, API1:381
CONTROL_STRING, API1:376
CONTROL_STYLE, API1:375, API1:377
Coord16From32, API1:234
Coord32To16, API1:234
CORKBOARD_WIN_NEW, API1:150
CORKBOARD_WIN_NEW ONLY, API1:149
COUNTER_ACTION, API1:386
COUNTER_NEW, API1:384
COUNTER_NEW ONLY, API1:383
COUNTER_NOTIFY, API1:386
COUNTER_STYLE, API1:383–384
CSTM_LAYOUT_CHILD_SPEC, API1:366
CSTM_LAYOUT_DIMENSION, API1:366
CSTM_LAYOUT_METRICS, API1:365, API1:367
CSTM_LAYOUT_TOKEN, API1:367
CSTM_LAYOUT_TOKEN, API1:366
CSTM_LAYOUT_TOKEN, API1:365
CSTM_LAYOUT_TOKEN, API1:366
CSTM_LAYOUT_TOKEN, API1:365, API1:368
CSTM_LAYOUT_TOKEN, API1:368
CURRENT_STD_PEN_DATA, API1:708
FxMulIntToInt, API2:125
FxMulIntToIntSC, API2:125
FxMulSC, API2:124
FxNegate, API2:124
FxRoundToInt, API2:128
FxRoundToIntSC, API2:128
FxSin, API2:126
FxSinFx, API2:127
FxStrToBin, API2:129
FxSub, API2:124
FxSubSC, API2:124
FxTan, API2:127
FxTanFx, API2:127

GESTURE_MARGIN_NEW, API2:219
GESTURE_MARGIN_NEWONLY, API2:219
GESTURE_MARGIN_STYLE, API2:219-220
GetAttr, API2:75
GetList, API2:77
GetListX, API2:76
GetNodeName, API2:75
GetSingleAttr, API2:75
GO_DIR_CACHE, API2:105
GO_DIR_ENTRY, API2:104
GO_DIR_ENTRY_HEADER, API2:104
GO_DIR_ENTRY_TYPES, API2:104
GO_DIR_FINDTYPE, API2:103
GO_DIR_USER_ATTR, API2:104
GOTO_BUTTON_GET_LABEL, API1:177
GOTO_BUTTON_NEW, API1:175
GOTO_BUTTON_NEWONLY, API1:175
GRAB_BOX_INFO, API1:418-419
GRAB_BOX_NEW, API1:418
GRAB_BOX_NEWONLY, API1:418
GRAB_BOX_STYLE, API1:417-419
GrabBoxIntersect, API1:420
GrabBoxLocToRect, API1:420
GrabBoxPaint, API1:420

GWIN_GESTURE, API1:642, API1:644,
GWIN_GESTURE_NEW, API1:642
GWIN_GESTURE_NEWONLY, API1:642
GWIN_STYLe, API1:641, API1:643

HASH_ENTRY, API2:224
HASH_INFO, API2:224
HashAddEntry, API2:225
HashDeleteEntry, API2:225
HashFindData, API2:225
HashFindTableEntry, API2:225
HashFree, API2:226
HashInit, API2:226
HashInitDefaults, API2:226
HighU16, API1:56
HighU8, API1:56
HIM_ATTR_ENGINE_AVAILABLE, API2:538
HIM_AVAILABILITY_NOTIFY, API2:539
HIM_GET_SET_ENGINE, API2:539
HIM_NEW, API2:538
HS_PACKET_CHAR_HANDLER, API2:396
HS_PACKET_METRICS, API2:395
HS_PACKET_NEW, API2:397
HS_PACKET_SEND_PACKET, API2:396
HS_PACKET_STATUS, API2:396
HWCUSTOM_NEW, API1:655-656
HWCUSTOM_NEWONLY, API1:655
HWLETTER_NEW, API1:657-658
HWLETTER_NEWONLY, API1:657
HWX_SVC_CURRENT_CHANGED, API2:581
HWX_SVC_NEW, API2:581
HWX_SVC_NEWONLY, API2:581

ICON_CHOICE_NEW, API1:423
ICON_CHOICE_NEWONLY, API1:423
ICON_CHOICE_STYLE, API1:423
ICON_COPY_PIXELS, API1:429
ICON_NEW, API1:426
ICON_NEWONLY, API1:426
ICON_PROVIDE_BITMAP, API1:428
ICON_SAMPLE_BIAS, API1:429
ICON_STYLE, API1:425, API1:427
ICON_TABLE_NEW, API1:431
ICON_TABLE_NEWONLY, API1:431
ICON_TABLE_STYLE, API1:431
ICON_WIN_NEW, API1:180
ICON_WIN_NEWONLY, API1:180
ICON_WIN_STYLE, API1:179, API1:181
IDataDeref, API1:9
IDataPtr, API1:9
IM_ACTIVATE, API2:560
IM_ADD_CARDS, API2:560
IM_ATTR_CURRENT, API2:547
IM_ATTR_DEPENDENT, API2:548
IM_ATTR_INUSE, API2:548
LVNativeName, API:2:112
LVNClose, API:2:108
LVNCreat, API:2:108
LVNDelete, API:2:108
LVNDirPostDeleteAdjust, API:2:109
LVNFush, API:2:112
LVNCreat, API:2:106
LVNCreatAndOpenByDirld, API:2:107
LVNCreatAndOpenParent, API:2:107
LVNCreatAttrInfo, API:2:110
LVNCreatDirld, API:2:109
LVNCreatNumAttr, API:2:110
LVNCreatSize, API:2:111
LVNCreatMove, API:2:108
LVNCreatName, API:2:109
LVNCreatOpen, API:2:107
LVNCreatRead, API:2:111
LVNCreatReadDir, API:2:109
LVNCreatRelease, API:2:107
LVNCreatSetAttrInfo, API:2:110
LVNCreatSetSize, API:2:112
LVNCreatWrite, API:2:111
LVNCreatVolName, API:2:106
LVNCreatSpecificMsg, API:2:106
LVNCreatStatus, API:2:105
LVNCreatUpdateInfo, API:2:106

MARK_COMPARE_TOKENS, API:1:193
MARK_COMPONENT, API:1:186,
API:1:191, API:1:194
MARK_GET_CHILD, API:1:196
MARK_GOTO, API:1:192
MARK_MESSAGE, API:1:187-190,
API:1:196-198
MARK_MSG_HEADER, API:1:187
MARK_NEW, API:1:188
MARK_NEW ONLY, API:1:188
MARK_POSITION_CHILD, API:1:195
MARK_POSITION_EDGE, API:1:195
MARK_POSITION_GESTURE, API:1:196
MARK_POSITION_SELECTION, API:1:196
MARK_POSITION_TOKEN, API:1:195
MARK_SEND, API:1:190
MARK_SHOW_TARGET, API:1:197
MARK_TOKEN, API:1:186, API:1:192-193,
API:1:198
MarkHandlerForClass, API:1:187
MAT, API:1:234
MatCreate, API:1:236
MatDump, API:1:237
MatIdentity, API:1:236
MatInvert, API:1:237
MatMultiply, API:1:237
MatRotate, API:1:237
MatScale, API:1:237
MatTransformRECT32, API:1:239
MatTranslate, API:1:237
MatWHTransform16, API:1:238
MatWHTransform32, API:1:238
MatXYTransform16, API:1:238
MatXYTransform32, API:1:238
Max, API:1:56
MENU_BUTTON_NEW, API:1:464
MENU_BUTTON_NEW ONLY, API:1:464
MENU BUTTON PROVIDE MENU,
API:1:463, API:1:468-469
MENU BUTTON SHOW MENU, API:1:467
MENU BUTTON STYLE, API:1:467,
API:1:465
MENU_NEW, API:1:475-476
MENU_NEW ONLY, API:1:475
MENU STYLE, API:1:475, API:1:477
MIL_svc_ADD TO CONFLICT_MANAGER,
API:2:587
MIL_svc ARE YOU CONNECTED,
API:2:587
MIL_svc_DEVICE, API:2:584, API:2:586
MIL_svc_NEW, API:2:585
MIL_svc_NEW ONLY, API:2:584
Min, API:1:56
MODAL FILTER METRICS, API:1:482
MODAL FILTER NEW, API:1:482
MODEM ACTIVITY, API:2:424
MODEM ANSWER MODE, API:2:429
MODEM AUTO ANSWER, API:2:429
MODEM CHARACTERISTICS, API:2:434
MODEM CONNECTION, API:2:427
MODEM CONNECTION INFO, API:2:427
MODEM DCE CONTROL, API:2:434
MODEM DIAL, API:2:429
MODEM DIAL MODE, API:2:428
MODEM DUPLEX MODE, API:2:431
MODEM_HARDWARE BUFFERS, API:2:434
MODEM_HARDWARE FEATURES,
API:2:433
MODEM HARDWARE MANUFACTURER,
API:2:433
MODEM HARDWARE MODEL, API:2:433
MODEM LINK CONTROL, API:2:427
MODEM METRICS, API:2:433
MODEM_MNP_BREAK_TYPE, API:2:432
MODEM_MNP_COMPRESSION, API:2:432
MODEM_MNP_FLOW CONTROL,
API:2:432
MODEM_MNP MODE, API:2:432
MODEM_NEW, API:2:434
MODEM RESPONSE, API:2:424
MODEM RESPONSE BEHAVIOR, API:2:426
MODEM RESPONSE_INFO, API:2:425
MODEM RESPONSE MODE, API:2:426
MODEM_SEND COMMAND, API:2:427
MODEM_SET_AUTO ANSWER, API:2:429
MODEM_SIGNALING MODES, API:2:430
MODEM_SIGNALING VOICEBAND,
API:2:430
MODEM_SIGNALING WIDEBAND,
API:2:430
MODEM SPEAKER_CONTROL, API:2:431
MODEM SPEAKER VOLUME, API:2:431
MODEM_TIMEOUT, API:2:426
MODEM TONE DETECTION, API:2:430
MOVE COPY ICON DONE, API:1:473
MOVE COPY ICON NEW, API:1:472
MOVE COPY ICON NEW ONLY, API:1:471
MOVE COPY ICON STYLE, API:1:471-473
MOVE ITEMS, API:2:254
MSG HANDLER_FLAGS, API:1:36
MSG_INFO, API:1:36
MSG NOT UNDERSTOOD, API:1:25
MSGABMGR_activate, API:2:347
MakeDialogTag, API:2:381
MakeDialogTag, API:2:550
MakeDynResld, API:2:493
MakeDynUUID, API:2:84
MakeGlobalWKN, API:1:57
MakeIndexResld, API:2:493
MakeInvalidUUID, API:2:83
MakeListResld, API:2:493
MakeMsg, API:1:9
MakeNilUUID, API:2:83
MakePrivateResAgent, API:2:493
MakePrivateWKN, API:1:57
MakeProcessGlobalWKN, API:1:57
MakeStatus, API:1:59
MakeTag, API:1:58
MakeTagWithFlags, API:1:58
MakeU16, API:1:56
MakeU32, API:1:56
MakeWarning, API:1:59
MakeWKN, API:1:57
MakeWknObjResld, API:2:493
MakeWknResld, API:2:493
MakeWknUUID, API:2:83
msgABMgrChanged, API2:348
msgABMgrClose, API2:347
msgABMgrDeactivate, API2:348
msgABMgrIsActive, API2:348
msgABMgrList, API2:347
msgABMgrOpen, API2:346
msgABMgrRegister, API2:346
msgABMgrUnregister, API2:346
msgAdded, API1:25
msgAdded, API1:781
msgAddrBookAdd, API2:357
msgAddrBookAddAttr, API2:361
msgABMgrDeactivate, API2:348
msgABMgrList, API2:347
msgAddrBookGet, API2:355
msgAddrBookSet, API2:356
msgABMgrOpen, API2:351
msgAddrBookSearch, API2:362
msgAddrBookCount, API2:362
msgAddrBookEntryChanged, API2:362
msgAddrBookDelete, API2:358
msgAddrBookRegister, API2:346
msgAddrBookUnregister, API2:346
msgAnimSPaperDone, API:635
msgAnimSPaperGetLine, API:634
msgAnimSPaperGetInterstroke, API:634
msgAnimSPaperGetInterstroke, API:634
msgAnimSPaperGetLine, API:634
msgAnimSPaperGetScale, API:635
msgAnimSPaperReadScribble, API:633
msgAnimSPaperSetDelay, API:634
msgAnimSPaperSetInterstroke, API:634
msgAnimSPaperSetLine, API:634
msgAnimSPaperSetScale, API:635
msgAnimSPaperWriteScribble, API:634
msgANMAddToStationeryMenu, API:522
msgANMCopyInDoc, API2:520
msgANMCreateDoc, API2:519
msgANMCreateSection, API2:519
msgANMDelAll, API2:521
msgANMDelAll, API2:521
msgANMGetNotebookPath, API2:521
msgANMGetNotebookUUID, API2:521
msgANMDelete, API2:521
msgANMDeleteAll, API2:521
msgANMGetNotebookPath, API2:521
msgANMGetNotebookUUID, API2:521
msgANMMoveInDoc, API2:520
msgANMOpenNotebook, API2:522
msgANMPopUpStationeryMenu, API2:522
msgANRMoveFromStationeryMenu, API2:523
msgANMStationeryMenuNameChanged, API2:523
msgAppAbout, API1:102
msgAppActivate, API1:84
msgAppActivateChild, API1:89
msgAppActivateChildren, API1:89
msgAppActivateCorkMarginChildren, API1:89
msgAppAddCards, API1:96
msgAppAddFloatingWin, API1:90
msgAppApplyEmbeddedProps, API1:97
msgAppChanged, API1:108
msgAppChildChanged, API1:106
msgAppClose, API1:87, API1:130
msgAppCloseChildren, API1:92
msgAppCloseChildren, API1:92
msgAppClose, API1:105
msgAppCloseTo, API1:94
msgAppCopied, API1:107
msgAppCopySel, API1:102
msgAppCreateClientWindow, API1:100
msgAppCreated, API1:106
msgAppCreateLink, API1:99
msgAppCreateMenuBar, API1:100
msgAppDelete, API1:108
msgAppDelete, API1:108
msgAppDelete, API1:106
msgAppDeleteLink, API1:100
msgAppDeleteSel, API1:103
msgAppDirAppGetAttrs, API1:113
msgAppDirGetBookmark, API1:116
msgAppDirGetClass, API1:114
msgAppDirGetDirectNumChildren, API1:117
msgAppDirGetFlags, API1:113
msgAppDirGetGlobalSequence, API1:116
msgAppDirGetNextInit, API1:117
msgAppDirGetNumChildren, API1:115
msgAppDirGetSequence, API1:115
msgAppDirGetTotalNumChildren, API1:118
msgAppDirGetUID, API1:114
msgAppDirGetUUID, API1:114
msgAppDirReset, API1:117
msgAppDirSetClass, API1:113
msgAppDirSetBookmark, API1:116
msgAppDirSetClass, API1:114
msgAppDirSetFlags, API1:113
msgAppDirSetNumChildren, API1:115
msgAppDirSetSequence, API1:115
msgAppDirSetUID, API1:115
msgAppDirSetUID, API1:114
msgAppDispatch, API1:88
msgAppExecute, API1:104, API2:458
msgAppExecuteGesture, API1:104
msgAppExport, API1:101
msgAppFindFloatingWin, API1:90
msgAppFloated, API1:106
msgAppGetAppWin, API1:99
msgAppGetAppWin, API1:93
msgAppGetBorderMetrics, API1:97
msgAppGetDocOptionSheetClient, API1:96
msgAppGetEmbeddedWin, API1:93
msgAppGetEmbeddor, API1:93
msgAppGetLink, API1:100
msgAppGetMetrics, API1:87
msgAppGetName, API1:88
msgAppGetOptionSheet, API1:95
msgAppGetRoot, API1:95
msgAppHide, API1:100
msgAppImport, API1:101
msgAppInit, API1:85, API1:129
msgAppInstall, API1:108
msgAppInvokeManager, API1:104
msgAppInstallPageLevel, API1:99
msgAppMgrActivate, API1:123
msgScrollWinGetInnerWin, API1:566
msgScrollWinGetDefaultDelta, API1:565
msgScrollWinGetVertScrollbar, API1:566
msgScrollWinAlign, API1:567
msgScrollWinGetClientWin, API1:565
msgScrollWinGetDefaultDelta, API1:567
msgScrollWinGetHorizScrollbar, API1:566
msgScrollWinGetInnerWin, API1:566
msgScrollWinGetMetrics, API1:564
msgScrollWinGetStyle, API1:563
msgScrollWinGetVertScrollbar, API1:566
msgScrollWinProvideDelta, API1:543, API1:566
msgScrollWinProvideSize, API1:566
msgScrollWinRefreshSize, API1:567
msgScrollWinRemoveClientWin, API1:565
msgScrollWinSetMetrics, API1:565
msgScrollWinSetStyle, API1:564
msgScrollWinShowClientWin, API1:565
msgScavenge, API1:168, API1:342,
msgSelDelete, API1:168, API1:730, API2:296
msgSelDelete, API1:168, API1:730, API2:296
msgSelChangedOwners, API1:474, API2:293
msgSelChoiceMgrAcquireSel, API1:542
msgSelChoiceMgrGetClient, API1:541
msgSelChoiceMgrGetId, API1:541
msgSelChoiceMgrGetRoot, API1:541
msgSelChoiceMgrGetSel, API1:541
msgSelChoiceMgrGetSelClient, API1:541
msgSelChoiceMgrGetSelId, API1:541
msgSelCopySelection, API1:168, API1:730, API2:296
msgSelCopySelection, API1:168, API1:730, API2:296
msgSelDelete, API1:168, API1:730, API2:248, API2:297
msgSelDemote, API1:295
msgSelIsSelected, API1:169, API2:296
msgSelMoveSelection, API1:168, API1:730, API2:297
msgSelOwner, API2:292
msgSelOwners, API2:293
msgSelPrimaryOwner, API2:293
msgSelPromote, API1:169, API2:295
msgSelPromotedOwner, API2:294
msgSelRememberSelection, API1:168, API2:297
msgSelSelect, API1:169, API1:342, API2:295
msgSelSetOwner, API2:291
msgSelSetOwnerPreserve, API2:291
msgSelYield, API1:169, API1:342, API2:294
msgSendServCreateAddrWin, API2:455
msgSendServCreateAddrData, API2:457
msgSendServCreateAddrWin, API2:456
msgSendServFillAddrWin, API2:456
msgSendServGetAddrDesc, API2:458
msgSendServGetAddrSummary, API2:456
msgSetLock, API1:18
msgSetOwner, API1:19, API1:685, API1:728
msgSetProp, API1:20
msgShadowGetBorderWin, API1:544
msgShadowGetShadowWin, API1:545
msgShadowGetStyle, API1:544
msgShadowGetBorderWin, API1:545
msgShadowSetStyle, API1:544
msgSIMGetMetrics, API2:571
msgSiobSetSet, API2:461
msgSiobBreakSend, API2:463
msgSiobBreakStatus, API2:463
msgSiobControlInStatus, API2:462
msgSiobControlOutSet, API2:462
msgSiobEventGet, API2:465
msgSiobEventHappened, API2:466
msgSiobEventSet, API2:465
msgSiobEventStatus, API2:465
msgSiobFlowControlCharSet, API2:463
msgSiobFlowControlSet, API2:464
msgSiobGetMetrics, API2:466
msgSiobInit, API2:466
msgSiobInputBufferFlush, API2:464
msgSiobInputBufferStatus, API2:464
msgSiobLineControlSet, API2:462
msgSiobOutputBufferFlush, API2:464
msgSiobOutputBufferStatus, API2:464
msgSiobReceiveErrorStatus, API2:463
msgSiobSetMetrics, API2:466
msgSiobSetReplaceCharProc, API2:467
msgSMAccess, API2:614
msgSMAccessDefaults, API2:614
msgSMBind, API2:615
msgSMClose, API2:618
msgSMConnectedChanged, API2:622
msgSMFindHandle, API2:620
msgSMGetCharacteristics, API2:619
msgSMGetClassMetrics, API2:621
msgSMGetOwner, API2:616
msgSMGetState, API2:621
msgSMOpen, API2:617
msgSMOpenDefaults, API2:617
msgSMOwnerChanged, API2:622
msgSMQuery, API2:619
msgSMQueryLock, API2:618
msgVSFormatCompleteNotify, API2:117
msgVSFormatMedia, API2:117
msgVSFormatVolume, API2:116
msgVSNameVolume, API2:117
msgVSUpdateVolumes, API2:117
msgWinBeginPaint, API1:284–285, API1:310
msgWinBeginRepaint, API1:284, API1:310
msgWinCleanRect, API1:311
msgWinCopyRect, API1:285, API1:311
msgWinDelta, API1:285, API1:301
msgWinDeltaOK, API1:315
msgWinDevBindPixelmap, API1:286, API1:322
msgWinDevBindPrinter, API1:321
msgWinDevBindScreen, API1:321
msgWinDevGetRootWindow, API1:321
msgWinDevPrintPage, API1:323
msgWinDevSetOrientation, API1:322
msgWinDevSizePixelmap, API1:286, API1:322
msgWinDirtyRect, API1:284, API1:310
msgWinDumpTree, API1:318
msgWinEndPaint, API1:310
msgWinEndRepaint, API1:310
msgWinEnum, API1:312
msgWinExtract, API1:300
msgWinExtracted, API1:316
msgWinExtractOK, API1:315
msgWinFindAncestorTag, API1:309
msgWinFindTag, API1:309
msgWinFreeOK, API1:316
msgWinGetBaseline, API1:304,
   API1:448, API1:459, API1:530,
   API1:609
msgWinGetDesiredSize, API1:304
msgWinGetEnv, API1:318
msgWinGetFlags, API1:306
msgWinGetMetrics, API1:285, API1:306
msgWinGetPopup, API1:308
msgWinGetTag, API1:307
msgWinHitDetect, API1:285, API1:319
msgWinInsert, API1:299
msgWinInserted, API1:316
msgWinInsertOK, API1:314
msgWinInsertSibling, API1:300
msgWinIsDescendant, API1:308
msgWinIsVisible, API1:307
msgWinLayout, API1:302
msgWinLayoutSelf, API1:303,
   API1:370, API1:447, API1:469,
   API1:489, API1:529, API1:535,
   API1:568, API1:575, API1:608,
   API1:729
msgWinMoved, API1:316
msgWinOrphaned, API1:314
msgWinRepaint, API1:314, API1:343,
   API1:448, API1:529, API1:535,
   API1:545, API1:729
msgWinSend, API1:305
msgWinSetFlags, API1:306, API1:415,
   API1:545, API1:568
msgWinSetLayoutDirty, API1:305
msgWinSetLayoutDirtyRecursive, API1:305
msgWinSetPaintable, API1:310
msgWinSetPopup, API1:309
msgWinSetTag, API1:307, API1:430
msgWinSetVisible, API1:309
msgWinSized, API1:317, API1:729
msgWinSort, API1:317
msgWinStartPage, API1:317, API1:459,
   API1:685
msgWinTransformBounds, API1:285,
   API1:310
msgWinUpdate, API1:311
msgWinVisibilityChanged, API1:316
msgXferGet, API1:731
msgXferGet, API2:336
msgXferList, API1:168, API1:731
msgXferList, API2:336
msgXferStreamAuxData, API2:338
msgXferStreamConnect, API2:338
msgXferStreamFree, API2:339
msgXferStreamSetAuxData, API2:339
msgXferStreamWrite, API2:339
msgXGestureComplete, API1:736
msgXTranslateCharConstrainsGet, API1:743
msgXTranslateCharConstrainsSet, API1:743
msgXTranslateCharMemoryGet, API1:744
msgXTranslateCharMemorySet, API1:744
msgXTranslateComplete, API1:746
msgXTranslateCompleted, API1:653, API1:728
msgXTranslateCompleted, API1:747
msgXData, API1:476, API1:770
msgXTranslateFlagsClear, API1:743
msgXTranslateGetFlags, API1:743
msgXTranslateGetHistoryTemplate, API1:745
msgXTranslateGetXlateCaseMetrics, API1:745
msgXTranslateMetricsGet, API1:741
msgXTranslateMetricsSet, API1:740
msgXTranslateModeGet, API1:740
msgXTranslateModeSet, API1:740
msgXTranslateSetFlags, API1:742
msgXTranslateSetHistoryTemplate, API1:746
msgXTranslateSetXlateCaseMetrics, API1:745
msgXTranslateStringSet, API1:741
msgXTranslateTemplateGet, API1:744
msgXTranslateTemplateSet, API1:744
msgXShapeRecognize, API1:765
msgXShapeShapeCompatible, API1:766
msgXShapeShapeEvaluate, API1:767
msgXShapeShapeLearn, API1:767
msgXShapeStrokePreview, API1:764
msgXTeachCompleted, API1:771
msgXTeachEvaluationGet, API1:770
msgXTeachExecute, API1:770
msgXTeachSetId, API1:770
msgXTeachSetTarget, API1:770
msgXTextComplete, API1:780
msgXTextGetXList, API1:780
msgXTextModLine, API1:780
msgXTextNewLine, API1:780
msgXTextWordList, API1:780
msgXtractComplete, API1:783
msgXtractGetScribble, API1:782
msgXtractStrokesClear, API1:782
msgXWordComplete, API1:785
msgZIPGetMyZone, API2:367
msgZIPGetZoneList, API2:367
INDEX 813

ZIP_GETZONES, API2:367

XDeltaDirection, API1:762
XSHAPE_COMPATIBLE, API1:762
XSHAPE_NEW, API1:763
XSHAPE_NEW_ONLY, API1:763
XSHAPE_RECOGNIZE, API1:765
XSHAPE_STROKE_PREVIEW, API1:764
XSNextDirectionCCW, API1:762
XSNextDirectionCW, API1:762
XSOppositeDirection, API1:762
XTEACH_DATA, API1:769
XTEMPLATE_GESTURE_LIST, API1:775
XTEMPLATE_METRICS, API1:774
XTEMPLATE_MODE, API1:774
XTEMPLATE_TRIE_HEADER, API1:774
XTEMPLATE_TYPE, API1:773
XTemplateAddWord, API1:777
XTemplateCheckGesture, API1:777
XTemplateCheckWord, API1:776
XTemplateCompile, API1:774
XTemplateDeleteWord, API1:777
XTemplateFree, API1:776
XTemplateGetMetrics, API1:776
XTemplateSetMode, API1:776
XTemplateWordListSort, API1:776
XTempInit, API1:777
XTEXT_WORD, API1:779
XTM_ARGS, API1:774
XTYPE, API1:752
XY16, API1:233
XY16ToPenStroke, API1:710
XY32, API1:233
XY32inRect32, API1:236

WORD_LIST, API1:746

WIN_COPY_FLAGS, API1:311
WIN_COPY_RECT, API1:311
WIN_DEV_NEW, API1:321
WIN_DEV_NEW_ONLY, API1:321
WIN_DEV_PIXELMAP, API1:322
WIN_ENUM, API1:312
WIN_ENUM_FLAGS, API1:312
WIN_ENV, API1:318
WIN_FLAGS, API1:297
WIN_NEW, API1:298–299
WIN_OPTIONS, API1:297
WIN_RESTORE_ENV, API1:318
WIN_SAVE_ENV, API1:318
WIN_SEND, API1:305
WIN_SEND_FLAGS, API1:305
WIN_SORT, API1:318
WinEachChild, API1:313
WinEndEachChild, API1:314
WinShrinkWrap, API1:297
WinShrinkWrapHeight, API1:297
WinShrinkWrapWidth, API1:297
WKNAdmin, API1:57
WknItemResId, API2:493
WknListResId, API2:493
WknObjResId, API2:493
WknResId, API2:493
WKNScope, API1:58
WKNVValue, API1:57
WKNVer, API1:57
WORD_ENTRY, API1:746
WORD_LIST, API1:746

XSTRING, API1:758
XSTRUNGCASE, API1:758
X2 gestures, API1:758
XFER_ASCII_METRICS, API2:338
XFER_BUF, API2:337
XFER_CONNECT, API2:338
XFER_FIXED_BUF, API2:337
XFER_OBJECT, API2:337
XferAddIds, API2:340
XferListSearch, API2:340
XferMatch, API2:339
XferStreamAccept, API2:341
XferStreamConnect, API2:341
XLATE_BDATA, API1:746
XLATE_CASE_FIELD, API1:739
XLATE_CASE_METRICS, API1:739, API1:745
XLATE_CASE_TYPE, API1:738
XLIST_CHAR_ATTRS, API2:46
XLIST_ELEMENT, API1:752
XLIST_METRICS, API1:754
XLIST_PARA_ATTRS, API2:46
XLIST_TABS, API2:47
XList2Gesture, API1:757
XList2String, API1:758
XList2StringLength, API1:758
XList2Text, API1:749
XListAlloc, API1:756
XListDump, API1:759
XListDumpSetup, API1:759
XList2Text, API1:749
XListDup, API1:757
XListDupElement, API1:757
XListFree, API1:753
XListFreeData, API1:756
XListGet, API1:756
XListGetFlags, API1:756
XListGetPtr, API1:756
XListIndex, API1:755
XListInsert, API1:754
XListMetrics, API1:754
XListNew, API1:753
XListSet, API1:755
XListSetFlags, API1:754
XListTraverse, API1:755
XS_ASCII_MATCH, API1:762
XS_DIRECTION, API1:762
XS_GESTURE_MATCH, API1:762
XS_LD_MATCH, API1:762
XS_MATCH_TYPE, API1:762
XS_OCTAGON, API1:762
XS_RESOURCE_TYPE, API1:761
XS_STROKE, API1:763
XSDeltaDirection, API1:762
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
- □ Minicomputer
- □ Macintosh
- □ 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</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</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
Operating Systems

*PenPoint™ Application Programmatic Interface, Volume I*

Together with Volume II, *PenPoint™ Application Programmatic Interface, Volume I* 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:

- PenPoint class manager
- PenPoint Application Framework™
- Input subsystem and handwriting translation
- User interface toolkit
- Windows and graphics subsystem

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 II* 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