



Penbex SDK

Reference Manual

Preface

This book is divided into several chapters. In order to get the most from reading this book, you should start reading from the first chapter- a brief introduction of Penbex OS. Then you may go on to understand the application architecture, container, and user interface in Penbex OS. If you have any questions about Penbex APIs, you may find a detailed explanation from chapter three and seven in this book. To find a specific function, you are suggest to look up in Index, which is at the end of this book.

Convention Used in This Manual

Some styles of text and layout are used in this book to help you differentiate between different kinds of information. Here are some examples of the styles:

- *Variable names* are in a italic font.
- Function definitions look like this: `BOOL ChangeActiveContainer(WORD id);`

Trademark Acknowledgements

Penbex has endeavored to provide trademark information about all the companies and products mentioned in this book by the appropriate use of capitals. However, Penbex cannot guarantee the accuracy of this information.

Microsoft, MS-DOS, Visual C++, Win32, Windows, and Windows NY are either registered trademarks or trademarks of Micorsoft Corporation in the United States and/or other countries.

Version 2 Release 1 Edition

This edition applies to Penbex OS 2.0 and replaces the Version Beta 2 Release 1.3 of "Penbex OS Software Development Kit Reference Manual". Penbex Data Systems welcome your comments. You may address you comments to the following address:

Penbex Data Systems, Inc.
5F - 1, 106 ChangAn West Road
Taipei, Taiwan (R.O.C.)

FAX: Your International Access Code + 886 + 2 2555 2600
Internet E-mail: penbex_gssd@penbex.com.tw

Please make sure to include the page number and topic related to your comment or note.

©Copyright Penbex Data Systems, Inc. 1999, 2002. All rights reserved.

Contents

Preface	3
Convention Used in This Manual	3
Trademark Acknowledgements	3
Version 2 Release 1 Edition	3
Contents	4
Introduction to Penbex OS	22
Overview.....	22
Features	22
Architecture	23
Penbex SDK (Software Development Kit)	24
Development	24
Application Architecture.....	25
Container Overview	26
User Interface	26
User Interface	27
Application Program Interface	31
UI Object.....	32
General UIOBJ	32
BroadcastMsg2Cont.....	34
ClearActiveParent	34
ClearHookUIMsg	34
EnableDirectMsg	34
ForwardMsg2Cont.....	35
GetCoverRect.....	35
GetUILastError	35
HookUIMsg.....	35
IsOnCoverRect.....	36
IsOnCoverArea.....	36
ProcessUIMsgDirect	36
ReleaseCoverRect	37
RIsLastCoverRect	37
SetActiveParent	37
SetCoverRect	37
uiClean	38
uiDelete	38
uiDeleteTimer	38
uiEnable	38
uiEnableTimer	39
UIExtraProcess	39
uiIsActive	39
uiGetActive	40
uiIsEnable.....	40
uiIsVisible	40
uiNewTimer	40
uiRedraw	41
uiSetActive	41
uiSetInterval	41
uiSetLastError	42
uiSetVisible.....	42
Label.....	43
labClearCaption	44
labGetCaption	44
labNew	44
labNew3DEx.....	45

labSetCaption	45
labSetThreeD	46
Button	47
btnClearCaption	48
btnFrame	48
btnGetCaption	48
btnGetCheckBoxState	48
btnNew	49
btnNew3DEx	49
btnNewEx	50
btnSetBtnGroup	50
btnSetCaption	51
btnSetCheckBoxState	51
btnSetRadioBtnGroup	51
btnSetStrPosition	51
btnSetRepeat	52
btnSetThreeD	52
List	53
GetItemString	55
listAddItem	55
listAddItemArray	55
listAddSpaceBar	55
listGetItemActive	56
listGetItemArray	56
listGetHotKeyText	56
listGetItemPosition	57
listGetItemText	57
listGetTopItem	57
listIsGrayItem	58
listLastActive	58
listMoveLine	58
listMovePage	59
listNew	59
listNewEx	60
listPopUp	60
listPopUpEx	61
listPopUpPro	62
listRealHeight	62
listRemoveItem	62
listRemoveItemAll	63
listSetHotKeyText	63
listSetItemActive	63
listSetItemPosition	64
listSetItemText	64
listSetTopItem	64
listTotalItem	65
listTotItem	65
Table	66
tablBmpOut	67
tablBmpNew	67
tablClearReserve	67
tablGetColumn	67
tablGetReserve	68
tablGetRow	68
tablInsertColumn	68
tablInsertRow	68
tablMoveColumn	69
tablMovePageColumn	69

tablMovePageRow	69
tablMoveRow	70
tablNew	70
tablNewEx	71
tablPopUp	71
tablRemoveColumn	72
tablRemoveRow	72
tablResetActiveZone	72
tablSetActive	73
tablSetActiveZone	73
tablGridToCoord	73
tablSetActiveZoneEx	74
tablSetNoActive	74
tablSetReserve	74
tablSubNew	75
tablTextOut	75
tablTextOutEx	75
Tool Bar	77
tbrAddBtn	78
tbrAddButton	78
tbrGetBtnState	78
tbrGetButtonBorder	79
tbrGetButtonState	79
tbrGetShowTipPos	79
tbrGetTipBorder	80
tbrGetTipFont	80
tbrGetTipHandle	80
tbrGetTipText	80
tbrNew	81
tbrNewEx	81
tbrRemoveBtn	82
tbrSetBtnState	82
tbrSetButtonBorder	82
tbrSetButtonState	83
tbrSetShowTipPos	83
tbrSetTipBorder	83
tbrSetTipFont	84
tbrSetTipText	84
tbrSetToolTip	84
tbrSubIDtoIndex	84
tbrToolBarsEnable	85
Whiteboard	86
wbClean	87
wbCopyBmp	87
wbNew	87
wbNewEx	87
wbPasteBmp	88
wbSetDrawStyle	88
wbSetPenColor	88
wbSetPenColorEx	89
wbSetPenMode	89
wbSetPenStyle	90
wbSetPenWidth	90
Progress Bar	91
progsChangeMaxRange	92
progsEnableTimer	92
progsGetIncVal	92
progsGetPercent	92

progsGetTimer	93
progsIncrease.....	93
progsNew	93
progsNewEx	94
progsReset	94
progsSetIncVal	94
progsSetPercent	95
progsSetTimer	95
Menu Bar	96
menuAddItem	97
menuAddItemArray	97
menuAddMenuHelp	97
menuAddMenuItem	97
menuAddMenuSection	98
menuAddSection	98
menuAddSectionHelp	98
menuGetItem	99
menuGetItemArray	99
menuGetItemIndex	99
menuGetItemState	100
menuGetSectionIndex	100
menuGrayItem	100
menuNew	101
menuNewEx	101
menuPopup	102
menuPopupMenuBar	102
menuRemoveItem	102
menuRemoveItemAll	103
menuRemoveSection	103
menuSectionFocus	103
menuSendHotKey	104
menuSetActiveMenu	104
menuSetItemState	104
menuSetSectionFocus	104
Message Box	106
FloatBox	107
MsgBox	107
Keyboard	109
kbdNew	110
kbdNewEx	110
kbdUnDefWordZone.....	110
SysDefKB	111
SysEngKB	111
Scroll Bar	112
sbarAddTotalSize	113
sbarGetMaxPos	113
sbarGetPageSize	113
sbarGetPos	113
sbarGetTotalSize	114
sbarMove	114
sbarMoveLine	114
sbarMovePage	115
sbarNew	115
sbarNewEx	116
sbarSetPageSize	116
sbarSetPos	116
sbarSetTotalSize	117
sbarShowAR	117

sbarSubTotalSize	117
Combo Box	119
comboAddItem	120
comboAddItemArray	120
comboGetItemActive	120
comboGetItemArray	121
comboGetItemId	121
comboGetItemText	121
comboLastActive	122
comboMoveLine	122
comboMovePage	122
comboNew	123
comboNewEx	123
comboRealHeight	124
comboRemoveItem	124
comboRemoveItemAll	124
comboSetItemActive	125
comboTotalItem	125
Tree	126
treeNew	127
treeRealHeight	127
Single-Line Editor	128
sedActiveBackSpace	129
sedCutSelect	129
sedDeatchData	129
sedDeleteStr	129
sedGetEditText	130
sedGetMaxCnt	130
sedGetModify	130
sedGetNumericFmt	130
sedGetSelect	131
sedGetTextLength	131
sedInfoA	131
sedInfoB	132
sedMove	132
sedNew	132
sedNewEx	133
sedSelectAll	133
sedSetBeginPos	134
sedSetCurPos	134
sedSetEditText	134
sedSetEditTextEx	135
sedSetFocus	135
sedSetMaxCnt	135
sedSetNumericFmt	135
sedSetSelect	136
sedSetModify	136
Multiple-Line Editor	137
medActiveBackSpace	138
medAppendStr	138
medAppendStrEx	138
medDeleteStr	138
medGetCurLineNum	139
medGetCursorCurrentPos	139
medGetEditText	139
medGetMaxCnt	140
medGetModify	140
medGetSelect	140

medGetTextLength	140
medGetTotalLine	141
medGetViewPos	141
medInfoA	141
medInfoB	142
medInfoC	142
medInsertStr	142
medIsOnFocus	143
medMove	143
medNew	143
medNewEx	144
medScrollDown	144
medScrollPageDown	144
medScrollPageUp	145
medScrollUp	145
medSelectAll	145
medSetCurPos	146
medSetEditFont	146
medSetEditText	146
medSetEditTextEx	147
medSetFocus	147
medSetMaxCnt	147
medSetModify	147
medSetSelect	148
medSetViewPos	148
medSetViewPosEx	148
Field	150
fldAddRecord	152
fldClearReserve	152
fldDelAllRecord	152
fldDelRecord	152
fldEnableRepeat	153
fldGetRecord	153
fldGetRecordReserve	153
fldGetRecordReserveNum	154
fldGetTop	154
fldGetWidth	154
fldMove	155
fldMoveLine	155
fldMovePage	155
fldNew	156
fldNewEx	156
fldSetRecordReserve	156
fldSetWidth	157
fldSetReserve	157
fldSetSort	157
fldSetSpecialReserve	158
fldSetTop	158
fldTotalRecord	158
CUI	159
GetFilename	160
GetFilenameEx	160
SearchComplete	160
SearchDrawTitle	161
SearchSaveMatched	161
SearchSetDefMode	161
SearchStrInStr	162
SysGetDate	162

SysGetTime.....	162
SysSearch.....	163
File and Database.....	164
File.....	164
FileAttrib.....	166
FileClose.....	166
FileDelete.....	166
FileDeleteDir.....	167
FileDiskAbort.....	167
FileDiskClose.....	167
FileDiskDefrag.....	168
FileDiskFormat.....	168
FileDiskOpen.....	168
FileFlush.....	169
FileGetCurrentDir.....	169
FileGetDefaultDriver.....	169
FileGetDone.....	170
FileGetFirst.....	170
FileGetFreeSpace.....	170
FileGetFreeSpaceEx.....	171
FileGetNext.....	171
FileGetTime.....	171
FileNewDir.....	172
FileOpen.....	172
FileRead.....	173
FileReadEx.....	173
FileRename.....	173
FileRewind.....	174
FileSeek.....	174
FileSetCurrentDir.....	174
FileSetDefaultDriver.....	175
FileSetTime.....	175
FileTell.....	176
FileTruncate.....	176
FileWrite.....	176
FileWriteEx.....	176
GetDiskVolumeInfo.....	177
GetFileLastError.....	177
NumberOfDir.....	178
NumberOfFile.....	178
SetDiskVolumeLabel.....	178
DR_DeIP.....	178
DR_GetP.....	179
DR_LockP.....	179
DR_MaxBlock.....	179
DR_NewP.....	180
DR_RenameP.....	180
DR_ResizeP.....	180
DR_SizeP.....	181
Database.....	182
DbAppend.....	183
DbCloseDb.....	183
DbCreateDb.....	183
DbCreateNdx.....	183
DbDelete.....	184
DbDeleteAndPack.....	184
DbDeleteDb.....	184
DbDeleteNdx.....	185

DbGotoBottom.....	185
DbGotoTop	185
DbInitial	186
DbIsDeleted.....	186
DbNameOfField.....	186
DbNext	186
DbNumberOfField	187
DbNumberOfRecord	187
DbNumberOfRecordBykey.....	187
DbOffsetBykey	188
DbOpenDb	188
DbOpenDbWithMulNdx	188
DbOpenDbWithNdx.....	189
DbPack.....	189
DbPackEx.....	189
DbPrev	189
DbRead	190
DbRelease.....	190
DbSearch	190
DbSearchFirst	191
DbSearchLast.....	191
DbSeek.....	192
DbSeekBykey.....	192
DbSetMasterNdx	192
DbSizeOfField	193
DbSizeOfRecord	193
DbUnDelete.....	193
DbWrite	194
Data Manager.....	195
DMAppend	197
DMCreateNDX	197
DMCreateNDX2	197
DMCreateNDX3	198
DMDBCclose.....	198
DMDBCreate	198
DMDBDelete	199
DMDBOpen	199
DMDelete	199
DMDeleteNDX.....	200
DMFlush	200
DMGetFieldNoByName.....	200
DMGetLastErrorCode	201
DMGotoBottom.....	201
DMGotoTop	201
DMInitial	202
DMNameOfField	202
DMNextWithNDX	202
DMNumberOfField	202
DMNumberOfRecord	203
DMNumberOfRecordBykey.....	203
DMOffsetBykey	204
DMOpenDBWithMulNDX	204
DMOpenDBWithNDX	204
DMPack.....	205
DMPositionOfNDX	205
DMPrevWithNDX	205
DMRead	206
DMRelease.....	206

DMSearchFirstWithNDX	206
DMSearchLastWithNDX.....	207
DMSearchWithNDX	207
DMSeek.....	208
DMSeekWithNDX.....	208
DMSeekWithNDXBykey.....	208
DMSetMasterNDX.....	209
DMSizeOfField	209
DMSizeOfLengthTable	210
DMSizeOfRecord	210
DMTypeOfField	210
DMWrite	211
Multimedia	212
Graphics	212
GmBmpCopy.....	214
GmBmpCopyEx.....	214
GmCalcStrWidth.....	214
GmCaptureBmp	215
GmCaptureGray4Bmp	215
GmCheckGray4BmpSize	215
GmCheckBmpSize	216
GmContrastDecrease	216
GmContrastGetValue	216
GmContrastIncrease	217
GmCreatePenQuickVal	217
GmDrawBmp.....	217
GmDrawBmpEx.....	217
GmDrawDot.....	218
GmDrawEllip	218
GmDrawLine	218
GmDrawRect.....	219
GmDrawSymbol	219
GmDrawSymbolCnt	219
GmFillArea	220
GmFillEllip	220
GmFillRect.....	220
GmFillScreen.....	221
GmGetCharWidth.....	221
GmGetCurBlinkRef	221
GmGetCurPos	222
GmGetDrawPage	222
GmGetFontHeight	222
GmGetPalette.....	223
GmGetPalette4.....	223
GmGetPenPos	223
GmGetScreen	223
GmGetScreenSize	224
GmGetStartAddr	224
GmGetStrWidth	224
GmGetStrWidthCnt	225
GmGrayArea	225
GmGrayScreen	225
GmIsCurActive	225
GmLoadBmp.....	226
GmLineTo.....	226
GmMagnifyBmp.....	226
GmMoveTo.....	227
GmPutScreen.....	227

GmReduceBmp	227
GmReleaseBmp	228
GmRevertRect	228
GmSetBrushPattern	228
GmSetCurBlinkRef	229
GmSetCurBlinkTime	229
GmSetCurPos	229
GmSetCurWH	229
GmSetCustomPattern	230
GmSetDrawPage	230
GmSetFont	230
GmSetPalette	231
GmSetPalettes	231
GmSetPalette4	231
GmSetPenBkColor	231
GmSetPenBkColorRGB	232
GmSetPenColor	232
GmSetPenColorRGB	232
GmSetPenMode	233
GmSetPenPattern	233
GmSetPenPos	233
GmSetPenQuick	234
GmSetPenQuickEx	234
GmSetPenWidth	234
GmSetTextClip	234
GmSetSymbol	235
GmShiftScrn	235
GmShowCur	235
GmTextOut	236
GmTextOutCnt	236
GmTextOutLF	236
GmVPageCopy	237
GmVPageCopyEx	237
GmVPageCreate	238
GmVPageDelete	238
Voice	239
sndClose	240
sndControl	240
sndGetFileInfo	241
sndGetLastError	241
sndGetVolume	241
sndInit	242
sndPause	242
SndPlay	242
sndQueryPosition	243
sndSetDirection	243
sndSetMode	243
sndSetPosition	244
sndSetSpeed	244
sndSetVolume	244
sndStart	244
sndStatus	245
sndStop	246
Tone	247
ToneBeep	248
ToneClose	248
ToneGetVolume	248
ToneOpen	248

TonePlayMIDIFile	248
TonePlayMusic	249
ToneSetFrequency	249
ToneSetValue	249
ToneSetVolume	250
ToneStopFrequency	250
ToneStopMusic	251
ToneStopValue	251
Communication	252
Uart	252
UrtCancelSendData	254
UrtCancelXmodemGet	254
UrtCancelXmodemSend	254
UrtClearRxFIFO	254
UrtClose	254
UrtCTSSStatus	255
UrtImmGetData	255
UrtImmSendData	255
UrtIsRxReady	256
UrtIsTxAvailable	256
UrtIsTxEmpty	256
UrtOpen	256
UrtRcvModeBegin	257
UrtRcvModeEnd	257
UrtRcvReset	257
UrtSendData	257
UrtSet	258
UrtSetRTS	258
UrtSetWay	259
UrtXGetContinue	259
UrtXmodemGet	259
UrtXmodemSend	259
Infrareds	261
IrLptBusy	264
IrLptEnd	264
IrLptSend	264
IrLptStart	264
IrObexAutoRecvSwitch	265
IrObexConnect	265
IrObexDisconnect	265
IrObexEnd	266
IrObexGetAutoRecvMode	266
IrObexGetData	266
IrObexIsConnected	266
IrObexRegister	267
IrObexSendData	267
IrObexSendObject	267
IrObexSetAutoRecvMode	267
IrObexStart	268
Ir_AdvanceCredit	268
Ir_Bind	268
Ir_ConnectIrLap	269
Ir_ConnectReq	269
Ir_ConnectRsp	269
Ir_DataReq	270
Ir_DisconnectIrLap	270
Ir_DiscoverReq	270
Ir_End	271

Ir_IAS_Add	271
Ir_IAS_GetInteger	271
Ir_IAS_GetIntLsap	272
Ir_IAS_GetObjectID	272
Ir_IAS_GetOctetString	272
Ir_IAS_GetOctetStringLen	272
Ir_IAS_GetType	273
Ir_IAS_GetUserString	273
Ir_IAS_GetUserStringCharSet	273
Ir_IAS_GetUserStringLen	274
Ir_IAS_Next	274
Ir_IAS_Query	274
Ir_IAS_SetDeviceName	274
Ir_IAS_StartResult	275
Ir_IsIrLapConnected	275
Ir_IsMediaBusy	275
Ir_IsNoProgress	275
Ir_IsRemoteBusy	276
Ir_LocalBusy	276
Ir_MaxRxSize	276
Ir_MaxTxSize	277
Ir_SetConTypeLMP	277
Ir_SetConTypeTTP	277
Ir_SetDeviceInfo	277
Ir_Start	278
Ir_TestReq	278
Ir_Unbind	278
TCP/IP Networking	280
NetAbort	286
NetAccept	286
NetAddRoute	286
NetBind	287
NetCheckFD	287
NetCloseSocket	287
NetConnect	288
NetDial	288
NetGetDialInfo	288
NetGetHostByAddr	289
NetGetHostByName	289
NetGetMailInfo	289
NetGetPeerName	289
NetGetSocketOption	290
NetGetIpAddr	290
NetHangUp	291
NetInitFD	291
NetIsConnected	291
NetListen	291
NetPop3Close	292
NetPop3Commit	292
NetPop3Delete	292
NetPop3List	293
NetPop3Logoff	293
NetPop3Logon	293
NetPop3Noop	294
NetPop3Reset	294
NetPop3RetrieveCallBack	294
NetPop3RetrieveMsg	294
NetPop3State	295

NetPop3Top.....	295
NetPop3Uidl.....	295
NetReceive.....	296
NetResetFD.....	296
NetReceiveFrom.....	296
NetSelect.....	297
NetSend.....	297
NetSendTo.....	297
NetSetBlock.....	298
NetSetDialInfo.....	298
NetSetFD.....	298
NetSetMailInfo.....	299
NetSetSocketOption.....	299
NetSocket.....	299
NetSmtpClose.....	300
NetSmtpData.....	300
NetSmtpExpn.....	300
NetSmtpHello.....	301
NetSmtpHelp.....	301
NetSmtpMail.....	301
NetSmtpMessage.....	302
NetSmtpNoop.....	302
NetSmtpQuit.....	302
NetSmtpRcpt.....	302
NetSmtpReset.....	303
NetSmtpStart.....	303
NetSmtpVerify.....	303
Net_Abort.....	304
Net_Accept.....	304
Net_CloseSocket.....	304
Net_Connect.....	305
Net_Dial.....	305
Net_GetHostByAddr.....	305
Net_GetHostByName.....	305
Net_GetPop3Info.....	306
Net_GetSmtpInfo.....	306
Net_Listen.....	306
Net_Pop3Close.....	307
Net_Pop3Commit.....	307
Net_Pop3Delete.....	307
Net_Pop3List.....	307
Net_Pop3Logoff.....	308
Net_Pop3Logon.....	308
Net_Pop3Noop.....	308
Net_Pop3Reset.....	308
Net_Pop3RetrieveMsg.....	309
Net_Pop3Top.....	309
Net_Pop3State.....	309
Net_Pop3Uidl.....	310
Net_Receive.....	310
Net_ReceiveFrom.....	310
Net_Select.....	311
Net_Send.....	311
Net_SendTo.....	311
Net_SmtpClose.....	312
Net_SmtpData.....	312
Net_SmtpExpn.....	312
Net_SmtpHello.....	312

Net_SmtpHelp	313
Net_SmtpMail	313
Net_SmtpMessage	313
Net_SmtpNoop	314
Net_SmtpRcpt	314
Net_SmtpStart	314
Net_SmtpQuit	314
Net_SmtpReset	315
Net_SmtpVerify	315
System Resource	316
Container	316
ChangeActiveContainer	317
ChangeActiveContainer	317
ContAddObj	317
ContGetObj	317
ContRedraw	317
ContRemoveObj	318
ContSetTitle	318
EnterCriticalSec	318
HookContMsg	318
HookPopContMsg	319
IsPopContActive	319
LeaveCriticalSec	319
PopContainer	320
PopContainerEnd	320
SetActiveContainer	320
SetContainerEntry	320
SetTabContEntry	321
SetTitleTabContEntry	321
Timer	322
DisableTimerMsg	323
EnableAutoSleep	323
EnableTimerMsg	323
GetAutoSleepTime	323
IsTimerMsgOn	324
SetAutoSleepTime	324
SetTimerMsg	324
Sleeping	324
Message	326
AppProcess	327
GetMsg	327
GetMsgNoWait	327
SendMsg	327
SendMsgTop	328
SysProcess	328
Key	329
EnableKeyRepeatMsg	330
GetCurrentKeyState	330
SetKeyRepeatTick	330
Pen	331
Calibration	332
EnablePenDoubleClick	332
EnablePenMoveMsg	332
EnablePenRepeatMsg	333
SetDoubleClickTick	333
SetPenMoveTick	333
SetPenRepeatTick	333
Application	335

AppAdd.....	337
AppCallAp	337
AppGetCell.....	337
AppGetID.....	338
AppGetIndex	338
AppIsReEntrant	338
AppInsert.....	338
AppInsertByDR.....	339
AppRemove.....	339
AppReturnHome	339
AppRunAp	340
AppRunApCont	340
AppRunApEx.....	340
AppRunPbxFile	340
AppRunPreAp	341
AppTotal	341
Memory.....	342
MmDelH	343
MmDelP	343
MmLockH	343
MmNewH.....	343
MmNewP	344
MmResizeH.....	344
MmResizeP	344
MmSizeofH.....	345
MmSizeofP	345
MmMaxFreeBlock	345
MmTotalFreeMem	345
MmUnlockH.....	346
Battery	347
BattCurrentValue	348
BattReadChargingStatus	348
Alarm	349
RtcAddAlarm	351
RtcDelAlarm	351
RtcDelAllAlarm	351
Real Time Clock	352
RtcDisableStopWatch	353
RtcEnableStopWatch	353
RtcGetDate.....	353
RtcGetDateTime.....	353
RtcGetTime	354
RtcGetTime2	354
RtcSetDate	354
RtcSetTime.....	355
Copy Pool.....	356
CPClean.....	357
CPGetData	357
CPGetDataInfo	357
CPPutData	357
Shared Library.....	359
SIClose	360
SICloseAll	360
SIFunc	360
SIInit	360
SIOpen	361
System Settings.....	362
SysGetHwInfo	363

SysGetIOInfo	363
SysGetOsInfo	364
Handwriting Recognition	365
EnhGetState	366
EnhRecognize	366
EnhSetState	366
hregGetDefPara	366
hregNewEx	367
hregSetDefPenWidth	367
hregSetDefTimeOut	367
PbxHregEnableInk	367
PbxHregGetParam	368
PbxHregInit	368
PbxHregPlugIn	368
PbxHregPlugOut	369
PbxHregSetParam	369
Trap	370
TrapGetEntry	371
TrapGetLastApiNum	371
TrapHookEntry	371
Miscellaneous	372
OSVersionStr	372
Appendix A	373
Supported ANSI C Library	373
Abs	373
ACos	373
Asctime	373
ASin	373
Assertion	373
ATan	373
ATan2	374
Atof	374
Atoi	374
Atol	374
Bsearch	374
CalcWeekDay	374
CalcWeekDay2	375
CalcYearDay	375
Calloc	375
Ceil	375
Clock	375
Cos	375
Cosh	376
CoTan	376
CTime	376
Difftime	376
Div	376
dw2Str	376
Exp	376
Fabs	377
Floor	377
Fmod	377
Free	377
Frexp	377
GmTime	377
Hypot	377
Isalnum	377
Isalpha	378

Isascii	378
Iscntrl	378
Isdigit	378
Isgraph	378
Isleadbyte	378
Islower	378
Isprint	378
Ispunct	379
Isspace	379
Isupper	379
Isxdigit	379
Labs	379
Ldexp	379
Ldiv	379
Localtime	379
Log	379
Log10	380
Longjmp	380
Lunar2Solar	380
Malloc	380
Memchr	380
Memcmp	381
Memcpy	381
Memmove	381
Memset	381
Mktime	381
Modf	381
Pow	382
Qsort	382
Rand	382
Realloc	382
SetGmRefTime	382
Setjmp	382
Sin	382
Sinh	383
Solar2Lunar	383
Sprintf	383
Sqrt	383
Srand	383
Str2lower	384
Str2upper	384
Strcat	384
Strchr	384
Strcmp	384
Strcpy	384
Strcspn	385
Strerror	385
Stricmp	385
Strlen	385
Strncat	385
Strncmp	385
Strncpy	385
Strpbrk	385
Strrchr	386
Strspn	386
Strstr	386
Strtod	386
Strtok	386

Strtol	386
Strtoul	386
Tan.....	387
Tanh.....	387
Time.....	387
Toascii	387
Tolower	387
Toupper	387
Index	388

Introduction to Penbex OS

Overview

With the coming of post-PC era, the fast development and application of Information Appliance have been the core of the development of the information products in the next generation. But in the present time, only PDA has brilliant performance and has become the darling recently. However, the major PDA companies are foreign ones and they are not aggressive on Chinese platforms. Thus, Penbex OS is born to be a Chinese operating system under the circumstances and it provides a multilingual version.

Custom-made for PDA, Penbex OS is a high-performance operating system released in 2000. Its major features are open and modular system architecture. Individual programmers and companies can easily create various kinds of applications. Penbex Data Systems, Inc. offers Penbex SDK for software publishers or individuals to download on Penbex world-wide websites (<http://www.penbex.com>, <http://www.penbex.com.cn>, <http://www.penbex.com.tw>). Programs which have been developed can be dynamically downloaded into PDA. Penbex Data systems also plans to develop freeware for PDA users to download. Thus, it can make PDA extends its functions to the limit.

Besides the powerful expanded features of Penbex OS, it provides UI which is easy to identify, easy to understand, and easy to operate with. If users want to input data, they can input it all by the English-Chinese handwriting recognition system. This allows users to use PDA more easily; furthermore, Penbex OS reduces a great amount of time to figure out how to create PDA applications. In addition, Penbex OS is suitable not only for PDA applications, but also for the development of various mobile information appliances. It also supports the well-known CPUs such as Motorola DragonBall and ARM in the industry. Such a versatile operating system is exactly the best one we want to offer users, developers and software publishers to develop PDA applications.

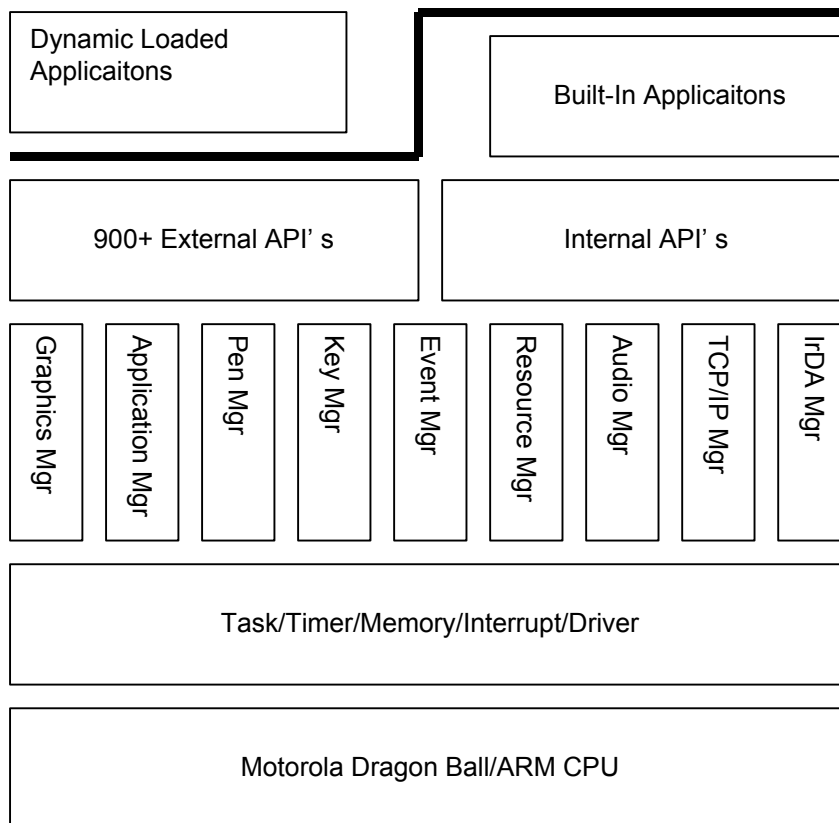
Features

- Micro kernel architecture, services could be added or removed.
- Non-preemptive multitasking based.
- "External" (Dynamic, Downloadable) AP downloadable.
- Open API architecture for accessing OS services.
- Free GNU tool chain for developing the "External" AP.
- Does not need extra resource files.
- Supports four gray levels LCD display.
- Supports only 160*160 resolution LCD display in SDK 1.0.
- Upgradeable OS.
- Built-in database engine.
- Built-in PPP, TCP/IP, UDP, POP3/SMTP services.
- Built-in IrDA 1.1 service, but not supports in Emulator.
- Built-in "Hand Writing Recognition" engine.
- Supports RS-232 driver.
- Supports audio service. More audio APIs will be available.
- Supports touch panel service.
- Supports keypad service.
- Supports power management service.
- Supports Direct-Sync (synchronization) service.
- IDE environment of emulator in Microsoft VC++6.0, full simulation in PC Windows environment.
- Source-level debugging, setting breakpoint, tracing, step tracing, stack watch, local/global variables watch in Microsoft VC++6.0.

- Easy to develop/debug applications.

Architecture

- Penbex OS is based on multi-tasking real time OS. This ensures real time response of the system.
- Hardware-related driver is a module within OS. Thus, it is easy for porting to a new hardware without affecting other modules.
- Penbex OS provides ANSI standard APIs, and Penbex OS APIs.



- Penbex OS provides several objects for application programmers.
- Application programmers shall make good use of Penbex OS APIs and objects as possible to relieve coding efforts.

Penbex SDK (Software Development Kit)

Development

Prerequisite

- Microsoft VC++ 6.0 Environment
- ANSI C Syntax
- Penbex OS Architecture
- Penbex Objects

Documents

- Documents for APIs, objects' behavior, messages.
- Developing/Debugging procedures for external application.

Emulator

- Workspace for IDE environment running on VC 6.0.
- Header files for IDE environment running on VC 6.0.
- OS library for IDE environment running on VC 6.0.

Sample Code

SDK includes some API demo sample codes. And more sample codes are available through Internet Penbex Developer On-line websites.

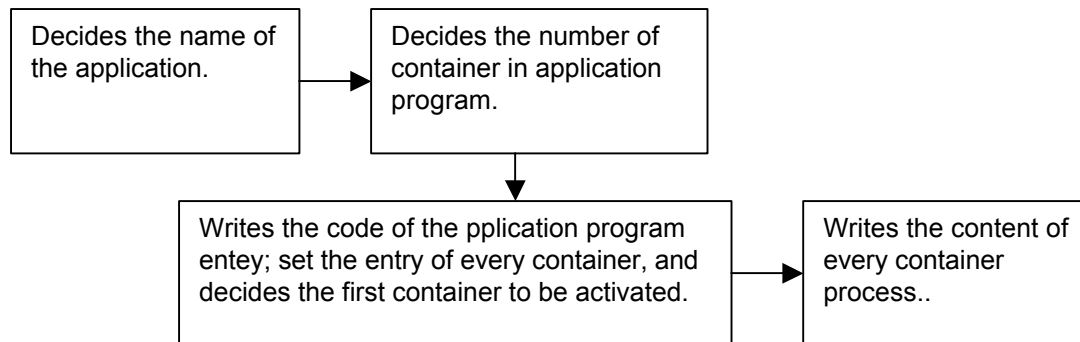
GNU Tool Chain

Penbex SDK provides GNU based developing environment for external applications. This environment includes cross-linker, cross-compiler, cross-linker, library, and header files for GNU tool.

Application Architecture

Message Process

Applications on Penbex OS are all message-driven. Only one application is running at a time; all other applications must be closed. This means that OS always closes the running application before activating another application.



When you're writing application programs, it is important to notice that what you write is a Download Application Program or a Preload Application Program. The greatest difference of the two is that the Preload Application Program needs a Global data structure. In addition, you need to include the application program by using the function AppAdd() in expreap.c. But there is no such a limit in Download Application Program. But one point to look for is that you cannot use const data type in Download Application Programs.

Global Data

Preload Application Program must have a single structure of Global Data to store the Global variables. The structure will be declared in the header file of every application, and its physical address, length, and structure pointer will be transferred to the system by AppAdd() in expreap.c. For example, if you declare a global data structure, and create a physical structure cAppVar and a structure pointer pApp, AppAdd(XX,YY, "App.px", AppMsgA[DKM_TITLE], App, 0, (void *)&cAppVar, sizeof(AppVar), (void **)&pApp), and then it will allow system to manage the global data.

Download Application Program has no limit as it has in Preload Application Program; the only limitation is that it cannot use const data type.

Localization

Penbex OS has supports for localization:

- Supported languages: Simplified Chinese, Traditional Chinese, English, Spanish and Germany version.
- Input locales: offers PingYing input locale for simplified Chinese version.
- System message: offers the Windows-like message box to display the system message.
- Hardware Dependent: provides panel control, and the Printed Pad controlling program to make changes if necessary.

Container Overview

Container, to explain it literally, is something you can put in, which can contain various things. When it is full, you have to find another container to put in more things. Therefore, the feature of a container is to be filled in and its size is limited. This can also apply to Penbex OS. For instance, if you create a screen and there is a title "Welcome to Penbex OS" on it, you can simply regard this screen as a container, and the title can be seen as an object in it. The words on the title can be seen as the features of an object. Likewise, you can put different kinds of things into the container and everything has its own features. If a container is filled with objects, you have to take another one. You can use the concept of object-oriented programming to think about objects and its features. However, it will be discussed later. Now let's clarify some points.

Container can have different kinds of objects or the same kind of objects. A container will function differently because of the different kinds of objects in it. For example, the container that is used to put candy in is called candy box. Objects decide the type of containers.

When the object is being put into the container, you have to have an action of "putting". It is important because if you just add a new object but you don't add it into the container, there is no object in the container. It is as you buy new candy, but you don't put it into the candy box. No matter how much candy you buy, but you fail to put it in the box, there is no candy in the candy box. Therefore, when you create a new object, you have to remember to add it into the container.

Once the objects have been added to Container, any commands to the objects will be seen as the part of the container. Any commands to the objects should be maintained by the container. Object itself does not receive the command and does things. It is, as you have to get candy by opening the candy box. When you get the candy, you can enjoy the "service" of the candy itself.

Besides, the objects cannot overlap with each other in the container. Every object has its own area, or container cannot manage the command to objects because it doesn't know which object will react. It is just as a bottle of clean water that is filled by ink. You don't know if you get the clean water or ink.

The explanations mentioned above are just related characteristics of a container. Then let's probe more about container in Penbex OS.

First, one thing to bear in mind is that one container is one screen. Then, let's look at figure 1. This is a screen on PDA, which contains five objects, one label and four buttons. Every button stands for one game. Thus, you can call this container as a game container. If you click one of the game buttons, it would enter the container. Therefore, there is only one active container. Not every object in container will interfere with each other. That is to say, if you press the button "minesweeper," then the current container will call the "minesweeper" container and then create a new screen as figure 2. We have mentioned that the function of a container is decided by its objects. Judging from figure 1, you can see that this container is a game menu.

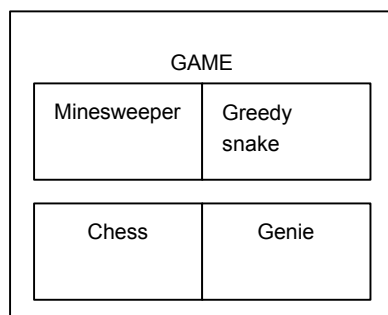


Figure 1

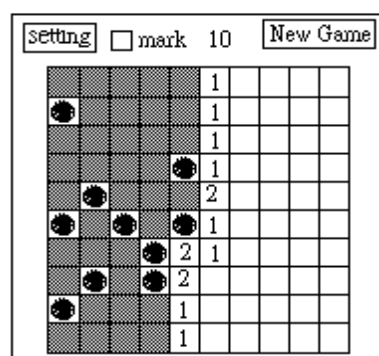


Figure 2

User Interface

Look & Feel

UI stands for User Interface. Put it simply, an interface provides a way for users to communicate with the operating system. For example, Windows is an application of UI; it is also called GUI (Graphics User Interface). Penbex OS also provides GUI, but it is still called UI here to differentiate itself from Windows. A tool enables users to use system resources. For example, there is a UI label object and a UI button object in figure 1. Both of them have their own function and behavior. To take label object as an example, it provides the function of displaying text and its behavior is to set the displayed text, get the text, and clean the text. We only need to know its function and know how to use it to achieve the effect of displaying one line. We don't have to know that how label object works. This is the benefit of UI objects. It is a bridge between system and us, and performs what we need it to do.

Before we start to use UI Objects, we have to figure out the features of every UI Object. Penbex OS currently provides some UI objects as following.

- Label
- Button
- List
- Table
- Toolbar
- Whiteboard
- Progress Bar
- Menu Bar
- Keyboard
- Message Box
- Scroll Bar
- Combo Box
- Tree
- Single-Line Editor
- Multiple-Line Editor
- Field

The GUI will be further explained later. In fact, the GUI objects are the more complicated objects, which combines many UI objects to perform the special function. For example, Penbex OS provides several kinds of GUI objects: SysGetDate() (to get the system date), SysGetTime() (to get the system time), GetFilename() (to get the file name), SysEngKB() (to use the system keyboard), etc. Those are complex but useful functions. Developers only need to call the function. They don't have to create it themselves.

How to Use UI Object

Now we have a basic understanding of UI provided by Penbex OS after looking at the table mentioned above. Now we will learn how to create an object and add it to the container, and finally, to activate it. First, we list the steps in the following and then we will implement a small program.

- Step 1. to create a new object first.
- Step 2. to add the new object to the container.
- Step 3. to paint it on the screen.
- Step 4. to write code of the object to execute.

Let's look at the code in the following. Then we can understand what's the meaning of the steps mentioned above.

```
01: #define ID_Lab_Time    100
02: static BOOL conAProcess(SysMsg *pMsg)
03: {
```

```

04:     UIOBJ *pLabelA;
05:
06:     if (pMsg->type == MT_SYS_CONT_BEGIN)
07:     {
08:         pLabelA = labNew(ID_Lab_Time, 10, 10, 55, 16, "OK",
09:                         LABSTYLE_STRING, LABSTRPOS_MIDDLE,
10:                         GM_FONT_MIDDLE, LABFRAME_YES);
11:         ContAddObj(pLabelA);
12:         ContRedraw(); // paint the object
13:     }
14: }

```

The program mentioned above is to create a label in Container and to display "Welcome" on the label. This program can be divided into three parts. Line 04~09 is to create a new object. Line 10 is to add the object to the container. Line 11 is to paint the object in the container. Here we see that line 04 is to declare an object pointer of label type which represents the new added object. Therefore, the object created in line 07~09 will be designated to pLabelA. Then Line 10 is to add the Label object to the container. By doing so, then every object can be displayed in Container. Finally, we have to paint the container and then we finish a simple screen.

We will not explain much every usage of the parameters of every object here. We only explain how to use UI object in Container. If you want to see the explanation of object's parameter, please refer to the reference manuals.

Then let's take a look at a more detailed program and figure out what does the steps mentioned above mean.

```

01: #define Lab_ID      10
02: #define Bnt_Hello_ID 11
03: #define Bnt_Bye_ID  12
04: static BOOL conAProcess(SysMsg *pMsg)
05: {
06:     UIOBJ *pLabelA;
07:     CButton *pBtn;
08:
09:     if (pMsg->type == MT_SYS_CONT_BEGIN)
10:     {
11:         pLabelA = labNew (Lab_ID, 10, 10, 55, 16, "OK",
12:                         LABSTYLE_STRING, LABSTRPOS_MIDDLE,
13:                         GM_FONT_MIDDLE, LABFRAME_YES);
14:         ContAddObj(pLabelA);
15:         pBtn = btnNew(Bnt_Hello_ID, 10, 30, 55, 16, "Hello",
16:                     LABSTYLE_STRING);
17:         ContAddObj(pBtn);
18:         pBtn = btnNew(Bnt_Bye_ID, 10, 50, 55, 16, "GoodBye",
19:                     LABSTYLE_STRING);
20:         ContAddObj(pBtn);
21:         ContRedraw(); // paint the object
22:     }
23:     else if(pMsg->type == MT_BUTTON_COMMAND)
24:     {
25:         if (pMsg->id == Bnt_Hello_ID)
26:         {
27:             pLabelA = ContGetObj(Lab_ID);
28:             labSetCaption(pLabelA, "Hello");
29:             uiRedraw(pLabelA);
30:         }

```

```

31:         else if (pMsg->id == Bnt_Bye_ID)
32:         {
33:             pLabelA = ContGetObj(Lab_ID);
34:             labSetCaption(pLabelA, "GoodBye");
35:             uiRedraw(pLabelA);
36:         }
37:     }
38: }

```

When the code is put into Application Program, it will create a screen like figure 3. Its function is that when you press the "Hello" button, the "Hello" message will be displayed on the uppermost label, which is presented in figure 4. When you press the "GoodBye" button, then the "GoodBye" message will be displayed as figure 5. Now let's see what's happening in the program. We add new button objects in the previous program. We can see that the way to create new UI objects is the same. The difference is that it's not the same object. It's worth mentioning that whenever the new objects are created, you have to call ContAddObj(). This step should not be forgotten. You have to tell container that you have added a new object. Then the second thing to notice is that the object pointer (such as pLabelA) is reusable because whenever we create new objects, we have used object pointers to register the new objects into container. That is why when we create new objects, we have to add them to container, or information of the object will be lost. After it is added to Container, then we can draw the container on the screen, and we will get the screen like figure 3. Now let's see line 22~31. It's the final step: to write about the function of the object. First, we look at what MT_BUTTON_COMMAND is like. This is the message of UI objects. It tells system that if the button is pressed, then the program will run. Then what is MT_SYS_CONT_BEGIN? This is the system message that lets system know that the program starts from here in the container. Then we will know that the execution of any program is activated by the transmission of messages. It uses pMsg->type to achieve the effect. And pMsg->id is what we choose to click. Therefore, we identify objects according to its ID. It's just as everyone has a social security number. We give a number to the object when we create it. From that time on, whenever we want to use it, then we only need to use ID to activate the UI object we want. Line 24 and 29 are to activate the objects by using its ID. Although we have designated an object to the container when we create it, when we want to use it we have to get it from the container and designate it to the object pointer. Please notice that different kinds of objects cannot share the same object pointer; that is to say, label object has to use the object pointer of label type. Likewise, Button object has to use the object pointer of button type. It's a common mistake that we may make. After we get the object pointer, then we can give commands to objects. LabSetCaption (pLabelA, "Hello") has been used in this example. We want to change the words on the label, but if we don't add uiRedraw(pLabelA), the result is not what we want. It's just as when we create new objects, although container has known that objects have been added, we still have to repaint the area. Therefore, when we give a command to the object, if it changes the object's outline, we have to call uiRedraw() to repaint the object.

That's all about the creation of a UI object. There are some points you have to notice:

- The object pointer can be reusable, but it is noticeable that it is the object's pointer of its own type that you can use.
- After you have created a new object, remember to use ContAddObj() to add the object to Container.
- The ID of the object is the way you identify this object. Please don't get confused with other objects.
- When you use the object, remember to get it from the Container. You can set (*object pointer*) = ContGetObj(*ID of the object*), and then you can use the object.
- When the output of the objects is affected because of calling or resetting the object, you have to repaint the objects. You can use uiRedraw(*object pointer*) to get the correct result.

The object's shared function and their introductions are in the following table:

uiDelete(UIOBJ *pObj)	To remove the object from the container permanently and release the memory of it.
uiRedraw(UIOBJ *pObj)	Repaint the object. It is needed if the appearance of the object has to be modified.

uiClean(UIOBJ *pObj)	Object cleans the area in which it occupies, but object itself still is in the container. Besides, if you click the position where the object is in, the object still accepts the message. Therefore, you need to disable it.
uiEnable(UIOBJ *pObj, BOOL isEnabled)	Set the status of the objects. When isEnabled=1, it means that the objects can receive messages or orders. If isEnabled=0, it means that the object has been closed and cannot receive messages or commands. This function can be used with uiClean().
uiSetVisable(UIOBJ *pObj, BOOL isVisible)	To set the object visible or invisible, when isVisible=0, it will automatically clear the area in which the object occupies. When isVisible=1, then it will repaint the object.
GetUILastError(void)	If the outcome of the action of the object doesn't match what you expect, then you can use this function to get the error number. The definition of error numbers can be found in Typedef.h

The functions mentioned above are the behaviors, which all the objects will have. For example, if you want to clear the object from the screen, you can use uiEnable() to set isEnabled as 0 in order to close the object. By doing so, you will not encounter that the object disappearing in the container continues to react to the screen.

Message (Event)

Any object in Penbex OS sends data or commands to each other by sending messages. It is important for developers to handle the messages in Penbex OS. There are several messages in Penbex OS:

PEN MESSAGES	To handle the event of the pen on Touch Pad.
KEY MESSAGES	To handle the event of external keys.
TIMER MESSAGES	To handle the timer event.
SPECIAL MESSAGES	To handle the special key event or other events which cannot be categorized.
UART MESSAGES	To handle the UART related event.
SYSTEM MESSAGES	To handle the system Container events.
UI MESSAGES	To handle the events of the UI related objects.

The list above is just a categorized one. Please look for detailed information in msg.h. There are all of the definitions of messages and how to use them in section 4.2 How to Use UI Object.

Application Program Interface

Penbex Library

Please refer to the more detailed information in pbxos.h, GUI.h, Gnetapi.h and Gansilib.h in the directory (*SDK source*)\Pbxgsdk\usr\include.

ANSI Library

Penbex OS supports some of the ANSI C library. These supported function calls can be found later in this book in Appendix A.

UI Object

In this chapter, some general UI functions are introduced.

General UIOBJ

General Description

The following constants are defined for UI object. Some of them are applied to every UI object.

UISTATUS

```
ERR_NO_MEMORY
ERR_POSITION
ERR_OVER_LIMIT
ERR_OBJECT_HANDLE
ERR_STYLE
ERR_OBJECT_ID
ERR_ITEM_ID
ERR_NOT_SUPPORTED
ERR_INVALID_PARAMETER
ERR_UNEXEC_COMMAND
ERR_NO_MATCH_ISSUE
ERR_FILE_OPEN
ERR_FILE_CONTAIN
```

UI_SUCCESS

The following are possible values of *state*:

```
0                Without frame, and selected area won't be reserved.
UI_RESERVED      Without frame, and selected area will be reserved.
UI_FRAME|UI_HORIZONTAL_LINE|UI_VERTICAL_LINE
                With frame, and selected area won't be reserved.
UI_FRAME|UI_HORIZONTAL_LINE|UI_VERTICAL_LINE|UI_RESERVED
                With frame, and selected area will be reserved.
UI_3D            With 3D frame, and selected area will be reserved.
UI_RESERVED|UI_3D With 3D frame, and selected area will be reserved.
```

The following are possible values of *style*:

```
UI_FRAME        FRAME
UI_HORIZONTAL_LINE  HORIZONTAL LINE
UI_VERTICAL_LINE  VERTICAL LINE
UI_3D           3D
UI_RESERVED     Highlight RESERVED
```

Related Data Structure

The following data structures are used for some of the UI objects.

mfUI Structure

This structure stores all information of UIOBJ.

```
typedef struct tagmfUI {
    BOOL (*ProcessMsg)(void *This, SysMsg *pMsg);    // Stores the message.
    void (*Delete)(void *This);                    // Stores the delete information.
```



```

        void (*Draw)(void *This);           // Stores the draw information.
        void (*Clean)(void *This);          // Stores the clean information.
    } mfUI;

```

UIOBJ Structure

This structure stores all information of UI object.

```

typedef struct tagUIOBJ {
    const mfUI *mf;           // The mfUI Structure.
    WORD id;                  // The ID of UIOBJ.
    DWORD dwUIState;          // The UI state for the UIOBJ.
} UIOBJ;

```

L_NODE Structure

This structure stores information of an object node.

```

typedef struct L_NODE {
    WORD id;                  // The ID for each item
    char *pText;              // The Context of each item.
} L_NODE;

```

System Message

No system message for general UI object.

BroadcastMsg2Cont

Synopsis

```
void BroadcastMsg2Cont(WORD id, SysMsg *msg);
```

Description

Broadcasts a message to other UI objects.

<i>id</i>	Specifies the UI object ID of the sender.
* <i>msg</i>	Specifies the system message to be broadcasted.

Returned Value

None.

ClearActiveParent

Synopsis

```
void ClearActiveParent(void);
```

Description

Disables the active parent object.

Returned Value

None.

ClearHookUIMsg

Synopsis

```
void ClearHookUIMsg(void);
```

Description

Stops hooking UI object.

Returned Value

None.

EnableDirectMsg

Synopsis

```
void EnableDirectMsg(BOOL isEnable);
```

Description

Enables or disables the ProcessUIMsgDirect() function.

<i>isEnable</i>	Specifies enabling flag. TRUE for enabling; FALSE for disabling.
-----------------	--

Returned Value

None.

ForwardMsg2Cont

Synopsis

```
void ForwardMsg2Cont(WORD id, SysMsg *msg);
```

Description

Forwards a message to other UI objects.

<i>id</i>	Specifies the UI object ID of the sender.
* <i>msg</i>	Specifies the system message to be forwarded.

Returned Value

None.

GetCoverRect

Synopsis

```
int GetCoverRect(CoverRect **cRect);
```

Description

Obtains current UI objects' expanding area.

** <i>cRect</i>	Specifies the array to receive current status.
-----------------	--

Returned Value

The returned value is the number of areas being occupied.

GetUILastError

Synopsis

```
UISTATUS GetUILastError(void);
```

Description

Obtains the message of last action of a UI object.

Returned Value

The returned value is the message of last action of a UI object.

HookUIMsg

Synopsis

```
void HookUIMsg(void *hookObj);
```

Description

Hooks messages of UI object sent by ProcessUIMsgDirect().

**hookObj* Specifies the UI object.

Returned Value

None.

IsOnCoverRect

Synopsis

```
BOOL IsOnCoverRect(int x, int y);
```

Description

Check whether the specify position is in the area of this UI object.

<i>x</i>	Specifies x-coordinate of the specific position in pixels.
<i>y</i>	Specifies y-coordinate of the specific position in pixels.

Returned Value

The returned value is TRUE is this position is in the area. Otherwise, the returned value is FALSE.

IsOnCoverArea

Synopsis

```
BOOL IsOnCoverArea(int x, int y, WORD w, WORD h);
```

Description

Check whether the specify position is in the area of this UI object.

<i>x</i>	Specifies x-coordinate of the specific position in pixels.
<i>y</i>	Specifies y-coordinate of the specific position in pixels.
<i>w</i>	Specifies the width of the area to be checked in pixels.
<i>h</i>	Specifies the height of the area to be checked in pixels.

Returned Value

The returned value is TRUE is this position is in the area. Otherwise, the returned value is FALSE.

ProcessUIMsgDirect

Synopsis

```
BOOL ProcessUIMsgDirect(SysMsg *msg);
```

Description

Sends messages while UI object stays in loop.

<i>*msg</i>	Specifies the system message to be sent.
-------------	--

Returned Value

The returned value is TRUE if messages are processed. Otherwise, the returned value is FALSE.

ReleaseCoverRect

Synopsis

```
void ReleaseCoverRect(void);
```

Description

Releases the first defined expanding area.

Returned Value

None.

RlsLastCoverRect

Synopsis

```
void RlsLastCoverRect(void);
```

Description

Releases the last defined expanding area.

Returned Value

None.

SetActiveParent

Synopsis

```
void SetActiveParent(void *parent);
```

Description

Sets a UI object to be an active parent object. Parent object is an object which will hook all messages to child objects.

**parent* Specifies the UI object.

Returned Value

None.

SetCoverRect

Synopsis

```
CoverRect *SetCoverRect(int x1, int y1, int x2, int y2);
```

Description

Sets expanding area for object to be identified.

x1 Specifies x-coordinate of the left side of this object, in pixels.
y1 Specifies y-coordinate of the top side of this object, in pixels.
x2 Specifies x-coordinate of the right side of this object, in pixels.

y2 Specifies y-coordinate of the bottom side of this object, in pixels.

Returned Value

The returned value is a pointer of CoverRect structure.

uiClean

Synopsis

```
void uiClean(UIOBJ *object);
```

Description

Sets the object to be invisible. But this object still work in background.

**object* Specifies the object to be cleaned.

Returned Value

None.

uiDelete

Synopsis

```
void uiDelete(UIOBJ *object);
```

Description

Removes the object from the container.

**object* Specifies the object to be removed.

Returned Value

None.

uiDeleteTimer

Synopsis

```
BOOL uiDeleteTimer(UITMR_HANDLE timer);
```

Description

Removes a UI timer object.

timer Specifies the timer object.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

uiEnable

Synopsis

```
void uiEnable(UIOBJ *object, BOOL isEnabled);
```

Description

Enables or disables the object.

<i>*object</i>	Specifies the object .
<i>isEnabled</i>	Specifies the setting. TRUE for enabling; FALSE for disabling.

Returned Value

None.

uiEnableTimer

Synopsis

```
BOOL uiEnableTimer(UITMR_HANDLE timer, BOOL isEnabled);
```

Description

Enables or disables the timer object.

<i>timer</i>	Specifies the timer object.
<i>isEnabled</i>	Specifies enabling flag. TRUE for enabling; FALSE for disabling.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

UIExtraProcess

Synopsis

```
BOOL UIExtraProcess(WORD id, SysMsg *msg);
```

Description

Posts messages to system at the end of its loop.

<i>id</i>	Specifies the UI object ID of the sender.
<i>*msg</i>	Specifies the system message.

Returned Value

The returned value is TRUE if messages are processed. Otherwise, the returned value is FALSE.

uiIsActive

Synopsis

```
BOOL uiIsActive(UIOBJ *object);
```

Description

Obtains the status of a UI object.

<i>*object</i>	Specifies the UI object.
----------------	--------------------------

Returned Value

The returned value is TRUE if this UI object is active. Otherwise, the returned value is FALSE.

uiGetActive

Synopsis

```
UIOBJ *uiGetActive(void);
```

Description

Obtains the active UI object.

Returned Value

The returned value is a pointer of UIOBJ if this function succeeds.

uiIsEnable

Synopsis

```
BOOL uiIsEnable(UIOBJ *object);
```

Description

Obtains the status of a UI object.

**object* Specifies the UI object.

Returned Value

The returned value is TRUE if this UI object is enabled. Otherwise, the returned value is FALSE.

uiIsVisible

Synopsis

```
BOOL uiIsVisible(UIOBJ *object);
```

Description

Obtains the status of a UI object.

**object* Specifies the UI object.

Returned Value

The returned value is TRUE if this UI object is visible. Otherwise, the returned value is FALSE.

uiNewTimer

Synopsis

```
UITMR_HANDLE uiNewTimer(WORD id);
```

Description

Creates a UI timer object.

id Specifies the timer ID.

Returned Value

The returned value is a handle of timer object if this function succeeds. Otherwise, the returned value is NULL.

uiRedraw

Synopsis

```
void uiRedraw(UIOBJ *object);
```

Description

Redraws the object in the container.

**object* Specifies the object to be redrawed.

Returned Value

None.

uiSetActive

Synopsis

```
void uiSetActive(UIOBJ *object);
```

Description

Activates the specific UI object.

**object* Specifies the UI object.

Returned Value

None.

uiSetInterval

Synopsis

```
DWORD uiSetInterval(UITME_HANDLE timer, DWORD interval);
```

Description

Sets the interval of the timer object.

timer Specifies the timer object.

interval Specifies interval of the time object in millisecond.

Returned Value

The returned value is a non-zero value if this function succeeds. Otherwise, the returned value is zero.

uiSetLastError

Synopsis

```
void uiSetLastError(UISTATUS errNo);
```

Description

Sets the last error codes for UI objects.

errNo Specifies the error number.

Returned Value

None.

uiSetVisible

Synopsis

```
void uiSetVisible(UIOBJ *object, BOOL isVisable);
```

Description

Sets the object to be visible or invisible.

**object* Specifies the object.

isVisable Specifies the visable setting. TRUE for visable; FALSE for invisible.

Returned Value

None.

Label

General Description

Displays text string on the screen.

Related Data Structure

UIOBJ Structure

System Message

No system message for label retrieving.

labClearCaption

Synopsis

```
void labClearCaption(UIOBJ *label);
```

Description

Clears the text of label.

**label* Specifies the label object.

Returned Value

None.

labGetCaption

Synopsis

```
char *labGetCaption(UIOBJ *label);
```

Description

Obtains the text of label.

**label* Specifies the label object.

Returned Value

The returned value is the pointer of the label text if the function succeeds. Otherwise, the returned value is NULL.

labNew

Synopsis

```
UIOBJ *labNew(WORD id, int x, int y, int w, int h, void *caption, WORD style,  
              WORD captionPos, BYTE fontId, BYTE frame);
```

Description

Creates a new label object.

<i>id</i>	Specifies the unique ID number of this label object.
<i>x</i>	Specifies x-coordinate of left side of this label object, in pixels.
<i>y</i>	Specifies y-coordinate of top side of this label object, in pixels.
<i>w</i>	Specifies the width of this label object.
<i>h</i>	Specifies the height of this label object.
<i>*caption</i>	Specifies the caption of the label object.
<i>style</i>	Specifies the style of the label object. The following are possible values: LABSTYLE_STRING The caption is a string. LABSTYLE_BMP The caption is a Bit MAP
<i>captionPos</i>	Specifies the caption position of the label object.
<i>fontId</i>	Specifies the font used by label. The available fonts so far are GM_FONT_SMALL, GM_FONT_SMALLBOLD, GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD, and GM_FONT_LARGE. Only GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD, and GM_FONT_LARGE can be applied to Chinese characters.
<i>frame</i>	Specifies frame option of the label object.

- 1 The label has frame.
- 0 The label doesn't have frame.

Returned Value

The returned value is a pointer of UIOBJ if this function succeeds. Otherwise, the returned value is NULL.

labNew3DEx

Synopsis

```
UIOBJ *labNew3DEx(WORD id, int x, int y, int w, int h, const void *caption, WORD style,
                  WORD captionPos, BYTE fontId, BYTE frame, BOOL isThreeD);
```

Description

Creates a new label object.

<i>id</i>	Specifies the unique ID number of this label object.
<i>x</i>	Specifies x-coordinate of left side of this label object, in pixels.
<i>y</i>	Specifies y-coordinate of top side of this label object, in pixels.
<i>w</i>	Specifies the width of this label object.
<i>h</i>	Specifies the height of this label object.
<i>*caption</i>	Specifies the caption of the label object.
<i>style</i>	Specifies the style of the label object. The following are possible values: LABSTYLE_STRING The caption is a string. LABSTYLE_BMP The caption is a Bit MAP
<i>captionPos</i>	Specifies the caption position of the label object.
<i>fontId</i>	Specifies the font used by label. The available fonts so far are GM_FONT_SMALL, GM_FONT_SMALLBOLD, GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD, and GM_FONT_LARGE. Only GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD, and GM_FONT_LARGE can be applied to Chinese characters.
<i>frame</i>	Specifies frame option of the label object. 1 The label has frame. 0 The label doesn't have frame.
<i>isThreeD</i>	Specifies whether or not the style of the label object is three dimension.

Returned Value

The returned value is a pointer of UIOBJ if this function succeeds. Otherwise, the returned value is NULL.

labSetCaption

Synopsis

```
void labSetCaption(UIOBJ *label, char *caption);
```

Description

Sets the text of label.

<i>*label</i>	Specifies the label object.
<i>*caption</i>	Specifies the caption of the label object.

Returned Value

None.

labSetThreeD

Synopsis

```
void labSetThreeD(UIOBJ *label, BOOL isThreeD);
```

Description

Sets the three dimensional frame state of the label object.

<i>*label</i>	Specifies the label object.
<i>isThreeD</i>	Specifies the three dimension frame state of the label object.
TRUE	3D frame
FALSE	Normal frame

Returned Value

None.

Button

General Description

Displays a button on the screen and obtains the related system message to do some reaction.

Related Data Structure

UIOBJ Structure

System Message

MT_BUTTON_COMMAND	Presses the button object.
pMsg->id	ID of this button.
pMsg->data.pos.x	The x-coordinate of pen down' s position.
pMsg->data.pos.y	The y-coordinate of pen down' s position.
MT_BUTTON_FOCUS_IN	Move pen on touch panel, and let the button object get focus.
pMsg->id	ID of this button.
pMsg->data.pos.x	The x-coordinate of pen down' s position.
pMsg->data.pos.y	The y-coordinate of pen down' s position.
MT_BUTTON_FOCUS_OUT	Move pen on touch panel, and let the button object lose focus.
pMsg->id	ID of this button.
pMsg->data.pos.x	The x-coordinate of pen down' s position.
pMsg->data.pos.y	The y-coordinate of pen down' s position.
MT_BUTTON_REPEAT	Holds on the button object, and don' t move.
pMsg->id	ID of this button.
pMsg->data.pos.x	The x-coordinate of pen down' s position.
pMsg->data.pos.y	The y-coordinate of pen down' s position.

btnClearCaption

Synopsis

```
void btnClearCaption(UIOBJ *button);
```

Description

Clears button caption.

**button* Specifies the button object.

Returned Value

None.

btnFrame

Synopsis

```
void btnFrame(UIOBJ *button, BYTE isFrame);
```

Description

Sets a button object with or without frame.

**button* Specifies the button object.
isFrame Specifies the frame flag of button object.
 BNTFRAME_YES Set button with frame
 BNTFRAME_NO Set button without frame

Returned Value

None.

btnGetCaption

Synopsis

```
char *btnGetCaption (UIOBJ *button);
```

Description

Obtains the caption of the button object.

**button* Specifies the button object.

Returned Value

The returned value is the character pointer of this caption of button if this function succeeds. Otherwise, the returned value is zero.

btnGetCheckBoxState

Synopsis

```
BOOL btnGetCheckBoxState (UIOBJ *button);
```


Description

Obtains the state of the checkbox button.

**button* Specifies the button object.

Returned Value

The returned value is TRUE if this checkbox button was checked. Otherwise, the returned is FALSE.

btnNew

Synopsis

```
UIOBJ *btnNew(WORD id, int x, int y, int w, int h, const void *caption, WORD style);
```

Description

Creates a button object.

<i>id</i>	Specifies the unique ID number of this button object.
<i>x</i>	Specifies x-coordinate of left side of this button object, in pixels.
<i>y</i>	Specifies y-coordinate of top side of this button object, in pixels.
<i>w</i>	Specifies the width of this button object.
<i>h</i>	Specifies the height of this button object.
<i>*caption</i>	Specifies the caption of the button object.
<i>style</i>	Specifies the style of the button caption. LABSTYLE_STRING character string LABSTYLE_BMP BMP

Returned Value

The returned value is a pointer of UIOBJ.

btnNew3DEx

Synopsis

```
UIOBJ *btnNew3DEx(WORD id, int x, int y, int w, int h, const void *caption, WORD style,  
WORD captionPos, BYTE fontId, WORD kind, BOOL isThreeD, BYTE frame);
```

Description

Creates a new button object.

<i>id</i>	Specifies the unique ID number of this button object.
<i>x</i>	Specifies x-coordinate of left side of this button object, in pixels.
<i>y</i>	Specifies y-coordinate of top side of this button object, in pixels.
<i>w</i>	Specifies the width of this button object.
<i>h</i>	Specifies the height of this button object.
<i>*caption</i>	Specifies the caption of the button object.
<i>style</i>	Specifies the style of the button object. The following are possible values: LABSTYLE_STRING The caption is a string. LABSTYLE_BMP The caption is a Bit MAP
<i>captionPos</i>	Specifies the caption position of the button object.
<i>fontId</i>	Specifies the font used by button. The available fonts so far are GM_FONT_SMALL, GM_FONT_SMALLBOLD, GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD, and

	GM_FONT_LAGE. Only GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD, and GM_FONT_LARGE can be applied to Chinese characters.
<i>kind</i>	Specifies the frame style of the label object.
<i>isThreeD</i>	Specifies whether or not the frame of the button object is three dimension.
<i>frame</i>	Specifies the frame option of the button object.
	1 The button object has frame.
	0 The button object doesn't have frame.

Returned Value

The returned value is a pointer of UIOBJ.

btnNewEx

Synopsis

```
UIOBJ *btnNew(WORD id, int x, int y, int w, int h, const void *caption, WORD style,
              WORD strPos, BYTE fontId, WORD kind);
```

Description

Creates a button object.

<i>id</i>	Specifies the unique ID number of this button object.
<i>x</i>	Specifies x-coordinate of left side of this button object, in pixels.
<i>y</i>	Specifies y-coordinate of top side of this button object, in pixels.
<i>w</i>	Specifies the width of this button object.
<i>h</i>	Specifies the height of this button object.
<i>*caption</i>	Specifies the content of the button caption.
<i>style</i>	Specifies the style of the button caption. LABSTYLE_STRING character string LABSTYLE_BMP BMP
<i>captionPos</i>	Specifies the caption position of the button object. The possible values are BTNSTRPOS_RIGHT, BTNSTRPOS_MIDDLE or BTNSTRPOS_MIDDLE.
<i>fontId</i>	Specifies the ID number of fonts.
<i>kind</i>	Specifies the form of the button frame.

Returned Value

The returned value is a pointer of UIOBJ.

btnSetBtnGroup

Synopsis

```
void btnSetBtnFroup(UIOBJ *button, WORD groupId);
```

Description

Adds the button to the group.

<i>*button</i>	Specifies the button object.
<i>groupId</i>	Specifies the number for the group ID.

Returned Value

None.

btnSetCaption

Synopsis

```
void btnSetCaption(UIOBJ *button, const char *caption);
```

Description

Changes button caption.

**button* Specifies the button object.
**caption* Specifies the data pointer of the button caption.

Returned Value

None.

btnSetCheckBoxState

Synopsis

```
void btnSetCheckBoxState(UIOBJ *button, BOOL state);
```

Description

Sets the state of the checkbox button.

**button* Specifies the button object.
state Specifies the state of checkbox button object.
 CheckBtnState_Select (TRUE) Set checkbox button checked
 CheckBtnState_NoSelect (FALSE) Set checkbox button unchecked

Returned Value

None.

btnSetRadioBtnGroup

Synopsis

```
void btnSetRadioBtnGroup(UIOBJ *button, WORD groupId);
```

Description

Sets a radio button to a specific group.

**button* Specifies the button object.
groupId Specifies the unique button group ID number.

Returned Value

None.

btnSetStrPosition

Synopsis

```
void btnSetStrPosition(UIOBJ *button, WORD captionPos);
```

Description

Sets the caption position of the button object.

<i>*button</i>	Specifies the button object.
<i>captionPos</i>	Specifies the caption position of the button object. The following are possible values:
BTN_STRPOS_RIGHT	The position string is on the right side of the button.
BTN_STRPOS_LEFT	The position string is on the left side of the button.
BTN_STRPOS_MIDDLE	The position string is in the middle of the button.

Returned Value

None.

btnSetRepeat

Synopsis

```
void btnSetRepeat(UIOBJ *button, BOOL isRepeat, int time);
```

Description

Sets the repeat interval of button object.

<i>*button</i>	Specifies the button object.
<i>isRepeat</i>	Specifies the repeat flag of button object.
<i>time</i>	Specifies the repeat time interval in 10 ms of button object. When button object has been pressed for over <i>time</i> , it will be taken as pressing or clicking twice.

Returned Value

None.

btnSetThreeD

Synopsis

```
void btnSetThreeD(UIOBJ *button, BOOL isThreeD);
```

Description

Sets the button object as a 3D button.

<i>*button</i>	Specifies the button object.
<i>isThreeD</i>	Specifies the flag of 3D button object.
TRUE	Set button 3D
FALSE	Set button not 3D

Returned Value

None.

List

General Description

List is composed of one or more than one items, but only one item can be active. To make the item active, user can click on it. Each item has its ID assigned by user and the item text displayed in the list. Some of the functions return 0xFFFF, which means that the data type of the returned value is unsigned (short/long) integer and the bit value is the same with -1 of signed integer. The difference between list and combo box is that list allows more than one item in the list window but the combo box allows only one.

Some character pattern can be added preceding item text. The following are these patterns.

%h <i>hot-key</i> %	This pattern sets hot-key text and must start with “%h” and end with another “%”. Between the percent pair, it is <i>hot-key</i> which is displayed at the right side of this item.
%-	This pattern inserts a separator line “===” in list. This line is also taken as an item, and it also has an item ID. Other patterns will be skipped if this one is used.
%g	This pattern sets the item text displayed in gray.
%n	<i>n</i> is the margin in pixels between the item text and the list frame.
%l	Sets the item text to align to left. If %n is used with %l, the <i>n</i> is the margin in pixels between the starting point of item text and the list frame.
%r	Sets the item text to align to right. If %n is used with %r, the <i>n</i> is the margin in pixels between the ending point of item text and the list frame.
%m	Sets the item text to align to middle.
%t	Sets the item text to align to top.
%b	Sets the item text to align to bottom.

Here are some examples to use these patterns.

ITEM TEXT	REAL OUTPUT
“%hCtrl+A%%g%5%l%mTest”	“ Test Ctrl+A”
“%hCtrl+A%g%5%l%mTest”	“ g%5%l%mTest Ctrl+A” (not as expected)
“%hCtrl+A%%r%5%Test”	“ Ctrl+A” (not as expected, text is overwritten by hot-key)
“%5Test%hCtrl+A”	“ Test%hCtrl+A” (not as expected)
“%5%-Ctrl+A”	“ =====”
“%%51234”	“ %51234” (not as expected)
“%51234”	“ 1234”

The default pattern is “%l%m%1”.

Related Data Structure

UIOBJ Structure

L_NODE Structure

System Message

MT_LIST_COMMAND	Select an item.
pMsg->id	ID of this list.
pMsg->data	data.w is the active item ID.
MT_LIST_FOCUS_IN	Pen position is entering the list.
pMsg->id	ID of this list.
pMsg->data	data.w is item ID to which the pen points.
MT_LIST_FOCUS_OUT	Pen position is leaving the list.
pMsg->id	ID of this list.

pMsg->data	data.w is item ID to which the pen points.
MT_LIST_FOCUS_MOVE	Pen position is moving in the list.
pMsg->id	ID of this list.
pMsg->data	data.w is item ID to which the pen points.

GetItemString

Synopsis

```
char *GetItemString(UIOBJ *list, WORD id);
```

Description

Obtains the item text by item ID. The difference between listGetItemText() and GetItemString() is the returned value. Since GetItemString() returns 0 either the item ID is 0 or this function fails which may cause ambiguity, listGetItemText() is recommended for obtaining the ID of the active item.

**list* Specifies the list object.
id Specifies the item ID in list.

Returned Value

The returned value is the item text if this function succeeds. Otherwise, the returned value is 0. GetUILastError() can be called to obtain detailed failure information.

listAddItem

Synopsis

```
BOOL listAddItem(UIOBJ *list, WORD id, char *itemText);
```

Description

Adds a new item to list object.

**list* Specifies the list object.
id Specifies the item ID in list.
**itemText* Specifies the text of this item. Some character patterns can be put into preceding text.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

listAddItemArray

Synopsis

```
BOOL listAddItemArray(UIOBJ *list, L_NODE *item, int itemNum);
```

Description

Adds one or more than one items to list object.

**list* Specifies the list object.
**item* Specifies the L_NODE which points to an item array.
itemNum Specifies the number of items to be added.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

listAddSpaceBar

Synopsis

```
BOOL listAddSpaceBar (UIOBJ *list, WORD id);
```

Description

Adds a separator ("=====") into list.

**list* Specifies the list object.
id Specifies the item ID of the separator in list.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

listGetItemActive

Synopsis

```
WORD listGetItemActive (UIOBJ *list);
```

Description

Obtains the ID of the active item.

**list* Specifies the list object.

Returned Value

The returned value is the ID of the active item if this function succeeds. Otherwise, the returned value is 0xFFFF.

listGetItemArray

Synopsis

```
L_NODE *listGetItemArray (UIOBJ *list, L_NODE *item, int itemNum);
```

Description

Obtains information of each item in list, including item ID and text. User has to prepare the memory space to which the L_NODE array is copied.

**list* Specifies the list object.
**item* Specifies the an L_NODE array which points to a memory space prepared by user.
The size of the memory space must be equal to *itemNum* * sizeof(L_NODE).
itemNum Specifies the number of items user wants to retrieve.

Returned Value

The returned value is information of items if this function succeeds. Otherwise, the returned value is NULL.

listGetHotKeyText

Synopsis

```
char *listGetHotKeyText (UIOBJ *list, WORD id);
```


Description

Obtains the hot-key text by item ID.

**list* Specifies the list object.
id Specifies the item ID in list.

Returned Value

The returned value is the hot-key text if the hot-key exists, and is NULL if the hot-key does not exist. Otherwise, the returned value is NULL.

listGetItemPosition

Synopsis

```
char *listGetItemPosition(UIOBJ *list, WORD id);
```

Description

Obtains the character pattern of position settings by item ID, not including item text and hot-key text.

**list* Specifies the list object.
id Specifies the item ID in list.

Returned Value

The returned value is the character pattern of position settings if this function succeeds. Otherwise, the returned value is NULL.

listGetItemText

Synopsis

```
char *listGetItemText(UIOBJ *list, WORD id);
```

Description

Obtains the item text by item ID.

**list* Specifies the list object.
id Specifies the item ID in list.

Returned Value

The returned value is the item text if this function succeeds. Otherwise, the returned value is NULL.

listGetTopItem

Synopsis

```
WORD listGetTopItem(UIOBJ *list);
```

Description

Obtains the first item of the visible window.

**list* Specifies the list object.

Returned Value

The returned value is item ID of first item if this function succeeds. Otherwise, the returned value is 0xFFFF.

listIsGrayItem

Synopsis

```
BOOL listIsGrayItem(UIOBJ *list, WORD id, BOOL isGray);
```

Description

Sets item enabled or disabled. If the item is set disabled, the item is in gray color and has no response even if it is clicked.

<i>*list</i>	Specifies the list object.
<i>id</i>	Specifies the item ID in list.
<i>isGray</i>	Specifies the item status to set.
TRUE	Enable the item
FALSE	Disable the item

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

listLastActive

Synopsis

```
WORD listLastActive(UIOBJ *list);
```

Description

Obtains the ID of the active item. The difference between listGetItemActive() and listLastItem() is the returned value. Since listLastItem() returns 0 either the item ID is 0 or this function fails which may cause ambiguity, listGetItemActive() is recommended for obtaining the ID of the active item.

**list* Specifies the list object.

Returned Value

The returned value is the ID of the active item if this function succeeds. Otherwise, the returned value is 0. GetUILastError() can be called to obtain detailed failure information.

listMoveLine

Synopsis

```
WORD listMoveLine(UIOBJ *list, BOOL isDown);
```

Description

Moves up or down one line of the list.

<i>*list</i>	Specifies the list object.
<i>isDown</i>	Specifies the direction to move.
TRUE	Move down
FALSE	Move up

Returned Value

The returned value is the ID of the first item in the list if this function succeeds. Otherwise, the returned value is 0. GetUILastError() can be called to obtain detailed failure information.

listMovePage

Synopsis

```
WORD listMovePage(UIOBJ *list, BOOL isDown);
```

Description

Moves up or down one page of the list.

<i>*list</i>	Specifies the list object.
<i>isDown</i>	Specifies the direction to move.
TRUE	Move down
FALSE	Move up

Returned Value

The returned value is the ID of the first item displayed in the list if this function succeeds. Otherwise, the returned value is 0. GetUILastError() can be called to obtain detailed failure information.

listNew

Synopsis

```
UIOBJ *listNew(WORD id, int x, int y, int w, BYTE itemInList, BYTE fontId, BYTE style);
```

Description

Creates a list object. Scroll bar is automatically generated if the number of items is more than the number of visible items in the list.

<i>id</i>	Specifies the object ID of this list.
<i>x</i>	Specifies x-coordinate of left side of this list object, in pixels.
<i>y</i>	Specifies y-coordinate of top side of this list object, in pixels.
<i>w</i>	Specifies the width of this list object in pixels. The minimal value is 9.
<i>itemInList</i>	Specifies the number of visible items in the list.
<i>fontId</i>	Specifies the font used by list. The available fonts so far are GM_FONT_SMALL, GM_FONT_SMALLBOLD, GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD, and GM_FONT_LARGE. Only GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD, and GM_FONT_LARGE can be applied to Chinese characters.
<i>style</i>	Specifies the style looking of the list. The following are possible style and only one style can be applied at the same time.
LIST_NONE	This is used when user does not want to set style at this moment.
LIST_BAR_RESERVE	The active item is in white text and black background
LIST_BAR_TEMPORAL	The active item is in white text and black background only

when pen points to that item.
 LIST_BAR_FRAME_ITEM The active item is framed like 3D.
 LIST_BAR_UNDERLINE All items have underline.

Returned Value

The returned value is the list object if this function succeeds. Otherwise, the returned value is NULL.

listNewEx

Synopsis

```
UIOBJ *listNewEx(WORD id, int x, int y, int w, BYTE itemInList, int itemHeight, BYTE fontId,  

  BYTE style, DWORD state);
```

Description

Creates a list object.

<i>id</i>	Specifies the ID of this list object.
<i>x</i>	Specifies x-coordinate of left side of this list object, in pixels.
<i>y</i>	Specifies y-coordinate of top side of this list object, in pixels.
<i>w</i>	Specifies the width of this list object in pixels. The minimal value is 25.
<i>itemInList</i>	Specifies the number of visible items in the list.
<i>itemHeight</i>	Specifies the height of each item in pixels.
<i>fontId</i>	Specifies the font used by list. The available fonts so far are GM_FONT_SMALL, GM_FONT_SMALLBOLD, GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD, and GM_FONT_LAGE. Only GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD, and GM_FONT_LARGE can be applied to Chinese characters.
<i>style</i>	Specifies the style looking of the list. The following are possible style and only one style can be applied at the same time. LIST_NONE This is used when user does not want to set style at this moment. LIST_BAR_RESERVE The active item is in white text and black background LIST_BAR_TEMPORAL The active item is in white text and black background only when pen points to that item. LIST_BAR_FRAME_ITEM The active item is framed like 3 D. LIST_BAR_UNDERLINE All items have underline.
<i>state</i>	Specifies the appearance (UI) state of list. 0 No frame. UI_FRAME With frame. UI_3D 3D frame. To let 3D effective, UI_FRAME has to be applied.

Returned Value

The returned value is the list object if this function succeeds. Otherwise, the returned value is NULL.

listPopUp

Synopsis

```
WORD listPopUp(int x, int y, int w, BYTE itemInList, BYTE fontId, int itemNum,  

  L_NODE *item);
```

Description

Pops up a list-like window, but not really creates a list object. After one of the list items is selected, the list-like window disappears. The window appears without having to call `ContRedraw()`. The difference between `listPopUp()` and `listPopUpPro()` is the returned value. Since `listPopUp()` returns 0 either the item ID is 0 or this function fails which may cause ambiguity, `listPopUpPro()` is recommended for obtaining the ID of the active item.

<i>x</i>	Specifies x-coordinate of left side of this list object, in pixels.
<i>y</i>	Specifies y-coordinate of top side of this list object, in pixels.
<i>w</i>	Specifies the width of this list object in pixels. The minimal value is 25.
<i>itemInList</i>	Specifies the number of visible items in the list.
<i>fontId</i>	Specifies the font used by list. The available fonts so far are GM_FONT_SMALL, GM_FONT_SMALLBOLD, GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD, and GM_FONT_LAGE. Only GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD, and GM_FONT_LARGE can be applied to Chinese characters.
<i>itemNum</i>	Specifies the number of items user wants to retrieve.
<i>*item</i>	Specifies the an L_NODE array which points to a memory space prepared by user. The size of the memory space must be equal to <i>itemNum</i> * sizeof(L_NODE).

Returned Value

The returned value is the active item number if this function succeeds. Otherwise, the returned value is 0. `GetUILastError()` can be called to obtain detailed failure information.

listPopUpEx

Synopsis

```
WORD listPopUpEx(int x, int y, int w, BYTE itemInList, BYTE fontId, int itemNum,  
                L_NODE *item, int actItem);
```

Description

Pops up a list-like window, but not really creates a list object. After one of the list items is selected, the list-like window disappears. The window appears without having to call `ContRedraw()`. The difference between `listPopUp()` and `listPopUpPro()` is the returned value. Since `listPopUp()` returns 0 either the item ID is 0 or this function fails which may cause ambiguity, `listPopUpPro()` is recommended for obtaining the ID of the active item.

<i>x</i>	Specifies x-coordinate of left side of this list object, in pixels.
<i>y</i>	Specifies y-coordinate of top side of this list object, in pixels.
<i>w</i>	Specifies the width of this list object in pixels. The minimal value is 25.
<i>itemInList</i>	Specifies the number of visible items in the list.
<i>fontId</i>	Specifies the font used by list. The available fonts so far are GM_FONT_SMALL, GM_FONT_SMALLBOLD, GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD, and GM_FONT_LAGE. Only GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD, and GM_FONT_LARGE can be applied to Chinese characters.
<i>itemNum</i>	Specifies the number of items user wants to retrieve.
<i>*item</i>	Specifies the an L_NODE array which points to a memory space prepared by user. The size of the memory space must be equal to <i>itemNum</i> * sizeof(L_NODE).
<i>actItem</i>	Specifies the active item number. The item number starts from 0 to <i>itemNum</i> –1.

Returned Value

The returned value is the active item number if this function succeeds. Otherwise, the returned

value is 0. GetUILastError() can be called to obtain detailed failure information.

listPopUpPro

Synopsis

```
WORD listPopUpPro(int x, int y, int w, BYTE itemInList, int itemHeight, BYTE fontId,  
                  int itemNum, L_NODE *item, int actItem, DWORD state);
```

Description

Pops up a list-like window, but not really creates a list object. After one of the list items is selected, the list-like window disappears. The window appears without having to call ContRedraw().

<i>x</i>	Specifies x-coordinate of left side of this list object, in pixels.
<i>y</i>	Specifies y-coordinate of top side of this list object, in pixels.
<i>w</i>	Specifies the width of this list object in pixels. The minimal value is 25.
<i>itemInList</i>	Specifies the number of visible items in the list.
<i>itemHeight</i>	Specifies the height of each item in pixels.
<i>fontId</i>	Specifies the font used by list. The available fonts so far are GM_FONT_SMALL, GM_FONT_SMALLBOLD, GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD, and GM_FONT_LARGE. Only GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD, and GM_FONT_LARGE can be applied to Chinese characters.
<i>itemNum</i>	Specifies the number of items user wants to retrieve.
<i>*item</i>	Specifies the an L_NODE array which points to a memory space prepared by user. The size of the memory space must be equal to <i>itemNum</i> * sizeof(L_NODE).
<i>actItem</i>	Specifies the active item number. The item number starts from 0 to <i>itemNum</i> -1.
<i>state</i>	Specifies the appearance (UI) state of list. 0 No frame. UI_FRAME With frame. UI_3D 3D frame. To let 3D effective, UI_FRAME has to be applied.

Returned Value

The returned value is the active item number if this function succeeds. Otherwise, the returned value is 0xFFFF.

listRealHeight

Synopsis

```
int listRealHeight(UIOBJ *list);
```

Description

Obtains the height of list in pixels, including frame (1 pixel) if it does have.

**list* Specifies the list object.

Returned Value

The returned value is the height of list if this function succeeds. Otherwise, the returned value is 0. GetUILastError() can be called to obtain detailed failure information.

listRemoveItem

Synopsis

```
int listRemoveItem(UIOBJ *list, WORD id);
```

Description

Removes an item from list object by item ID.

<i>*list</i>	Specifies the list object.
<i>id</i>	Specifies the item ID in list.

Returned Value

The returned value is the number of items of list after removal. Otherwise, the returned value is 0. GetUILastError() can be called to obtain detailed failure information.

listRemoveItemAll

Synopsis

```
int listRemoveItemAll(UIOBJ *list);
```

Description

Removes all items from list object.

<i>*list</i>	Specifies the list object.
--------------	----------------------------

Returned Value

The returned value is 1 if this function succeeds. Otherwise, the returned value is 0. GetUILastError() can be called to obtain detailed failure information.

listSetHotKeyText

Synopsis

```
BOOL listSetHotKeyText(UIOBJ *list, WORD id, char *itemText);
```

Description

Sets the hot key value. It is the same with the pattern “%hhot-key%”, but need not include the percent pair. Only hot-key text is needed in *itemText* and hot-key option is clear if *itemText* is NULL string.

<i>*list</i>	Specifies the list object.
<i>id</i>	Specifies the item ID in list.
<i>*itemText</i>	Specifies the text of this item. Not allow character patterns.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

listSetItemActive

Synopsis

```
BOOL listSetItemActive(UIOBJ *list, WORD id);
```

Description

Sets an item to be active (selected).

**list* Specifies the list object.
id Specifies the item ID in list object.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

listSetItemPosition

Synopsis

```
BOOL listSetItemPosition(UIOBJ *list, WORD id, char *itemText);
```

Description

Sets the text position. Character pattern can be used here, but not including “%h %”, “%-“ and “%g”.

**list* Specifies the list object.
id Specifies the item ID in list.
**itemText* Specifies the text of this item. Some character patterns can be put into preceding text.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

listSetItemText

Synopsis

```
BOOL listSetItemText(UIOBJ *list, WORD id, char *itemText);
```

Description

Modifies item text of list object.

**list* Specifies the list object.
id Specifies the item ID in list.
**itemText* Specifies the text of this item. Character patterns are not allowed.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

listSetTopItem

Synopsis

```
BOOL listSetTopItem(UIOBJ *list, WORD id);
```

Description

Moves the specified item to be the first item of the visible window.

**list* Specifies the list object.
id Specifies the item ID in list.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

listTotalItem

Synopsis

```
int listTotalItem(UIOBJ *list) ;
```

Description

Obtains the number of items of list object.

**list* Specifies the list object.

Returned Value

The returned value is the number of items of list if this function succeeds. Otherwise, the returned value is -1.

listTotItem

Synopsis

```
int listTotItem(UIOBJ *list) ;
```

Description

Obtains the number of items of list object. The difference between listTotItem() and listTotalItem() is the returned value. Since listTotItem() returns 0 either the number of items is 0 or this function fails which may cause ambiguity, listTotalItem() is recommended for obtaining the number of items.

**list* Specifies the list object.

Returned Value

The returned value is the number of items of list if this function succeeds. Otherwise, the returned value is 0. GetUILastError() can be called to obtain detailed failure information.

Table

General Description

Displays a table on the screen.

The following are possible values of *style* when creating a new table:

TS_NONE	Table without frame, and selected area won' t be reserved.
TS_RESERVE	Table without frame, and selected area will be reserved.
TS_FRAME	Table with frame, and selected area won' t be reserved.
TS_NORMAL	Table with frame, and selected area will be reserved.
TS_FRAME_3D	Table with 3D frame, and selected area won' t be reserved.
TS_NORMAL_3D	Table with 3D frame, and selected area will be reserved.

The following are possible value of *type* when creating a new table:

TS_STRING_LEFT
TS_STRING_MIDDLE
TS_STRING_RIGHT

Related Data Structure

UIOBJ Structure

System Message

MT_TABLE_COMMAND	Select an item.
pMsg->id	ID of this table
pMsg->data.pos.x	The row number of the item.
pMsg->data.pos.y	The column number of the item.
MT_TABLE_FOCUS_IN	Move pen on touch panel, and let this table object get focus.
pMsg->id	ID of this table
pMsg->data.pos.x	The row number of the item.
pMsg->data.pos.y	The column number of the item.
MT_TABLE_FOCUS_OUT	Move pen on touch panel, and let this button object lose focus.
pMsg->id	ID of this table.
pMsg->data.pos.x	The row number of the item.
pMsg->data.pos.y	The column number of the item.
MT_TABLE_RETURN	When item in table is selected (reserved), this item is drag by pen and moved to the area outside the table. When pen is leaving the screen, this message is generated.
pMsg->id	ID of this table.
pMsg->data.pos.x	The row number of the item.
pMsg->data.pos.y	The column number of the item.

tblBmpOut

Synopsis

```
void tblBmpOut(UIOBJ *table, int col, int row, LBmp *bmp);
```

Description

Sets the bit mat content of the cell in a table.

<i>*table</i>	Specifies the table object.
<i>col</i>	Specifies the column number of the table object.
<i>row</i>	Specifies the row number of the table object.
<i>*bmp</i>	Specifies the the bit map array.

Returned Value

None.

tblBmpNew

Synopsis

```
void tblBmpOut(UIOBJ *table, int col, int row, LBmp *bmp);
```

Description

(This function is reserved for future implementation.)

tblClearReserve

Synopsis

```
void tblClearReserve(UIOBJ *table);
```

Description

Clears the reserve area in a table.

<i>*table</i>	Specifies the table object.
---------------	-----------------------------

Returned Value

None.

tblGetColumn

Synopsis

```
long tblGetColumn(UIOBJ *table);
```

Description

Obtains the total column in the table.

<i>*table</i>	Specifies the table object.
---------------	-----------------------------

Returned Value

The returned value is the total column number of the table object.

tblGetReserve

Synopsis

```
void tblGetReserve(UIOBJ *table, int *col, int *row);
```

Description

Obtains the reserved area in a table.

<i>*table</i>	Specifies the table object.
<i>*col</i>	Specifies the memory buffer to which the column number of this table is copied.
<i>*row</i>	Specifies the memory buffer to which the row number of this table is copied.

Returned Value

None.

tblGetRow

Synopsis

```
long tblGetRow(UIOBJ *table);
```

Description

Obtains the total row in the table.

<i>*table</i>	Specifies the table object.
---------------	-----------------------------

Returned Value

The returned value is total row number of the table if this function succeeds.

tblInsertColumn

Synopsis

```
BOOL tblInsertColumn(UIOBJ *table, long col);
```

Description

Inserts a column into the table object.

<i>*table</i>	Specifies the table object.
<i>col</i>	Specifies the column to be inserted in the table object.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

tblInsertRow

Synopsis

```
BOOL tblInsertRow(UIOBJ *table, long row);
```

Description

Inserts a row into the table object.

<i>*table</i>	Specifies the table object.
<i>row</i>	Specifies the row to be inserted in the table object.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

tblMoveColumn

Synopsis

```
UINT tblMoveColumn(UIOBJ *table, BOOL isRight);
```

Description

Scrolls one column in the visible area of the table object.

<i>*table</i>	Specifies the table object.
<i>isRight</i>	Specifies the setting to scroll right or left for one column.
FALSE	Scroll left
TRUE	Scroll right

Returned Value

The returned value is the column number of the left cell of the table object if the function succeeds. Otherwise, the returned value is 0xFFFF.

tblMovePageColumn

Synopsis

```
UINT tblMovePageColumn(UIOBJ *table, BOOL isRight);
```

Description

Scrolls one page in the visible area of the table object.

<i>*table</i>	Specifies the table object.
<i>isRight</i>	Specifies the setting to scroll right or left for one page.
FALSE	Scroll left
TRUE	Scroll right

Returned Value

The returned value is the column number of the left cell of the table object if the function succeeds. Otherwise, the returned value is 0xFFFF.

tblMovePageRow

Synopsis

```
UINT tblMovePageRow(UIOBJ *table, BOOL isDown);
```

Description

Scrolls one page in the visible area of the table object.

<i>*table</i>	Specifies the table object.
<i>isDown</i>	Specifies the setting to scroll up or down for one page.
FALSE	Scroll up
TRUE	Scroll down

Returned Value

The returned value is the row number of the top cell of the table object if the function succeeds. Otherwise, the returned value is 0xFFFF.

tblMoveRow

Synopsis

```
UINT tblMoveRow(UIOBJ *table, BOOL isDown);
```

Description

Scrolls one row in the visible area of the table object.

<i>*table</i>	Specifies the table object.
<i>isDown</i>	Specifies the setting to scroll up or down for one row.
FALSE	Scroll up
TRUE	Scroll down

Returned Value

The returned value is the row number of the top cell of the table object if the function succeeds. Otherwise, the returned value is 0xFFFF.

tblNew

Synopsis

```
UIOBJ *tblNew(WORD id, int x, int y, int w, int h, int column, int row, WORD style);
```

Description

Creates a table object.

<i>id</i>	Specifies the ID number of this table object.
<i>x</i>	Specifies x-coordinate of left side of this table, in pixels.
<i>y</i>	Specifies y-coordinate of top side of this table, in pixels.
<i>w</i>	Specifies the width of this table object in pixels. The minimum value is 12.
<i>h</i>	Specifies the height of this table object in pixels. The minimum value is 12.
<i>column</i>	Specifies the number of columns in this table.
<i>row</i>	Specifies the number of rows in this table.
<i>style</i>	Specifies the style of the table object. The following are possible values of style: 0, UI_RESERVED, UI_FRAME UI_HORIZONTAL_LINE UI_VERTICAL_LINE, UI_FRAME UI_HORIZONTAL_LINE UI_VERTICAL_LINE UI_RESERVED, UI_3D and UI_RESERVED UI_3D.

Returned Value

The returned value is a pointer of UIOBJ.

tablNewEx

Synopsis

```
UIOBJ *tablNewEx(WORD id, int x, int y, int w, int h, int pageCol, int pageRow,  
                int colNum, int rowNum, char scrollbar, DWORD style);
```

Description

Creates a table object.

<i>id</i>	Specifies the ID number of this table object.
<i>x</i>	Specifies the x-coordinate of left side of the table object.
<i>y</i>	Specifies the y-coordinate of top side of the table object.
<i>w</i>	Specifies the width of the table object in pixels. The minimum value is 12.
<i>h</i>	Specifies the height of the table object in pixels. The minimum value is 12.
<i>pageCol</i>	Specifies the column number of the visible area in the table.
<i>pageRow</i>	Specifies the row number of the visible area in the table.
<i>colNum</i>	Specifies the total column number in the table.
<i>rowNum</i>	Specifies the total row number in the table.
<i>scrollbar</i>	Specifies the setting to show scrollbar or not. 1 Show scrollbar. 0 Hide scrollbar.
<i>style</i>	Specifies the style of the table object. The following are possible values of style: 0, UI_RESERVED, UI_FRAME UI_HORIZONTAL_LINE UI_VERTICAL_LINE, UI_FRAME UI_HORIZONTAL_LINE UI_VERTICAL_LINE UI_RESERVED, UI_3D and UI_RESERVED UI_3D.

Returned Value

The returned value is a pointer of UIOBJ.

tablPopUp

Synopsis

```
WORD tablPopUp(int x, int y, int w, int h, int pageCol, int pageRow, int totCol,  
              int totRow, int itemNum, T_NODE *item, int revertItem, DWORD style);
```

Description

Pops out a new table.

<i>x</i>	Specifies x-coordinate of left side of this table object, in pixels.
<i>y</i>	Specifies y-coordinate of top side of this table object, in pixels.
<i>w</i>	Specifies the width of this table object. The minimum value is 12.
<i>h</i>	Specifies the height of this table object. The minimum value is 12.
<i>pageCol</i>	Specifies how many columns shown in one page.
<i>pageRow</i>	Specifies how many rows shown in one page.
<i>totCol</i>	Specifies the total columns in the table.
<i>totRow</i>	Specifies the total rows in the table.
<i>itemNum</i>	Specifies the total items of the visible area. It should be TotCol x TotRow.
* <i>item</i>	Specifies a T_NODE structure.
<i>revertItem</i>	Reserve item.
<i>style</i>	Specifies the style of the table object. The possible values of table style are

UI_FRAME, UI_HORIZONTAL_LINE, UI_VERTICAL_LINE, UI_3D and UI_RESERVED.

Returned Value

The returned value is the ID number of T_NODE if the function succeeds. Otherwise, the returned value is 0xFFFF.

tblRemoveColumn

Synopsis

```
BOOL tblRemoveColumn(UIOBJ *table, long col);
```

Description

Removes a column from the table.

<i>*table</i>	Specifies the table object.
<i>col</i>	Specifies the column to be removed from the table.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

tblRemoveRow

Synopsis

```
BOOL tblRemoveRow(UIOBJ *table, long row);
```

Description

Removes a row from the table object.

<i>*table</i>	Specifies the table object.
<i>row</i>	Specifies the row to be removed from the table object.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

tblResetActiveZone

Synopsis

```
void tblResetActiveZone(UIOBJ *table);
```

Description

Resets the active zone of the table object. All cells will become enabled.

<i>*table</i>	Specifies the table object.
---------------	-----------------------------

Returned Value

None.

tblSetActive

Synopsis

```
BOOL tblSetActive(UIOBJ *table, int col, int row);
```

Description

Activates the grid of table object.

<i>*table</i>	Specifies the table object.
<i>col</i>	Specifies the column of the cell to be active.
<i>row</i>	Specifies the row of the cell object to be active.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

tblSetActiveZone

Synopsis

```
void tblSetActiveZone(UIOBJ *table,int col1, int row1, int col2, int row2);
```

Description

Sets Active Zone of the table object. Outside the active zone, all cells will gray out and disable.

<i>*table</i>	Specifies the table object.
<i>col1</i>	Specifies the beginning of the column in the table.
<i>row1</i>	Specifies the beginning of the row in the table.
<i>col2</i>	Specifies the ending of the column in the table.
<i>row2</i>	Specifies the ending of the row in the table.

Returned Value

None.

tblGridToCoord

Synopsis

```
BOOL tblGridToCoord(UIOBJ *table,int col,int row,long *x1,long *y1,long *x2,long *y2);
```

Description

Sets Active Zone of the table object. Outside the active zone, all cells will gray out and disable.

<i>*table</i>	Specifies the table object.
<i>col</i>	Specifies the the column in the table.
<i>row</i>	Specifies the the row in the table.
<i>x1</i>	Specifies x-coordinate of right side of the cell in this table, in pixels.
<i>y1</i>	Specifies y-coordinate of top side of the cell in this table, in pixels.
<i>x2</i>	Specifies x-coordinate of left side of the cell in this table, in pixels.
<i>y2</i>	Specifies y-coordinate of bottom side of the cell in this table, in pixels.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

tblSetActiveZoneEx

Synopsis

```
void tblSetActiveZoneEx (UIOBJ *table, int col1, int row1, int col2, int row2);
```

Description

Sets active zone. Outside the active zone, all cells will gray out and disabled.

<i>*table</i>	Specifies the table object.
<i>col1</i>	Specifies the beginning of the column in the table.
<i>row1</i>	Specifies the beginning of the row in the table.
<i>col2</i>	Specifies the ending of the column in the table.
<i>row2</i>	Specifies the ending of the row in the table.

Returned Value

None.

tblSetNoActive

Synopsis

```
BOOL tblSetNoActive (UIOBJ *table, int col, int row);
```

Description

Sets the cell of the table object not to active.

<i>*table</i>	Specifies the table object.
<i>col</i>	Specifies the column of the cell.
<i>row</i>	Specifies the row of the cell.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

tblSetReserve

Synopsis

```
void tblSetReserve(UIOBJ *table, int col, int row);
```

Description

Sets the reserved area in a table.

<i>*table</i>	Specifies the table object.
<i>col</i>	Specifies the column number of this table.
<i>row</i>	Specifies the row number of this table.

Returned Value

None.

tblSubNew

Synopsis

```
UIOBJ *tblSubNew(WORD id, int x, int y, int *wCol, int *hRow, int col, int row,  
                DWORD style);
```

Description

Creates a table.

<i>id</i>	Specifies the unique ID number of this button object.
<i>x</i>	Specifies x-coordinate of left side of this table object, in pixels.
<i>y</i>	Specifies y-coordinate of top side of this table object, in pixels.
* <i>wCol</i>	Specifies the width of each column.
* <i>hRow</i>	Specifies the height of each row.
<i>col</i>	Specifies the number of columns in the table.
<i>row</i>	Specifies the number of rows in the table.
<i>style</i>	Specifies the style of the table object. The possible values are UI_HORIZONTAL_LINE, UI_VERTICAL_LINE, UI_3D and UI_RESERVED.

Returned Value

The returned value is a pointer of UIOBJ.

tblTextOut

Synopsis

```
void tblTextOut(UIOBJ *table, int col, int row, char *data, BYTE fontId);
```

Description

Sets the content of the cell in a table.

* <i>table</i>	Specifies the table object.
<i>col</i>	Specifies the column of the table.
<i>row</i>	Specifies the row of the table.
* <i>data</i>	Specifies the content of the cell.
<i>fontId</i>	Specifies the ID number of font.

Returned Value

None.

tblTextOutEx

Synopsis

```
void tblTextOutEx(UIOBJ *table, int col, int row, char *data, BYTE fontId, int type);
```

Description

Sets the string content of the cell in a table.

* <i>table</i>	Specifies the table object.
<i>col</i>	Specifies the column of the table.
<i>row</i>	Specifies the row of the table.
* <i>data</i>	Specifies the string content of the cell.

<i>fontId</i>	Specifies the ID number of fonts.
<i>type</i>	Specifies the position of content.

Returned Value

None.

Tool Bar

General Description

The following are possible values of *style* when creating Tool Bar:

TBUI_STYLE_BUTTON
TBUI_STYLE_CHECKGROUP

The following are possible values of *state* when creating Tool Bar:

TBUI_STATE_NORMAL
TBUI_STATE_PRESSED
TBUI_STATE_DISABLE.

The following are possible values of *button border mode*:

TBUI_BUTTON_MODE_SQUARE
TBUI_BUTTON_MODE_ROUND

The following are possible values of *position*:

TBUI_TOOLTIP_BOTTOM
TBUI_TOOLTIP_TOP
TBUI_TOOLTIP_HIDE.

The following are possible values of *tip mode*:

TBUI_TIP_MODE_SQUARE
TBUI_TIP_MODE_ROUND

Related Data Structure

UIOBJ Structure

System Message

MT_TOOLBAR_COMMAND	Presses the toolbar object.
pMsg->id	ID of this toolbar.
pMsg-> msg.data.w	ID of toolbar' s button.
MT_TOOLBAR_DRAW_TIP	The tip of the button shown on screen.
MT_TOOLBAR_REMOVE_TIP	The tip of the button disappear.

tbrAddBtn

Synopsis

```
WORD tbrAddBtn(UIOBJ *tbar, WORD cmdId, WORD x, WORD y, WORD cx, WORD cy,  
               WORD state, const unsigned char *bmp, const char *tip);
```

Description

Adds a new button into toolbar.

<i>tbar</i>	Specifies the toolbar object.
<i>x</i>	Specifies the x-coordinate of left side of the toolbar object.
<i>y</i>	Specifies the y-coordinate of top side of the toolbar object.
<i>cx</i>	Specifies the width of this toolbar in pixels. This should be more than 3.
<i>cy</i>	Specifies the height of this toolbar in pixels. This should be more than 3.
<i>bmp</i>	Specifies the Object Bmp data pointer, the Bmp data should be within the scope of cx and cy. If the pointer refers to BUI_NO_BMP or 0, the Bmp file will not be load.
<i>cmdId</i>	Specifies the command ID of button
<i>state</i>	Specifies the state of the button.
<i>tip</i>	Specifies the content for the tip.

Returned Value

The returned value is a pointer of UIOBJ.

tbrAddButton

Synopsis

```
WORD tbrAddButton(UIOBJ *tbar, WORD x, WORD y, WORD cx, WORD cy, WORD state,  
                  const char *tip);
```

Description

Adds a new button into toolbar.

<i>tbar</i>	Specifies the toolbar object.
<i>x</i>	Specifies the x-coordinate of left side of the toolbar object.
<i>y</i>	Specifies the y-coordinate of top side of the toolbar object.
<i>cx</i>	Specifies the width of this toolbar in pixel., it should be more than 3.
<i>cy</i>	Specifies the height of this toolbar in pixel, it should be more than 3.
<i>cmdId</i>	Specifies the command ID of button
<i>state</i>	Specifies the state of the button.
<i>tip</i>	Specifies the content for the tip.

Returned Value

The returned value is a pointer of UIOBJ.

tbrGetBtnState

Synopsis

```
WORD tbrSetBtnState(UIOBJ *tbar, int idx, WORD state);
```

Description

Obtains the button state of the toolbar object.

<i>tbar</i>	Specifies the toolbar object.
<i>idx</i>	specifies the index number of the button.
<i>state</i>	Specifies the state of the button.

Returned Value

The returned value is the state of the button if this function succeeds. Otherwise, the returned value is 0xFFFF.

tbrGetButtonBorder

Synopsis

```
WORD tbrGetButtonBorder(UIOBJ *tbar);
```

Description

Obtains the border mode of the toolbar button.

<i>*tbar</i>	Specifies the table object.
--------------	-----------------------------

Returned Value

The returned value is the previous mode of the toolbar button if this function succeeds. Otherwise, the returned value is 0xFFFF.

tbrGetButtonState

Synopsis

```
WORD tbrGetButtonState(UIOBJ *tbar, int idx);
```

Description

Obtains the state of the button.

<i>*tbar</i>	Specifies the Toolbar object.
<i>idx</i>	Specifies the index number of the button.

Returned Value

The returned value is the state of the button if this function succeeds. Otherwise, the returned value is 0xFFFF.

tbrGetShowTipPos

Synopsis

```
WORD tbrGetShowTipPos(UIOBJ *tbar);
```

Description

Obtains the position of the tip.

<i>*tbar</i>	Specifies the Toolbar object.
--------------	-------------------------------

Returned Value

The returned value is the position of the tip if the function succeeds. Otherwise, the returned value is 0xFFFF.

tbrGetTipBorder

Synopsis

```
WORD tbrGetTipBorder(UIOBJ *tbar);
```

Description

Obtains the border of the tip.

**tbar* Specifies the Toolbar object.

Returned Value

The returned value is mode of the border if this function succeeds. Otherwise, the returned value is 0xFFFF.

tbrGetTipFont

Synopsis

```
WORD tbrGetTipFont(UIOBJ *tbar);
```

Description

Obtains the font of the tip.

**tbar* Specifies the Toolbar object.

Returned Value

The returned value is the current font if the function succeeds. Otherwise return 0xFFFF.

tbrGetTipHandle

Synopsis

```
VHANDLE tbrGetTipHandle(UIOBJ *tbar, int subId);
```

Description

Obtains the content of the tip.

**tbar* Specifies the Toolbar object.
subId Specifies the command id for this button.

Returned Value

The returned value is the content of the tip if the function succeeds. Otherwise, the returned value is 0.

tbrGetTipText

Synopsis

```
VHANDLE tbrGetTipText(UIOBJ *tbar, WORD subId);
```

Description

Obtains the content of the tip.

**tbar* Specifies the toolbar object.
subId Specifies the command id for this button.

Returned Value

The returned value is the content of the tip if the function succeeds. Otherwise, the returned value is 0.

tbrNew

Synopsis

```
UIOBJ *tbrNew(unsigned short id, const unsigned char *bmp, int x, int y, int style);
```

Description

Creates the Toolbar.

id Specifies the ID number of this toolbar object.
**bmp* Specifies the bit map of the toolbar.
x Specifies x-coordinate of left side of this toolbar, in pixels.
y Specifies y-coordinate of top side of this toolbar, in pixels.
style Specifies the style of this Toolbar.

Returned Value

The returned value is a pointer of UIOBJ.

tbrNewEx

Synopsis

```
UIOBJ *tbrNewEx(unsigned short id, int x, int y, int cx, int cy,  
const unsigned char *bmp, int btnStyle, DWORD style);
```

Description

Creates the Toolbar object.

id Specifies the ID number of this toolbar object.
x Specifies the x-coordinate of left side of the toolbar object.
y Specifies the y-coordinate of top side of the toolbar object.
cx Specifies the width of this toolbar in pixel., it should be more than 3.
cy Specifies the height of this toolbar in pixel, it should be more than 3.
bmp Specifies the Object Bmp data pointer, the Bmp data should be within the scope of cx and cy. If the pointer refers to BUI_NO_BMP or 0, the Bmp file will not be load.
btnStyle Specifies the style of the button. The possible values are TBUI_STYLE_BUTTON and TBUI_STYLE_CHECKGROUP. If this is set to TBUI_STYLE_CHECKGROUP, it disables UI_RESERVED.
style Specifies the style of this toolbar object. The possible values are UI_FRAME, UI_3D, UI_RESERVED, UI_BODER and UI_ITEM_FRAME. If this is set to UI_3D, it disables

UI_BODER and UI_ITEM_FRAME.

Returned Value

The returned value is a pointer of UIOBJ.

tbrRemoveBtn

Synopsis

```
void tbrRemoveBtn(UIOBJ *tbar, WORD cmdId, BOOL reDraw);
```

Description

Removes the button from the toolbar. Releases the memory space allocated for button and by tbrGetTipHandle() and tbrGetTipText() string pointer.

<i>tbar</i>	Specifies the toolbar object.
<i>cmdId</i>	Specifies the command id of the button.
<i>reDraw</i>	Specifies the setting to redraw toolbar object automatically.

Returned Value

None.

tbrSetBtnState

Synopsis

```
WORD tbrSetBtnState(UIOBJ *tbar, int btnId, WORD state);
```

Description

Sets the state of the button of the Toolbar object,

<i>tbar</i>	Specifies the Toolbar object.
<i>btnId</i>	Specifies the Command ID of button
<i>state</i>	Specifies the state of the toolbar.

Returned Value

The returned value is previous state if this function succeeds. Otherwise, the returned value is 0xFFFF.

tbrSetButtonBorder

Synopsis

```
WORD tbrSetButtonBorder(UIOBJ *tbar, WORD mode);
```

Description

Sets the border mode of the toolbar button.

<i>*tbar</i>	Specifies the Toolbar object.
<i>mode</i>	Specifies the border mode of the button.

Returned Value

The returned value is the previous mode of the toolbar button if this function succeeds. Otherwise, the returned value is 0xFFFF.

tbrSetButtonState

Synopsis

```
WORD tbrSetButtonState(UIOBJ *tbar, int idx, WORD state);
```

Description

Sets the state of the button.

<i>*tbar</i>	Specifies the Toolbar object.
<i>idx</i>	Specifies the index number of the button.
<i>state</i>	Specifies the state of the toolbar.

Returned Value

The returned value is previous state if this function succeeds. Otherwise, the returned value is 0xFFFF.

tbrSetShowTipPos

Synopsis

```
WORD tbrSetShowTipPos(UIOBJ *tbar, WORD position);
```

Description

Sets the position of the tip.

<i>*tbar</i>	Specifies the Toolbar object.
<i>position</i>	Specifies the position of the tip.

Returned Value

The returned value is the previous position if the function succeeds. Otherwise, the returned value is 0xFFFF.

tbrSetTipBorder

Synopsis

```
WORD tbrSetTipBorder(UIOBJ *tbar, WORD mode);
```

Description

Sets the border of the tip.

<i>*tbar</i>	Specifies the Toolbar object.
<i>mode</i>	Specifies the mode of tip border

Returned Value

The returned value is previous mode if this function succeeds. Otherwise, the returned value is 0xFFFF.

tbrSetTipFont

Synopsis

```
WORD tbrSetTipFont(UIOBJ *tbar, WORD fontId);
```

Description

Sets the font of the tip.

<i>*tbar</i>	Specifies the Toolbar object.
<i>fontId</i>	Specifies the ID number of fonts.

Returned Value

The returned value is the previous font if the function succeeds. Otherwise, the returned value is 0xFFFF.

tbrSetTipText

Synopsis

```
void tbrSetTipText(UIOBJ *tbar, WORD cmdId, const char *tip);
```

Description

Sets the button tip of the Toolbar object.

<i>tbar</i>	Specifies the toolbar object.
<i>cmdId</i>	Specifies the command id of the button.
<i>*tip</i>	Specifies the new content for the tip.

Returned Value

None.

tbrSetToolTip

Synopsis

```
void tbrSetToolTip(UIOBJ *tbar, int idx, const char *tip);
```

Description

Sets the content of the tip.

<i>*tbar</i>	Specifies the Toolbar object.
<i>idx</i>	Specifies the index number of the button.
<i>*tip</i>	Specifies the new content for the tip.

Returned Value

None.

tbrSubIDtoIndex

Synopsis

```
int tbrSubIDtoIndex(UIOBJ *tbar, int subId);
```

Description

Converts button's command id to index number.

<i>*tbar</i>	Specifies the toolbar object.
<i>subId</i>	Specifies the command id for this button.

Returned Value

The returned value is index number if this function succeeds. Otherwise, the returned value is -1.

tbrToolBarIsEnable

Synopsis

```
WORD tbrToolBarIsEnable(UIOBJ *tbar);
```

Description

Enables or disables the toolbar.

<i>*tbar</i>	Specifies the ID number of this Toolbar object.
--------------	---

Returned Value

The returned value is the TBUI_DISABLE_TOOLBAR or TBUI_ENABLE_TOOLBAR if the function succeeds. Otherwise, the returned value is 0xFFFF.

Whiteboard

General Description

Displays a whiteboard object on the screen.

Related Data Structure

UIOBJ Structure

System Message

MT_WHITEBOARD_COMMAND	Presses the whiteboard object.
pMsg->id	ID of this whiteboard.

wbClean

Synopsis

```
void wbClean(UIOBJ *wboard);
```

Description

Cleans the content of the whiteboard.

**wboard* Specifies the whiteboard object.

Returned Value

None.

wbCopyBmp

Synopsis

```
BOOL wbCopyBmp(UIOBJ *wboard);
```

Description

Copies the selected area in the whiteboard into the memory buffer

**wboard* Specifies the whiteboard object.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

wbNew

Synopsis

```
UIOBJ *wbNew(WORD id, int x, int y, int w, int h);
```

Description

Creates a whiteboard.

<i>id</i>	Specifies the unique ID number of this whiteboard object.
<i>x</i>	Specifies x-coordinate of left side of this whiteboard object, in pixels.
<i>y</i>	Specifies y-coordinate of top side of this whiteboard object, in pixels.
<i>w</i>	Specifies the width of this whiteboard object.
<i>h</i>	Specifies the height of this whiteboard object.

Returned Value

The returned value is a pointer of UIOBJ.

wbNewEx

Synopsis

```
UIOBJ *wbNewEx(WORD id, int x, int y, int w, int h, DWORD style);
```

Description

Enhances the function of `wbNew` for the 3D attribution setting.

<i>id</i>	Specifies the unique ID number of this whiteboard object.
<i>x</i>	Specifies x-coordinate of left side of this whiteboard object, in pixels.
<i>y</i>	Specifies y-coordinate of top side of this whiteboard object, in pixels.
<i>w</i>	Specifies the width of this whiteboard object.
<i>h</i>	Specifies the height of this whiteboard object.
<i>style</i>	Specifies the style of this whiteboard object. The only possible values is <code>UI_3D</code> .

Returned Value

The returned value is a pointer of `UIOBJ`.

wbPasteBmp

Synopsis

```
BOOL wbPasteBmp(UIOBJ *wboard, WORD x, WORD y);
```

Description

Pastes the selected area in the whiteboard from the memory buffer

<i>*wboard</i>	Specifies the whiteboard object.
<i>x</i>	Specifies x-coordinate of the selected area.
<i>y</i>	Specifies y-coordinate of the selected area.

Returned Value

The returned value is `TRUE` if this function succeeds. Otherwise, the returned value is `FALSE`.

wbSetDrawStyle

Synopsis

```
void wbSetDrawStyle(UIOBJ *wboard, int drawStyle);
```

Description

Sets the draw style of the whiteboard.

<i>*wboard</i>	Specifies the whiteboard object.
<i>drawStyle</i>	Specifies the setting of the draw mode. The following are possible values: <code>WB_DrawStyle_NORMAL</code> <code>WB_DrawStyle_LINE</code> <code>WB_DrawStyle_RECT</code> <code>WB_DrawStyle_CIRCULAR</code> <code>WB_DrawStyle_GETBMP</code>

Returned Value

None.

wbSetPenColor

Synopsis

```
void wbSetPenColor(UIOBJ *wboard, BYTE penColor);
```

Description

Sets the pen color of the whiteboard.

<i>*wboard</i>	Specifies the whiteboard object.
<i>penColor</i>	Specifies the setting of the pen color. The following are possible values:
PENCLR_LEVEL0	Specifies the white.
PENCLR_LEVEL1	Specifies the light gray.
PENCLR_LEVEL2	Specifies the dark gray.
PENCLR_LEVEL3	Specifies the black.

Returned Value

None.

wbSetPenColorEx

Synopsis

```
void wbSetPenColorEx(UIOBJ *wboard, BYTE penColor, WORD style);
```

Description

Sets the pen color of the whiteboard.

<i>wboard</i>	Specifies the whiteboard object.
<i>penColor</i>	Specifies the setting of the pen color. The following are possible values:
PENCLR_LEVEL0	Specifies the white.
PENCLR_LEVEL1	Specifies the light gray.
PENCLR_LEVEL2	Specifies the dark gray.
PENCLR_LEVEL3	Specifies the black.
<i>style</i>	Specifies the style of filling. The possible values are FILLBMP_NO and FILLBMP_YES.

Returned Value

None.

wbSetPenMode

Synopsis

```
void wbSetPenMode(UIOBJ *wboard, BYTE penMode);
```

Description

Sets the pen mode of the whiteboard.

<i>*wboard</i>	Specifies the whiteboard object.
<i>penMode</i>	Specifies the setting of the pen mode. The following are possible values:
WB_PENMODE_REPLACE	Specifies the replace mode.
WB_PENMODE_XOR	Specifies the xor mode.
WB_PENMODE_TRANSPARENT	Specifies the transparent mode.

Returned Value

None.

wbSetPenStyle

Synopsis

```
void wbSetPenStyle(UIOBJ *wboard, BYTE penStyle);
```

Description

Sets the pen style of the whiteboard.

<i>*wboard</i>	Specifies the whiteboard object.
<i>penStyle</i>	Specifies the setting of the pen mode. The following are possible values:
WB_PEN_PATTERN_SOLID	Specifies the solid line.
WB_PEN_PATTERN_DOT	Specifies the dot line.
WB_PEN_PATTERN_DASH	Specifies the dash line.

Returned Value

None.

wbSetPenWidth

Synopsis

```
void wbSetPenWidth(UIOBJ *wboard, BYTE penWidth);
```

Description

Sets the pen width of the whiteboard.

<i>*wboard</i>	Specifies the whiteboard object.
<i>penWidth</i>	Specifies the setting of the pen width. The following are possible values:
WB_PW_LEVEL1	The pen width is level 1. This is the default value and the narrowest pen width.
WB_PW_LEVEL2	The pen width is level 2.
WB_PW_LEVEL3	The pen width is level 3.
WB_PW_LEVEL4	The pen width is level 4.
WB_PW_LEVEL5	The pen width is level 5.
WB_PW_LEVEL6	The pen width is level 6. This is the widest pen width.

Returned Value

None.

Progress Bar

General Description

The following are possible values of *style*:

CPB_NORMAL	General style
CPB_VERTICAL	Vertical style
CPB_HORIZONTAL	Horizontal style
CPB_GRID	Grid style

Related Data Structure

UIOBJ Structure

System Message

MT_PROGRESS_PROCESS_END	The time interval of the progress bar is passed, and the value of the progress bar reach 100 persentage.
pMsg->data.w	The value (in persentang) of the progress bar.
MT_PROGRESS_TIMER_INC	The time interval of the progress bar is passed.
pMsg->data.w	The value (in persentang) of the progress bar.

progsChangeMaxRange

Synopsis

```
void progsChangeMaxRange(UIOBJ *pbar, int max);
```

Description

Sets the maximum value of the progress bar. Before it is changed, you have to execute ProgsReset(), then you can redefine the max range of the Progress Bar.

<i>*pbar</i>	Specifies the progress bar object.
<i>max</i>	Specifies the maximum value of the progress bar.

Returned Value

None.

progsEnableTimer

Synopsis

```
void progsEnableTimer(UIOBJ *pbar, BOOL isEnabled);
```

Description

Enables or disables the timer function of the progress bar or not.

<i>*pbar</i>	Specifies the progress bar object.
<i>isEnabled</i>	Specifies the setting for timer function.
TRUE	Enable the timer function.
FALSE	Disable the timer function.

Returned Value

None.

progsGetIncVal

Synopsis

```
DWORD progsGetIncVal(UIOBJ *pbar);
```

Description

Obtains the value by which the value of the progress bar to be increased.

**pbar* Specifies the progress bar object.

Returned Value

The returned value is the value of the progress bar to be increased. if this function succeeds. Otherwise, the returned value is 0.

progsGetPercent

Synopsis

```
int progsGetPercent (UIOBJ *pbar) ;
```

Description

Obtains current value of the progress bar in percentage.

**pbar* Specifies the progress bar object.

Returned Value

The returned value is the percentage if this function succeeds. Otherwise, the returned value is -1.

progsGetTimer

Synopsis

```
DWORD progsGetTimer (UIOBJ *pbar) ;
```

Description

Obtains the time interval to to send out message.

**pbar* Specifies the progress bar object.

Returned Value

The returned value is the value of time function of the progress bar if this function succeeds. Otherwise, the returned value is 0.

progsIncrease

Synopsis

```
BOOL progsIncrease (UIOBJ *pbar, int value) ;
```

Description

Adds the range value of the Progress.

**pbar* Specifies the progress bar object.
value Specifies the value to add.

Returned Value

The returned value is TRUE if the percent of the progress bar reaches 100. Otherwise, the returned value is FALSE.

progsNew

Synopsis

```
UIOBJ *progsNew (WORD id, int x, int y, int max) ;
```

Description

Creates a new progress bar object.

id Specifies the unique ID number of this progress bar object.
x Specifies x-coordinate of left side of this progress bar object, in pixels.

<i>y</i>	Specifies y-coordinate of top side of this progress bar object, in pixels.
<i>max</i>	Specifies the maximum value for this progress bar.

Returned Value

The returned value is a pointer of UIOBJ.

progsNewEx

Synopsis

```
UIOBJ *progsNewEx(WORD id, int x, int y, int max, int len, int wh, WORD style,
                  DWORD state);
```

Description

Creates a new Progress.

<i>id</i>	Specifies the ID number of this progress bar object.
<i>x</i>	Specifies x-coordinate of left side of this progress bar object, in pixels.
<i>y</i>	Specifies y-coordinate of top side of this progress bar object, in pixels.
<i>max</i>	Specifies the maximum value of this progress bar object.
<i>len</i>	Specifies the length of progress bar.
<i>wh</i>	Specifies the width of progress bar.
<i>style</i>	Specifies the style of the progress bar object. The following are possible values:
	CPB_NORMAL General style
	CPB_VERTICAL Vertical style
	CPB_HORIZONTAL Horizontal style
	CPB_GRID Grid style
<i>state</i>	Specifies the state of the progress bar object. The following are possible values:
	0 General view
	UI_3D 3D view

Returned Value

The returned value is a pointer of UIOBJ.

progsReset

Synopsis

```
void progsReset(UIOBJ *pbar);
```

Description

Sets the value of the progress bar to 0.

<i>*pbar</i>	Specifies the progress bar object.
--------------	------------------------------------

Returned Value

None.

progsSetIncVal

Synopsis

```
void progsSetIncVal(UIOBJ *pbar, DWORD interval);
```

Description

Sets the value by which the value of the progress bar to be increased.

<i>*pbar</i>	Specifies the progress bar object.
<i>interval</i>	Specifies the value to be added.

Returned Value

None.

progsSetPercent

Synopsis

```
void progsSetPercent(UIOBJ *pbar, int percent);
```

Description

Sets the percent value of the progress bar.

<i>*pbar</i>	Specifies the progress bar object.
<i>percent</i>	Specifies the percentage value to be set.

Returned Value

None.

progsSetTimer

Synopsis

```
void progsSetTimer(UIOBJ *pbar, DWORD interval);
```

Description

Sets the time interval to send out message in ms.

<i>*pbar</i>	Specifies the progress bar object.
<i>interval</i>	Specifies the time interval to send out message in ms. The minimum value is 10 ms.

Returned Value

None.

Menu Bar

General Description

Displays a menu bar on screen.

Related Data Structure

UIOBJ Structure

System Message

MT_MENUBAR_COMMAND	Selects an item in the menu bar.
pMsg->id	The ID of this menu bar.
pMsg->data	The ID of the the item.

menuAddItem

Synopsis

```
WORD menuAddMenuItem(UIOBJ *mbar, WORD secId, WORD cmdId, const char *item, char hotKey);
```

Description

Adds a new item depend on section for this menubar.

<i>*mbar</i>	Specifies the Menubar object.
<i>secId</i>	Specifies the section ID.
<i>cmdId</i>	Specifies the notification command id for this item.
<i>*item</i>	Specifies the content of the new Item.
<i>hotKey</i>	Specifies the key value of the hot key. This value can be set to MBUI_NO_HOTKEY if no hot key is used.

Returned Value

The returned value is 1 if this function succeeds. Otherwise, the returned value is 0.

menuAddItemArray

Synopsis

```
WORD menuAddItemArray(UIOBJ *mbar, WORD secId, L_NODE *item, int itemNum);
```

Description

Adds a new item array depend on section for this menubar.

<i>*mbar</i>	Specifies the Menubar object.
<i>secId</i>	Specifies the section ID.
<i>item</i>	Specifies the item array in L_NODE structure.
<i>itemNum</i>	Specifies the number of items.

Returned Value

The returned value is 1 if this function succeeds. Otherwise, the returned value is 0.

menuAddMenuHelp

Synopsis

```
WORD menuAddMenuHelp(UIOBJ *mbar, const char *sect);
```

Description

Adds the help section align right for this Menubar.

<i>*mbar</i>	Specifies the Menubar object.
<i>*sect</i>	Specifies the content of the new section.

Returned Value

The returned value is 1 if this function succeeds. Otherwise, the returned value is 0.

menuAddMenuItem

Synopsis

```
WORD menuAddMenuItem(UIOBJ *mbar, WORD idx, WORD cmdId, const char *item, char hotKey);
```

Description

Adds a new item depend on section for this menubar.

<i>*mbar</i>	Specifies the Menubar object.
<i>idx</i>	Specifies the index of the section.
<i>cmdId</i>	Specifies the notification command id for this item.
<i>*item</i>	Specifies the content of the new Item.
<i>hotKey</i>	Specifies the key value of the hot key. This value can be set to MBUI_NO_HOTKEY if no hot key is used.

Returned Value

The returned value is 1 if this function succeeds. Otherwise, the returned value is 0.

menuAddMenuSection

Synopsis

```
WORD menuAddMenuSection(UIOBJ *mbar, const char *sect);
```

Description

Adds a new section align right for this Menubar.

<i>*mbar</i>	Specifies the Menubar object.
<i>*sect</i>	Specifies the content of the new section.

Returned Value

The returned value is 1 if this function succeeds. Otherwise, the returned value is 0.

menuAddSection

Synopsis

```
WORD menuAddSection(UIOBJ *mbar, WORD secId, const char *item);
```

Description

Adds a new section for this menubar.

<i>*mbar</i>	Specifies the Menubar object.
<i>secId</i>	Specifies the sectionID.
<i>*item</i>	Specifies the content of the new item.

Returned Value

The returned value is 1 if this function succeeds. Otherwise, the returned value is 0.

menuAddSectionHelp

Synopsis

```
WORD menuAddSectionHelp(UIOBJ *mbar, WORD secId, const char *string);
```

Description

Adds the help section for this menubar.

<i>*mbar</i>	Specifies the Menubar object.
<i>secId</i>	Specifies the section ID.
<i>*string</i>	Specifies the content of this help section.

Returned Value

The returned value is 1 if this function succeeds. Otherwise, the returned value is 0.

menuGetItem

Synopsis

```
VHANDLE menuGetItem(UIOBJ *mbar, WORD secId, WORD item);
```

Description

Obtains item text of the menu section.

<i>*mbar</i>	Specifies the Menubar object.
<i>secId</i>	Specifies the section ID.
<i>item</i>	Specifies the item ID

Returned Value

The returned value is item string handle if the function succeeds. Otherwise, the returned value is 0.

menuGetItemArray

Synopsis

```
int menuGetItemArray(UIOBJ *mbar, WORD secId, L_NODE *item, int itemNum);
```

Description

Obtains the item array ID and text of the menu section.

<i>*mbar</i>	Specifies the Menubar object.
<i>secId</i>	Specifies the section ID.
<i>item</i>	Specifies the item array in L_NODE structures.
<i>itemNum</i>	Specifies the number of items in this array.

Returned Value

The returned value is number of items if the function succeeds. Otherwise, the returned value is 0.

menuGetItemIndex

Synopsis

```
int menuGetItemIndex(UIOBJ *mbar, WORD sect, const char *string);
```

Description

Obtains the item index with the content matching the text string.

<i>*mbar</i>	Specifies the Menubar object.
<i>*sect</i>	Specifies the index of the section to be search.
<i>*string</i>	Specifies the text string to be found.

Returned Value

The returned value is the item index if this function succeeds. Otherwise, the returned value is -1.

menuGetItemState

Synopsis

```
WORD menuGetItemState(UIOBJ *mbar, WORD sectId, WORD item);
```

Description

Obtains the item array status of the menu section.

<i>*mbar</i>	Specifies the Menubar object.
<i>sectId</i>	Specifies the section ID.
<i>item</i>	Specifies the item ID

Returned Value

The returned value is the current state if this function succeeds. Otherwise, the returned value is 0xFFFF.

menuGetSectionIndex

Synopsis

```
int menuGetSectionIndex(UIOBJ *mbar, const char *string);
```

Description

Obtains the section index with the content matching the text string.

<i>*mbar</i>	Specifies the Menubar object.
<i>*string</i>	Specifies the text string to be found.

Returned Value

The returned value is the section index if this function succeeds. Otherwise, the returned value is -1.

menuGrayItem

Synopsis

```
void menuGrayItem(UIOBJ *mbar, WORD sect, WORD item, char isGray);
```

Description

Sets the item to be disabled (gray) or not.

<i>*mbar</i>	Specifies the Menubar object.
<i>sect</i>	Specifies the index of the section to be set.
<i>item</i>	Specifies the index of the item to be set.
<i>isGray</i>	Specifies the flag of setting enabling or not.
1	Set the item to be disabled.
0	Set the item to be enabled.

Returned Value

None.

menuNew

Synopsis

```
UIOBJ *menuNew(unsigned short id, short y, WORD style, WORD fontId);
```

Description

Creates the Menubar control.

<i>id</i>	Specifies the object ID for this Menubar.
<i>y</i>	Specifies y-coordinate of top side of this menu bar object, in pixels.
<i>style</i>	Specifies the style of the menu bar object. The possible values are MBUI_MENUBAR_STYLE_SQUARE and MBUI_MENUBAR_STYLE_ROUND.
<i>fontId</i>	Specifies the font used by label. The available fonts so far are GM_FONT_SMALL, GM_FONT_SMALLBOLD, GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD, and GM_FONT_LAGE. Only GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD, and GM_FONT_LARGE can be applied to Chinese characters.

Returned Value

The returned value is a pointer of UIOBJ if this function succeeds. Otherwise, the returned value is NULL.

menuNewEx

Synopsis

```
UIOBJ *menuNewEx(unsigned short id, short y, WORD mStyle, WORD fontId, DWORD style);
```

Description

Create new Menu object and allocate memory.

<i>id</i>	Specifies the object ID for this Menubar.
<i>y</i>	Specifies y-coordinate of top side of this menu bar object, in pixels.
<i>mStyle</i>	Specifies the menu style. Some of the styles can be combined with the OR operation. The following are possible values: (<i>a number</i>)
MBUI_MENU_IN_BOTTOM	Shadow pixels with the range from 0 to 15 must be used with this style. Menubar on screen bottom. This setting makes the parameter of y-coordinate ineffective.
MBUI_MENU_POP_UP	Pop up a submenu. This style must be used with MBUI_MENU_IN_BOTTOM.
Here are two valid examples: 10 MBUI_MENU_IN_BOTTOM MBUI_MENU_POP_UP 0 MBUI_MENU_POP_UP	

<i>fontId</i>	Specifies the font used by label. The available fonts so far are GM_FONT_SMALL, GM_FONT_SMALLBOLD, GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD, and GM_FONT_LARGE. Only GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD, and GM_FONT_LARGE can be applied to Chinese characters.	
<i>style</i>	Specifies the style of the menu bar object. The following are possible values:	
	UI_FRAME	Set the menu bar and submenu with frame.
	UI_3D	Set the menu bar and submenu with 3D effect. This setting makes both UI_FRAME and UI_ITEM_FRAME ineffective.
	UI_RESERVED	Highlight the selected command.
	UI_BODER	Set the menu bar with border.
	UI_ITEM_FRAME	Highlight the selected item.

Returned Value

The returned value is a pointer of UIOBJ if this function succeeds. Otherwise, the returned value is NULL.

menuPopup

Synopsis

```
void menuPopup(WORD menuId);
```

Description

Sets Menu to be active and displays it on screen.

menuId Specifies the menu bar ID. This function displays menu without needing to call the function of menuSetActiveMenu() and menuPopupMenuBar().

Returned Value

None.

menuPopupMenuBar

Synopsis

```
void menuPopupMenuBar(void);
```

Description

Shows the active menubar.

Returned Value

None

menuRemoveItem

Synopsis

```
void menuRemoveItem UIOBJ *mbar, WORD secId, WORD item);
```

Description

Removes Item from the menu section.

<i>*mbar</i>	Specifies the Menubar object.
<i>secId</i>	Specifies the section ID.
<i>item</i>	Specifies the item ID

Returned Value

None.

menuRemoveItemAll

Synopsis

```
void menuRemoveItemAll(UIOBJ *mbar, WORD secId);
```

Description

Removes all Items from the Menu Section exclude the Section.

<i>*mbar</i>	Specifies the Menubar object.
<i>secId</i>	Specifies the section ID.

Returned Value

None.

menuRemoveSection

Synopsis

```
void menuRemoveSection(UIOBJ *mbar, WORD secId);
```

Description

Removes all Items from the Menu Section include the Section.

<i>*mbar</i>	Specifies the Menubar object.
<i>secId</i>	Specifies the section ID.

Returned Value

None.

menuSectionFocus

Synopsis

```
void menuSectionFocus(UIOBJ *mbar, WORD secId);
```

Description

Highlights the selected section to the menu object and expands all items.

<i>*mbar</i>	Specifies the Menubar object.
<i>secId</i>	Specifies the section ID.

Returned Value

None.

menuSendHotKey

Synopsis

```
void menuSendHotKey(char hotKey);
```

Description

Sends the “MT_HOT_KEY” message.

hotKey Specifies the key value of the hot key.

Returned Value

None

menuSetActiveMenu

Synopsis

```
void menuSetActiveMenu(WORD id);
```

Description

Sets the active menubar.

id Specifies the command ID for this Menubar.

Returned Value

None.

menuSetItemState

Synopsis

```
void menuSetItemState(UIOBJ *mbar, WORD secId, WORD item, WORD state);
```

Description

Sets the Item status to the menu section.

<i>*mbar</i>	Specifies the Menubar object.
<i>secId</i>	Specifies the section ID.
<i>item</i>	Specifies the item ID
<i>state</i>	Specifies the state of the menu bar object. The following are possible values: MBUI_MENUBAR_ITEM_NORMAL Normal status MBUI_MENUBAR_ITEM_PRESSED Item is selected MBUI_MENUBAR_ITEM_DISABLE Item is disabled

Returned Value

None.

menuSetSectionFocus

Synopsis


```
void menuSetSectionFocus (UIOBJ *mbar, WORD sect);
```

Description

Sets focus section.

<i>*mbar</i>	Specifies the Menubar object.
<i>idx</i>	Specifies the index of the section.

Returned Value

None.

Message Box

General Description

Pops out a message box on the screen.

Related Data Structure

UIOBJ Structure

System Message

No system message for message box retrieving.

FloatBox

Synopsis

```
void FloatBox(char *message, int dTime, FONTID fontId);
```

Description

Pops up a window and displays a message on that window for a period. That window disappears after that period.

<i>*message</i>	Specifies the message to be displayed.
<i>dTime</i>	Specifies the time interval to display the message. The following are possible values: DELAY300ms 300 ms DELAY500ms 500 ms DELAY1000ms 1000 ms DELAY2000ms 2000 ms
<i>fontId</i>	Specifies the font used by combo box. The available fonts so far are GM_FONT_SMALL, GM_FONT_SMALLBOLD, GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD, and GM_FONT_LAGE. Only GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD, and GM_FONT_LARGE can be applied to Chinese characters.

Returned Value

None.

MsgBox

Synopsis

```
int MsgBox(int y, char *title, char *message, int button, int bmp, BYTE fontId);
```

Description

Pops up a window and displays a message on that window. That window disappears only after one of the button(s) is clicked.

<i>y</i>	Specifies the y-coordinate of the top side of the window.
<i>*title</i>	Specifies the title of the message window.
<i>*message</i>	Specifies the message to be displayed.
<i>button</i>	Specifies the number and the text of the buttons to be displayed on the message window. The following are possible values: OK_Only Only one button: "OK" OkCancel Two buttons: "OK" and "Cancel" AbortRetryIgnore Three buttons: "Abort", "Retry" and "Ignore" YesNoCancel Three buttons: "Yes", "No" and "Cancel" YesNo Two buttons: "Yes" and "No"
<i>bmp</i>	Specifies the bitmap image array.
<i>fontId</i>	Specifies the font used by combo box. The available fonts so far are GM_FONT_SMALL, GM_FONT_SMALLBOLD, GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD, and GM_FONT_LAGE. Only GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD, and GM_FONT_LARGE can be applied to Chinese characters.

Returned Value

The returned value is the button being clicked. The following are possible values:
MSGBOX_ERROR This function fails.

MSGBOX_CMD_OK	"OK" button was clicked.
MSGBOX_CMD_CANCEL	"Cancel" button was clicked.
MSGBOX_CMD_ABORT	"Abort" button was clicked.
MSGBOX_CMD_RETRY	"Retry" button was clicked.
MSGBOX_CMD_IGNORE	"Ignore" button was clicked.
MSGBOX_CMD_YES	"Yes" button was clicked.
MSGBOX_CMD_NO	"No" button was clicked.

Keyboard

General Description

Provides interfaces to defined a customized software keyboard.

Related Data Structure

KeyData Structure

This structure stores all information of a keyboard.

```
typedef struct tagKeyData {  
    short Left,Top,Right,Bottom;           // The coordinate for a key in the keyboard.  
    WORD KeyStyle;                         // The style setting for the keyboard. The following  
                                           // are possible value:  
                                           // KB_NORMAL        Normal style  
                                           // KB_CAPS          CAPS key is pressed.  
                                           // KB_SHIFT         SHIFTS key is pressed.  
                                           // KB_STRING  
                                           // KB_DBL_STRING  
    BYTE *Key[2];                         // The ASCII code for the key.  
} KeyData;
```

InputMethod Structure

System Message

MT_KEYBOARD_COMMAND pMsg->id	Select a key of the keyboard. The ID of this keyboard.
MT_KEYBOARD_FUNC pMsg->id pMsg->data.d	Selects a function key of the keyboard The ID of this keyboard. The ID of the function key.
MT_KEYBOARD_FOCUS_IN pMsg->id	Pen position is entering the keyboard. The ID of this keyboard.
MT_KEYBOARD_FOCUS_OUT pMsg->id	Pen position is leaving the keyboard. The ID of this keyboard.

kbdNew

Synopsis

```
UIOBJ *kbdNew(unsigned short id, int x, int y, int initDelay, int delay,  
              unsigned char **bmpList, int keyNum, KeyData *keyList);
```

Description:

Creates a software keyboard

<i>id</i>	Specifies the object identifier.
<i>x</i>	Specifies x-coordinate of left side of this object, in pixels.
<i>y</i>	Specifies y-coordinate of top side of this object, in pixels.
<i>initDelay</i>	Specifies the initial delay time.
<i>delay</i>	Specifies the delay time, but not including the initialization time.
** <i>bmpList</i>	Specifies the picture array of keyboard in bitmaps.
<i>keyNum</i>	Specifies the total number of keys.
* <i>keyList</i>	Specifies the KeyData structure of the key list.

Returned Value

The returned value is the object pointer if this function succeeds. Otherwise, the returned value is NULL.

kbdNewEx

Synopsis

```
UIOBJ *kbdNewEx(unsigned short id, int x, int y, unsigned char **bmpList, int keyNum,  
                KeyData *keyList, int x1, int y1, int w, int h, unsigned char *symbol,  
                InputMethod method);
```

Description

Creates a software keyboard with external input locales.

<i>id</i>	Specifies the object identifier.
<i>x</i>	Specifies x-coordinate of left side of this object, in pixels.
<i>y</i>	Specifies y-coordinate of top side of this object, in pixels.
** <i>bmpList</i>	Specifies the picture array of keyboard in bitmaps.
<i>keyNum</i>	Specifies the total number of keys.
* <i>keyList</i>	Specifies the KeyData structure of the key list.
<i>x1</i>	Specifies x-coordinate of the frame of the input locale.
<i>y1</i>	Specifies y-coordinate of the frame of the input locale.
<i>w</i>	Specifies the width of the frame of the input locale.
<i>h</i>	Specifies the height of the frame of the input locale.
* <i>symbol</i>	Specifies this argument is set by user. If the value is 0, the keyboard is the standard English keyboard. Otherwise, it should be set to a non-zero value.
<i>method</i>	Specifies the program of input locale.

Returned Value

The returned value is the object pointer if this function succeeds. Otherwise, the returned value is NULL.

kbdUnDefWordZone

Synopsis

```
void kbdUnDefWordZone(UIOBJ *kboard, int x, int y, int w, int h, UINT col, DWORD state);
```

Description

Sets the zone of candidating words.

<i>*kboard</i>	Specifies the Keyboard structure.
<i>x</i>	Specifies x-coordinate of left side of the zone of candidating words, in pixels.
<i>y</i>	Specifies y-coordinate of top side of the zone of candidating words, in pixels.
<i>w</i>	Specifies the width of the zone of candidating words.
<i>h</i>	Specifies the height of the zone of candidating words.
<i>col</i>	Specifies the number of columns of the zone of candidating words.
<i>state</i>	Specifies the UIState of the zone of candidating words.

Returned Value

None.

SysDefKB

Synopsis

```
UIOBJ *SysDefKB(WORD kbID);
```

Description

Obtains the default keyboard object of system.

kbId Specifies the ID number of Keyboard.

Returned Value

None.

SysEngKB

Synopsis

```
UIOBJ *SysEngKB(WORD kbId);
```

Description

Obtains the English keyboard object of system.

kbId Specifies the ID number of Keyboard.

Returned Value

None.

Scroll Bar

General Description

Creates a scroll bar object.

Related Data Structure

UIOBJ Structure

System Message

MT_SCROBAR_COMMAND pMsg->id pMsg->data.w	Presses the scroll bar object. ID of this scroll bar. The tank' s current position of the scroll bar.
MT_SCROBAR_POS_MOVE pMsg->id pMsg->data.val.s1 pMsg->data.val.s2	Moves the tank' s position of the scroll bar ID of this scroll bar. Diffence from last tank' s position to tank' s current position. The tank' s current position of the scroll bar.
MT_SCROBAR_ON_TOP pMsg->id	The tank reaches the top. ID of this scroll bar.
MT_SCROBAR_ON_BOTTOM pMsg->id	The tank reaches the bottom. ID of this scroll bar.
MT_SCROBAR_ENTER_UP pMsg->id pMsg->data.w	Pens down on the up arrow of scroll bar. ID of this scroll bar. The tank' s current position of the scroll bar.
MT_SCROBAR_ENTER_DN pMsg->id pMsg->data.w	Pens down on the down arrow of scroll bar. ID of this scroll bar. The tank' s current position of the scroll bar.
MT_SCROBAR_ENTER_PGUP pMsg->id pMsg->data.w	Pens down on the page up area of scroll bar. ID of this scroll bar. The tank' s current position of the scroll bar.
MT_SCROBAR_ENTER_PGDN pMsg->id pMsg->data.w	Pens down on the page down area of scroll bar. ID of this scroll bar. The tank' s current position of the scroll bar.
MT_SCROBAR_ENTER_TANK pMsg->id pMsg->data.w	Pens down on the tank of scroll bar. ID of this scroll bar. The tank' s current position of the scroll bar.

sbarAddTotalSize

Synopsis

```
void sbarAddTotalSize(UIOBJ *sbar, UINT addValue);
```

Description

Adds the value to the maximum value of of the scroll bar object.

**sbar* Specifies the scroll bar object.
addValue Specifies the value to be added to the maximum value of the scroll bar object.

Returned Value

None.

sbarGetMaxPos

Synopsis

```
UINT sbarGetMaxPos(UIOBJ *sbar);
```

Description

Obtains the maximum value of this scroll bar.

**sbar* Specifies the scroll bar object.

Returned Value

The returned value is the maximum value of this scroll bar if this function succeeds. Otherwise, the returned value is 0xFFFF.

sbarGetPageSize

Synopsis

```
UINT sbarGetPageSize(UIOBJ *sbar);
```

Description

Obtains the page size of the scroll bar object.

**sbar* Specifies the scroll bar object.

Returned Value

The returned value is the page size of the scroll bar object if this function succeeds. Otherwise, the returned value is 0xFFFF.

sbarGetPos

Synopsis

```
UINT sbarGetPos(UIOBJ *sbar);
```

Description

Obtains the current value of the scroll bar object.

**sbar* Specifies the scroll bar object.

Returned Value

The returned value is the current value of the scroll bar object if this function succeeds. Otherwise, the returned value is 0xFFFF.

sbarGetTotalSize

Synopsis

```
UINT sbarGetTotalSize(UIOBJ *sbar);
```

Description

Obtains the total size of the scroll bar.

**sbar* Specifies the scroll bar object.

Returned Value

The returned value is total size of the scroll bar if this function succeeds. Otherwise, the returned value is 0xFFFF.

sbarMove

Synopsis

```
BOOL sbarMove(UIOBJ *sbar, int x, int y, int len, int w);
```

Description

Resets some settings of a scroll bar.

<i>*sbar</i>	Specifies the scroll bar object.
<i>x</i>	Specifies the new x-coordinate of the scroll bar object.
<i>y</i>	Specifies the new y-coordinate of the scroll bar object.
<i>len</i>	Specifies the length of the scroll bar object.
<i>w</i>	Specifies the width of the scroll bar object.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

sbarMoveLine

Synopsis

```
UINT sbarMoveLine(UIOBJ *sbar, BOOL isDown);
```

Description

Scrolls up or down one line.

<i>*sbar</i>	Specifies the scroll bar object.
<i>isDown</i>	Specifies the setting for scrolling the scroll bar. The following are possible values:

TRUE	Scrolls down one line.
FALSE	Scrolls up one line.

Returned Value

The returned value is the current value of the scroll bar object if this function succeeds. Otherwise, the returned value is 0xFFFF.

sbarMovePage

Synopsis

```
UINT sbarMovePage(UIOBJ *sbar, BOOL isDown);
```

Description

Scrolls up or down one page.

<i>*sbar</i>	Specifies the scroll bar object.
<i>isDown</i>	Specifies the setting for scrolling the scroll bar.
TRUE	Scrolls down one page.
FALSE	Scrolls up one page.

Returned Value

The returned value is the current value of the scroll bar object if this function succeeds. Otherwise, the returned value is 0xFFFF.

sbarNew

Synopsis

```
UIOBJ *sbarNew(WORD id, int x, int y, int width, int len, UINT totalSize,
               UINT pageSize, WORD style);
```

Description

Creates a new scroll bar.

<i>id</i>	Specifies the ID number of this scroll bar object.
<i>x</i>	Specifies the x-coordinate of scroll bar object.
<i>y</i>	Specifies the y-coordinate of scroll bar object.
<i>width</i>	Specifies the width of the scroll bar object.
<i>len</i>	Specifies the length of the scroll bar object.
<i>totalSize</i>	Specifies the total size of the scroll bar object.
<i>pageSize</i>	Specifies the page size of the scroll bar object.
<i>style</i>	Specifies the style of the scroll bar object. The following are possible values: SBS_VERTICAL SBS_HORIZONTAL SBS_FULL_PAGE_CHANGE SBS_LESS_PAGE_CHANGE SBS_SLIDER

Returned Value

The returned value is a pointer of UIOBJ.

sbarNewEx

Synopsis

```
UIOBJ *sbarNewEx(WORD id, int x, int y, int width, int length, UINT maxValue,  
                UINT pageSize, WORD style, DWORD state);
```

Description

Creates a new scroll bar.

<i>id</i>	Specifies the ID number of this scroll bar object.
<i>x</i>	Specifies the x-coordinate of scroll bar object.
<i>y</i>	Specifies the y-coordinate of scroll bar object.
<i>width</i>	Specifies the width of the scroll bar object.
<i>length</i>	Specifies the length of the scroll bar object.
<i>maxValue</i>	Specifies the setting of the maximum value of the scroll bar object.
<i>pageSize</i>	Specifies the page size of the scroll bar object.
<i>style</i>	Specifies the scroll bar style. The following are possible values: SBS_VERTICAL Vertical Scrollbar SBS_HORIZONTAL Horizontal Scrollbar SBS_FULL_PAGE_CHANGE Do not show the last item of last page while page down. SBS_LESS_PAGE_CHANGE Show the last item of last page while page down. SBS_SLIDER SLIDER Scrollbar SBS_TANK_NOPRESENT Scrollbar without Tank SBS_ARROW_AUTOHIDE Hide the arrow (for the first or the last page)
<i>state</i>	Specifies the display settings. The following are possible values: 0 2D view UI_3D 3D view

Returned Value

The returned value is a pointer of UIOBJ.

sbarSetPageSize

Synopsis

```
void sbarSetPageSize(UIOBJ *sbar, UINT pageSize);
```

Description

Sets the page size of the scroll bar object.

<i>*sbar</i>	Specifies the scroll bar object.
<i>pageSize</i>	Specifies the value to which the page size of the scroll bar object is copied.

Returned Value

None.

sbarSetPos

Synopsis

```
UINT barSetPos(UIOBJ *sbar, UINT value);
```

Description

Sets the new value of the scroll bar object.

**sbar* Specifies the scroll bar object.
value Specifies the new value of the scroll bar object.

Returned Value

The returned value is the previous value of the scroll bar object if this function succeeds.
Otherwise, the returned value is 0xFFFF.

sbarSetTotalSize

Synopsis

```
void sbarSetTotalSize(UIOBJ *sbar, UINT totalSize);
```

Description

Sets the total size of the scroll bar.

**sbar* Specifies the scroll bar object.
**maxValue* Specifies the maximum value of the scroll bar object.

Returned Value

None.

sbarShowAR

Synopsis

```
void sbarShowAR(UIOBJ *sbar, BOOL showAR1, BOOL showAR2);
```

Description

Sets to display the arrow of the scroll bar or not.

**sbar* Specifies the scroll bar object.
showAR1 Specifies the setting to show UP side (or RIGHT side for horizontal scroll bar) of the arrow. The following are possible values of showAR1:
TRUE Show the UP (or RIGHT side for horizontal scroll bar) side of the arrow
FALSE Hide the UP (or RIGHT side for horizontal scroll bar) side of the arrow
showAR2 Specifies the setting to show DOWN side (or LEFT side for horizontal scroll bar) of the arrow. The following are possible values of showAR2:
TRUE Show the DOWN (or LEFT side for horizontal scroll bar) side of the arrow
FALSE Hide the DOWN (or LEFT side for horizontal scroll bar) side of the arrow

Returned Value

None.

sbarSubTotalSize

Synopsis

```
void sbarSubTotalSize(UIOBJ *sbar, UINT subValue);
```

Description

Subtracts the value to the maximum value of of the scroll bar object.

**sbar* Specifies the scroll bar object.

subValue Specifies the value to be subtracted to the maximum value of the scroll bar object.

Returned Value

None.

Combo Box

General Description

Combo box is composed of one or more than one items, but only one item can be active. To select the active item, user can click on the arrow of the combo box to pull down the list of all items and then click on a certain item. After one of the items is selected, the list disappears and only the new active item is visible. Each item has its ID assigned by user and the item text displayed in the list. Some of the functions return 0xFFFF, which means that the data type of the returned value is unsigned (short/long) integer and the bit value is the same with -1 of signed integer.

Related Data Structure

UIOBJ Structure

L_NODE

System Message

MT_COMBO_COMMAND	Select an item after opening the list of combo box.
pMsg->id	ID of this combo box.
pMsg->data	data.w is the active item ID.
MT_COMBO_FOCUS_IN	Pen position is entering the list after it is opened.
pMsg->id	ID of this combo box.
pMsg->data	data.w is item ID to which the pen points.
MT_COMBO_FOCUS_OUT	Pen position is leaving the list after it is opened.
pMsg->id	ID of this combo box.
pMsg->data	data.w is item ID to which the pen points.
MT_COMBO_FOCUS_MOVE	Pen position is moving in the list after it is opened.
pMsg->id	ID of this combo box.
pMsg->data	data.w is item ID to which the pen points.

comboAddItem

Synopsis

```
BOOL comboAddItem(UIOBJ *combo, WORD id, char *itemText);
```

Description

Adds a new item to combo box object.

**combo* Specifies the combo box object.
id Specifies the item ID of combo box.
**itemText* Specifies the text of this item.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE. GetUILastError() can be called to obtain detailed failure information, and the possible value can be ERR_NO_MEMORY, ERR_ITEM_ID, ERR_OBJECT_HANDLE or ERR_INVALID_PARAMETER.

comboAddItemArray

Synopsis

```
BOOL comboAddItemArray(UIOBJ *combo, L_NODE *item, int itemNum);
```

Description

Adds one or more than one items to combo box object.

**combo* Specifies the combo box object.
**item* Specifies the L_NODE which points to an item array.
itemNum Specifies the number of items to be added.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE. GetUILastError() can be called to obtain detailed failure information, and the possible value can be ERR_OBJECT_HANDLE or ERR_INVALID_PARAMETER.

comboGetItemActive

Synopsis

```
WORD comboGetItemActive(UIOBJ *combo);
```

Description

Obtains the ID of the active item.

**combo* Specifies the combo box object.

Returned Value

The returned value is the ID of the active item if this function succeeds. Otherwise, the returned value is 0xFFFF. GetUILastError() can be called to obtain detailed failure information, and the possible value can be ERR_OBJECT_HANDLE.

comboGetItemArray

Synopsis

```
L_NODE *comboGetItemArray(UIOBJ *combo, L_NODE *item, int itemNum);
```

Description

Obtains information of each item in combo box, including item ID and text. User has to prepare the memory space to which the L_NODE array is copied.

<i>*combo</i>	Specifies the combo box object.
<i>*item</i>	Specifies the an L_NODE array which points to a memory space prepared by user. The size of the memory space must be equal to <i>itemNum</i> * sizeof(L_NODE).
<i>itemNum</i>	Specifies the number of items user wants to retrieve.

Returned Value

The returned value is information of items if this function succeeds. Otherwise, the returned value is NULL. GetUILastError() can be called to obtain detailed failure information, and the possible value can be ERR_OBJECT_HANDLE or ERR_INVALID_PARAMETER.

comboGetItemId

Synopsis

```
WORD comboGetItemId(UIOBJ *combo, char *itemText);
```

Description

Obtains the item ID by item text.

<i>*combo</i>	Specifies the combo box object.
<i>*itemText</i>	Specifies the item text to search for.

Returned Value

The returned value is the item ID if this function succeeds. Otherwise, the returned value is 0xFFFF. GetUILastError() can be called to obtain detailed failure information, and the possible value can be ERR_OBJECT_HANDLE or ERR_INVALID_PARAMETER.

comboGetItemText

Synopsis

```
char *comboGetItemText(UIOBJ *combo, WORD id);
```

Description

Obtains the item text by item ID.

<i>*combo</i>	Specifies the combo box object.
<i>id</i>	Specifies the item ID of combo box.

Returned Value

The returned value is the item text if this function succeeds. Otherwise, the returned value is 0xFFFF. GetUILastError() can be called to obtain detailed failure information, and the possible value can be ERR_OBJECT_HANDLE or ERR_INVALID_PARAMETER.

comboLastActive

Synopsis

```
WORD comboLastActive(UIOBJ *combo);
```

Description

Obtains the ID of the active item. The difference between `comboGetItemActive()` and `comboLastActive()` is the returned value. Since `comboLastActive()` returns 0 either the item ID is 0 or this function fails which may cause ambiguity, `comboGetItemActive()` is recommended for obtaining the ID of the active item.

**combo* Specifies the combo box object.

Returned Value

The returned value is the ID of the active item if this function succeeds. Otherwise, the returned value is 0. `GetUILastError()` can be called to obtain detailed failure information, and the possible value can be `ERR_OBJECT_HANDLE`.

comboMoveLine

Synopsis

```
WORD comboMoveLine(UIOBJ *combo, BOOL isDown);
```

Description

Moves up or down one line of the list of combo box.

**combo* Specifies the combo box object.
isDown Specifies the direction to move.
TRUE Move down
FALSE Move up

Returned Value

The returned value is the ID of the first item in the list if this function succeeds. Otherwise, the returned value is 0. `GetUILastError()` can be called to obtain detailed failure information, and the possible value can be `ERR_OBJECT_HANDLE` or `ERR_INVALID_PARAMETER`.

comboMovePage

Synopsis

```
WORD comboMovePage(UIOBJ *combo, BOOL isDown);
```

Description

Moves up or down one page of the list of combo box.

**combo* Specifies the combo box object.
isDown Specifies the direction to move.
TRUE Move down
FALSE Move up

Returned Value

The returned value is the ID of the first item in the list if this function succeeds. Otherwise, the returned value is 0. GetUILastError() can be called to obtain detailed failure information, and the possible value can be ERR_OBJECT_HANDLE or ERR_INVALID_PARAMETER.

comboNew

Synopsis

```
UIOBJ *comboNew(WORD id, int x, int y, int w, BYTE fontId, BYTE itemInList);
```

Description

Creates a combo box object.

<i>id</i>	Specifies the object ID of this combo box.
<i>x</i>	Specifies x-coordinate of left side of this combo box object, in pixels.
<i>y</i>	Specifies y-coordinate of top side of this combo box object, in pixels.
<i>w</i>	Specifies the width of this combo box object in pixels. The minimal value is 25.
<i>fontId</i>	Specifies the font used by combo box. The available fonts so far are GM_FONT_SMALL, GM_FONT_SMALLBOLD, GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD, and GM_FONT_LAGE. Only GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD, and GM_FONT_LARGE can be applied to Chinese characters.
<i>itemInList</i>	Specifies the number of visible items in the list.

Returned Value

The returned value is the combo box object if this function succeeds. Otherwise, the returned value is NULL. GetUILastError() can be called to obtain detailed failure information, and the possible value can be ERR_OBJECT_ID, ERR_OVER_LIMIT, ERR_NO_MEMORY, ERR_POSITION, ERR_EXEC_COMMAND or ERR_INVALID_PARAMETER.

comboNewEx

Synopsis

```
UIOBJ *comboNewEx(WORD id, int x, int y, int w, BYTE fontId, BYTE itemInList, BYTE style,  
                  DWORD state);
```

Description

Creates a combo box object.

<i>id</i>	Specifies the ID of this combo box object.
<i>x</i>	Specifies x-coordinate of left side of this combo box object, in pixels.
<i>y</i>	Specifies y-coordinate of top side of this combo box object, in pixels.
<i>w</i>	Specifies the width of this combo box object in pixels. The minimal value is 25.
<i>fontId</i>	Specifies the font used by combo box. The available fonts so far are GM_FONT_SMALL, GM_FONT_SMALLBOLD, GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD, and GM_FONT_LAGE. Only GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD, and GM_FONT_LARGE can be applied to Chinese characters.
<i>itemInList</i>	Specifies the number of visible items in the list.
<i>style</i>	Specifies the position of dropdown (arrow) button of the combo box. 0 Set the dropdown button at right side of the combo box. COMBO_ICON_LEFT Set the dropdown button at left side of the combo box.

	COMBO_NO_ICON	Set the combo box without dropdown button at all. This combo box will be expanded when being clicked.
<i>state</i>	Specifies the appearance (UI) state of combo box.	
	0	No frame.
	UI_FRAME	With frame.
	UI_3D	3D frame. To let 3d effective, UI_FRAME has to be applied.

Returned Value

The returned value is the combo box object if this function succeeds. Otherwise, the returned value is NULL. GetUILastError() can be called to obtain detailed failure information, and the possible value can be ERR_OBJECT_ID, ERR_OVER_LIMIT, ERR_NO_MEMORY, ERR_POSITION, ERR_EXEC_COMMAND or ERR_INVALID_PARAMETER.

comboRealHeight

Synopsis

```
int comboRealHeight(UIOBJ *combo);
```

Description

Obtains the height of combo box in pixels, including frame (1 pixel) if it does have. The height is different from the height of opened list.

**combo* Specifies the combo box object.

Returned Value

The returned value is the height of combo box if this function succeeds. Otherwise, the returned value is 0. GetUILastError() can be called to obtain detailed failure information, and the possible value can be ERR_OBJECT_HANDLE.

comboRemoveItem

Synopsis

```
int comboRemoveItem(UIOBJ *combo, WORD id);
```

Description

Removes an item from combo box object by item ID.

**combo* Specifies the combo box object.

id Specifies the item ID of combo box.

Returned Value

The returned value is the number of items of combo box after removal. Otherwise, the returned value is -1. GetUILastError() can be called to obtain detailed failure information, and the possible value can be ERR_ITEM_ID or ERR_OBJECT_HANDLE.

comboRemoveItemAll

Synopsis

```
int comboRemoveItemAll(UIOBJ *combo);
```

Description

Removes all items from combo box object.

**combo* Specifies the combo box object.

Returned Value

The returned value is 1 if this function succeeds. Otherwise, the returned value is 0.

GetUILastError() can be called to obtain detailed failure information, and the possible value can be ERR_OBJECT_HANDLE.

comboSetItemActive

Synopsis

```
BOOL comboSetItemActive(UIOBJ *combo, WORD id);
```

Description

Sets an item to be active (selected).

**combo* Specifies the combo box object.

id Specifies the item ID of combo box object.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

GetUILastError() can be called to obtain detailed failure information, and the possible value can be ERR_ITEM_ID or ERR_OBJECT_HANDLE.

comboTotalItem

Synopsis

```
int comboTotalItem(UIOBJ *combo);
```

Description

Obtains the number of items of combo box object.

**combo* Specifies the combo box object.

Returned Value

The returned value is the number of items of combo box if this function succeeds. Otherwise, the returned value is -1. GetUILastError() can be called to obtain detailed failure information, and the possible value can be ERR_OBJECT_HANDLE.

Tree

General Description

Displays a hierarchical tree structure.

Related Data Structure

TREENODE Structure

This structure stores all data for a tree.

```
typedef struct tagTREENODE{  
    WORD        Step;           // hierarchy step (start 0)  
    char        *pText;         // node string  
    NODE_PARM   *Parm;          // to control struct  
}
```

System Message

MT_TREE_COMMAND	Selects the tree object.
pMsg->id	The ID of this tree.
pMsg->data	The full path data of the the item in the tree.
MT_TREE_ITEM_FOCUS	Select the item of the tree.
pMsg->id	The ID of this tree.
pMsg->data	The content of the the item in the tree.

treeNew

Synopsis

```
UIOBJ *treeNew(WORD id, int x, int y, int w, int pageSize, int style, BYTE fontId,  
               TREENODE *tNode);
```

Description

Creates a tree object.

<i>id</i>	Specifies the ID number of this tree object.
<i>x</i>	Specifies x-coordinate of left side of this tree, in pixels.
<i>y</i>	Specifies y-coordinate of top side of this tree, in pixels.
<i>w</i>	Specifies the width of the tree.
<i>pageSize</i>	Specifies how many rows in one page.
<i>style</i>	Specifies the style of this tree following are possible values: OPEN_ALL Expend all items after drawing the tree object. TREE_SOLID_LINE set the link line to be solid.
<i>fontId</i>	Specifies the ID number of font.
<i>*tNode</i>	Specifies the TREENODE structure which stores all data.

Returned Value

The returned value is a pointer of UIOBJ.

treeRealHeight

Synopsis

```
int treeRealHeight(UIOBJ *tree);
```

Description

Obtains the height of the tree.

<i>*tree</i>	Specifies the tree.
--------------	---------------------

Returned Value

The returned value is the height of the tree if this function succeeds. Otherwise, the returned value is 0.

Single-Line Editor

General Description

Creates a single-line editor.

Related Data Structure

UIOBJ Structure

System Message

T_SINGLEEDIT_COMMAND pMsg->id	Selects an item of the single-line edit object. The ID of this list.
MT_SINGLEEDIT_SELECTED pMsg->id pMsg->data.val.s1 pMsg->data.val.s2	Selects some content of the single-line edit object. The ID of this single-line edit object. The start position of the the selected area. The end position of the the selected area.
MT_SINGLEEDIT_PENUP pMsg->id pMsg->data.val.s1 pMsg->data.val.s2	Pen leave the single -line edit object. The ID of this single -line edit object. The postion of the cursor. The character at the postion of the cursor.
MT_SINGLEEDIT_CHANGE pMsg->id	The content of the single-line edit has been changed. The ID of this single-line edit object.

sedActiveBackSpace

Synopsis

```
void sedActiveBackSpace(void);
```

Description

Deletes the character or space at the left side of the cursor in the single-line editor object.

Returned Value

None.

sedCutSelect

Synopsis

```
void sedCutSelect(UIOBJ *seditor);
```

Description

Deletes the content of the selection in the single-line editor object.

**seditor* Specifies the single-line editor object.

Returned Value

None.

sedDeatchData

Synopsis

```
void sedDeatchData(UIOBJ *seditor);
```

Description

Deletes the character or space to the right of the cursor in the single-line editor object.

**seditor* Specifies the single-line editor object.

Returned Value

None.

sedDeleteStr

Synopsis

```
void sedDeleteStr(UIOBJ *seditor);
```

Description

Deletes the content of the selection in the single-line editor object.

**seditor* Specifies the single-line editor object.

Returned Value

None.

sedGetEditText

Synopsis

```
VHANDLE sedGetEditText(UIOBJ *seditor);
```

Description

Obtains the context of the single-line editor object.

**seditor* Specifies the single-line editor object.

Returned Value

The returned value is the handle containing the text if this function succeeds. Otherwise, the returned value is 0.

sedGetMaxCnt

Synopsis

```
WORD sedGetMaxCnt(UIOBJ *seditor);
```

Description

Sets the maximum limit of the chars to display in the single-line editor.

**seditor* Specifies the single-line editor object.

Returned Value

The returned value is the maximum limit of the chars to display in the single-line editor.

sedGetModify

Synopsis

```
BYTE sedGetModify(UIOBJ *seditor);
```

Description

Obtains the flag that indicate the single-line editor object has been modified or not

**seditor* Specifies the single-line editor object.

Returned Value

The returned value is 1 if the single-line editor object has been modified. Otherwise, the returned value is 0.

sedGetNumericFmt

Synopsis

```
BYTE sedGetNumericFmt(UIOBJ *seditor);
```

Description

Obtains the number of the digits after fixed-point in single-line editor.

**seditor* Specifies the single-line editor object.

Returned Value

The returned value is the setting of the number of the digits after fixed-point if this function succeeds. Otherwise, the returned value is 0xFF.

sedGetSelect

Synopsis

```
void sedGetSelect(UIOBJ *seditor, WORD *start, WORD *end);
```

Description

Obtains the the start position and the end position of the selection in the single-line editor.

**seditor* Specifies the single-line editor object.
**start* Specifies the start position of the selection to which the value is copied.
**end* Specifies the end position of the selection to which the value is copied.

Returned Value

None.

sedGetTextLength

Synopsis

```
WORD sedGetTextLength(UIOBJ *seditor);
```

Description

Obtains the string length of the context in the single-line editor object.

**seditor* Specifies the single-line editor object.

Returned Value

The returned value is the string length of the context if this function succeeds. Otherwise, the returned value is 0.

sedInfoA

Synopsis

```
BOOL sedInfoA(UIOBJ *seditor, WORD *beginPos, WORD *cursorPos);
```

Description

Obtains the Information of the single-line editor.

**seditor* Specifies the single-line editor object.

- *beginPos* Specifies the number to which the start position of the content In the single-line editor object is copied.
- *cursorPos* Specifies the number to which the position of the cursor. In the single-line editor object is copied

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

sedInfoB

Synopsis

```
BOOL sedInfoB(UIOBJ *seditor, WORD *height, WORD *width);
```

Description

Obtains the Information of the single-line editor.

- *seditor* Specifies the single-line editor object.
- *height* Specifies the number to which the height of the single-line editor object is copied.
- *width* Specifies the number to which the weight of the single-line editor object is copied.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

sedMove

Synopsis

```
BOOL sedMove(UIOBJ *seditor, WORD x, WORD y, WORD w, BYTE fontId, WORD position,
             DWORD attribs);
```

Description

Resets some setting of a single-line editor object.

- *seditor* Specifies the single-line editor object.
- x* Specifies x-coordinate of left side of this single-line editor object, in pixels.
- y* Specifies y-coordinate of top side of this single-line editor object, in pixels.
- w* Specifies the width of this single-line editor object.
- fontId* Specifies the font used by single-line editor. The available fonts so far are GM_FONT_SMALL, GM_FONT_SMALLBOLD, GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD, and GM_FONT_LAGE. Only GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD, and GM_FONT_LARGE can be applied to Chinese characters.
- contentPos* Specifies the content position of this single-line editor object.
- attribs* Specifies the attribute of this single-line editor object.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

sedNew

Synopsis

```
UIOBJ *sedNew(unsigned short id, short x, short y, short cx, BYTE style, FONTID fontId,  
             BYTE dockMode);
```

Description

Creates a new single-line editor object.

<i>id</i>	Specifies the unique ID number of this single-line editor object.
<i>x</i>	Specifies the x-coordinate.
<i>y</i>	Specifies the y-coordinate.
<i>cx</i>	Specifies the width for this edit in pixel.
<i>style</i>	Specifies the style of the single-line editor object. The following are possible values: ES_STYLE_NORMAL, ES_STYLE_PASSWORD, ES_STYLE_NUMERIC, ES_STYLE_ALIGN_RIGHT, ES_STYLE_NO_BORDER, ES_STYLE_NO_EXTERN, ES_STYLE_READONLY, ES_STYLE_NO_POPUP_KB.
<i>fontId</i>	Specifies the ID number of fonts.
<i>dockMode</i>	Specifies the position of the viewpoint. The following are possible values: ES_UI_DOCK_TOP, ES_UI_DOCK_RIGHT, ES_UI_DOCK_BOTTOM.

Returned Value

The returned value is a pointer of UIOBJ.

sedNewEx

Synopsis

```
UIOBJ *sedNewEx(WORD id, WORD x, WORD y, WORD w, BYTE fontId, WORD mode, DWORD attrs);
```

Description

Creates a new single-line editor object.

<i>id</i>	Specifies the unique ID number of this single-line editor object.
<i>x</i>	Specifies x-coordinate of left side of this single-line editor object, in pixels.
<i>y</i>	Specifies y-coordinate of top side of this single-line editor object, in pixels.
<i>w</i>	Specifies the width of this single-line editor object.
<i>fontId</i>	Specifies the font used by label. The available fonts so far are GM_FONT_SMALL, GM_FONT_SMALLBOLD, GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD, and GM_FONT_LARGE. Only GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD, and GM_FONT_LARGE can be applied to Chinese characters.
<i>mode</i>	Specifies the mode of this single-line editor object.
<i>attrs</i>	Specifies the attribute of this single-line editor object.

Returned Value

The returned value is a pointer of UIOBJ.

sedSelectAll

Synopsis

```
void sedSelectAll(UIOBJ *seditor);
```

Description

Sets all the content of the single-line editor object to be selected.

**seditor* Specifies the single-line editor object.

Returned Value

None.

sedSetBeginPos

Synopsis

```
void sedSetCurPos(UIOBJ *seditor, WORD pos);
```

Description

Sets the position of the cursor in the single-line editor.

**seditor* Specifies the single-line editor object.
pos Specifies the character number of the content to be shown at the left side in the single-line editor object.

Returned Value

None.

sedSetCurPos

Synopsis

```
void sedSetCurPos(UIOBJ *seditor, WORD pos);
```

Description

Sets the position of the cursor in the single-line editor.

**seditor* Specifies the single-line editor object.
pos Specifies the position of the cursor.

Returned Value

None.

sedSetEditText

Synopsis

```
int sedSetEditText(UIOBJ *seditor, const char *content);
```

Description

Sets the context of the single-line editor object.

**seditor* Specifies the single-line editor object.
**content* Specifies the content of the single-line editor object.

Returned Value

The returned value is 1 if this function succeeds. Otherwise, the returned value is 0.

sedSetEditTextEx

Synopsis

```
int sedSetEditTextEx(UIOBJ *seditor, const char *content, BOOL redraw);
```

Description

Sets the context of the single-line editor object.

<i>*seditor</i>	Specifies the single-line editor object.
<i>*content</i>	Specifies the content of the single-line editor object.
<i>redraw</i>	Specifies the setting to redraw the single-line editor immediately or not.
TRUE	Redraw the single-line editor immediately.
FALSE	Don't redraw the single-line editor immediately.

Returned Value

The returned value is 1 if this function succeeds. Otherwise, the returned value is 0.

sedSetFocus

Synopsis

```
void sedSetFocus(UIOBJ *seditor);
```

Description

Sets the single-line editor object to get focus.

<i>*seditor</i>	Specifies the single-line editor object.
-----------------	--

Returned Value

None.

sedSetMaxCnt

Synopsis

```
void sedSetMaxCnt(UIOBJ *seditor, WORD cnt);
```

Description

Sets the maximum limit of the chars to display in the single-line editor.

<i>*seditor</i>	Specifies the single-line editor object.
<i>cnt</i>	Specifies the Maximum amount of text, in bytes.

Returned Value

None.

sedSetNumericFmt

Synopsis

```
void sedSetNumericFmt(UIOBJ *seditor, BYTE fmt);
```

Description

Sets the number of the digits after fixed-point in single-line editor.

<i>*seditor</i>	Specifies the single-line editor object.
<i>fmt</i>	Specifies the number of the digits after fixed-point. The following are possible values:
ES_NUM_FORMAT_NO_DOT	Shows no digits after fixed-point.
ES_NUM_FORMAT_DOT1	Shows one digits after fixed-point.
ES_NUM_FORMAT_DOT2	Shows two digits after fixed-point.
ES_NUM_FORMAT_DOT3.	Shows three digits after fixed-point.

Returned Value

None.

sedSetSelect

Synopsis

```
void sedSetSelect(UIOBJ *seditor, WORD start, WORD end);
```

Description

Sets the the start position and the end position of the selection in the single-line editor.

<i>*seditor</i>	Specifies the single-line editor object.
<i>start</i>	Specifies the start position of the selection.
<i>end</i>	Specifies the end position of the selection.

Returned Value

None.

sedSetModify

Synopsis

```
void sedSetModify(UIOBJ *seditor, BOOL isModify);
```

Description

Sets the flag that indicate the content of the single-line editor object has been modified or not

<i>*seditor</i>	Specifies the single-line editor object.
<i>isModify</i>	Specifies the flag for the single-line editor object. The following are possible values:
1	Indicate the content of the single-line editor object has been modified.
0	Indicate the content of the single-line editor object remain unchanged.

Returned Value

None.

Multiple-Line Editor

General Description

Creates a multiple-line editor.

Related Data Structure

UIOBJ Structure

System Message

MT_MULTIPLE_EDIT_COMMAND pMsg->id	Selects the multiple-line edit object. The ID of this multiple-line edit object.
MT_MULEDIT_SELECTED pMsg->id pMsg->data.val.s1 pMsg->data.val.s2	Selects some content of the multiple-line edit object. The ID of this multiple-line edit object. The start position of selected area in the multiple-line edit. The end position of selected area in the multiple-line edit.
MT_MULEDIT_PENUP pMsg->id pMsg->data.val.s1 pMsg->data.val.s2	Pen leave the multiple-line edit object. The ID of this multiple-line edit object. The postion of the cursor. The character at the postion of the cursor.
MT_MULEDIT_LINECHANGE pMsg->id pMsg->data.val.s1 pMsg->data.val.s2	Adds a new line int the multiple-line edit object. The ID of this multiple-line edit object. The difference of the line number. The total line number.
MT_MULEDIT_CHANGE pMsg->id	The content of the multiple-line edit has been changed. The ID of this multiple-line edit object.

medActiveBackSpace

Synopsis

```
void medActiveBackSpace(void);
```

Description

Deletes the character or space at the left side of the cursor in the multiple-line editor object.

Returned Value

None.

medAppendStr

Synopsis

```
WORD medAppendStr(UIOBJ *meditor, const char *string);
```

Description

Adds a string at the end of the multiple-line editor.

**meditor* Specifies the multiple-line editor object.
**string* Specifies the text string to be added.

Returned Value

The returned value is the number of words being added if this function succeeds. Otherwise, the returned value is 0.

medAppendStrEx

Synopsis

```
WORD medAppendStrEx(UIOBJ *meditor, const char *string, BOOL redraw)
```

Description

Adds a string at the end of the multiple-line editor.

**meditor* Specifies the multiple-line editor object.
**string* Specifies the text string to be added.
redraw Specifies the flag of rerefreshing multiple-line editor object.
 TRUE Refresh multiple-line editor
 FALSE Do not refresh multiple-line editor

Returned Value

The returned value is the number of words being added if this function succeeds. Otherwise, the returned value is 0.

medDeleteStr

Synopsis

```
void medDeleteStr(UIOBJ *meditor);
```

Description

Clears all the content of the multiple-line editor.

**meditor* Specifies the multiple-line editor object.

Returned Value

None.

medGetCurLineNum

Synopsis

```
WORD medGetCurLineNum(UIOBJ *meditor);
```

Description

Obtains current position of the cursor in line numbers.

**meditor* Specifies the multiple-line editor object.

Returned Value

The returned value is the line number if the function succeeds. Otherwise, the returned value is 0.

medGetCursorCurrentPos

Synopsis

```
WORD medGetCursorCurrentPos(UIOBJ *meditor);
```

Description

Obtains the current cursor position in char.

**meditor* Specifies the multiple-line editor object.

Returned Value

The returned value is the current cursor position in char. if the function succeeds. Otherwise, the returned value is 0xFFFF.

medGetEditText

Synopsis

```
VHANDLE medGetEditText(UIOBJ *meditor);
```

Description

Obtains the text of the multiple-line editor.

**meditor* Specifies the multiple-line editor object.

Returned Value

The returned value is the handle of the text if this function succeeds. Otherwise, the returned value

is NULL.

medGetMaxCnt

Synopsis

```
WORD medGetMaxCnt(UIOBJ *meditor);
```

Description

Obtains the text limit for the multiple-line editor.

**meditor* Specifies the multiple-line editor object.

Returned Value

The returned value is the maximum amount of the text if the function succeeds. Otherwise, the returned value is 0.

medGetModify

Synopsis

```
BYTE medGetModify(UIOBJ *meditor)
```

Description

Obtains the modified pointer of the multiple-line editor. If this editor is not modified, it will be set to 0.

**meditor* Specifies the multiple-line editor object.

Returned Value

The returned value is 1 if this this object is modified. Otherwise, the returned value is 0.

medGetSelect

Synopsis

```
void medGetSelect(UIOBJ *meditor, WORD *start, WORD *end);
```

Description

Obtains the status of the current selectio.

**meditor* Specifies the multiple-line editor object.
**start* Specifies the memory buffer to which the starting position of selection is copied.
**end* Specifies the memory buffer to which the ending position of selection is copied.

Returned Value

None.

medGetTextLength

Synopsis

```
WORD medGetTextLength(UIOBJ *meditor);
```

Description

Obtains the length of the text of the multiple-line editor.

**meditor* Specifies the multiple-line editor object.

Returned Value

The returned value is the text length if the function succeeds. Otherwise, the returned value is 0.

medGetTotalLine

Synopsis

```
WORD medGetTotalLine(UIOBJ *meditor);
```

Description

Obtains the number of the lines.

**meditor* Specifies the multiple-line editor object.

Returned Value

The returned value is the line number if the function succeeds. Otherwise, the returned value is 0.

medGetViewPos

Synopsis

```
WORD medGetViewPos(UIOBJ *meditor)
```

Description

Obtains the position of the view.

**meditor* Specifies the multiple-line editor object.

Returned Value

The returned value is 1 if this function succeeds. Otherwise, the returned value is 0.

medInfoA

Synopsis

```
BOOL medInfoA(UIOBJ *meditor, int *visibleLine, int *topLine);
```

Description

Sets the position of the cursor to be shown.

**meditor* Specifies the multiple-line editor object.

**visibleLine* Specifies the number to which the visible lines number in the multiple edit is copied.

**topLine* Specifies the number to which the first line shown in the multiple edit is copied.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

medInfoB

Synopsis

```
BOOL medInfoB(UIOBJ *meditor, int *x, int *y);
```

Description

Obtains the information of the multiple-line editor.

<i>*meditor</i>	Specifies the multiple-line editor object.
<i>*x</i>	Specifies the memory buffer to which the x-coordinate of the cursor position is copied.
<i>*y</i>	Specifies the memory buffer to which the y-coordinate of the cursor position is copied.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

medInfoC

Synopsis

```
BOOL medInfoC(UIOBJ *meditor, int *lineHeight, int *totHeight);
```

Description

Obtains the information of the multiple-line editor.

<i>*meditor</i>	Specifies the multiple-line editor object.
<i>*lineHeight</i>	Specifies the memory buffer to which the height per line is copied.
<i>*totHeight</i>	Specifies the memory buffer to which the height of the multiple-line editor is copied.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

medInsertStr

Synopsis

```
void medInsertStr(UIOBJ *meditor, const char *string);
```

Description

Inserts a string to the multiple-line editor at the cursor position.

<i>*meditor</i>	Specifies the multiple-line editor object.
<i>*string</i>	Specifies the text string to be inserted.

Returned Value

None.

medIsOnFocus

Synopsis

```
BOOL medIsOnFocus(UIOBJ *meditor);
```

Description

Checks the status of input focus.

**meditor* Specifies the multiple-line editor object.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

medMove

Synopsis

```
BOOL medMove(UIOBJ *meditor, int x, int y, int w, int lineNo, int style, int fontId);
```

Description

Moves the current object.

**meditor* Specifies the multiple-line editor object.

x Specifies the new x-coordinate of left side of this multiple-line editor object, in pixels.

y Specifies the new y-coordinate of top side of this multiple-line editor object, in pixels.

w Specifies the new width of this multiple-line editor object.

lineNo Specifies the new line number.

style Specifies the new style. The possible styles are EM_STYLE_HAVE_BORDER, EM_STYLE_NO_BORDER, EM_STYLE_READONLY, EM_STYLE_NO_SHOWENTER, EM_STYLE_NO_UNLINE, EM_STYLE_NO_POPUP_KB and EM_STYLE_NO_AUTOCLIP.

fontId Specifies the character font. The available fonts so far are GM_FONT_SMALL, GM_FONT_SMALLBOLD, GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD, and GM_FONT_LARGE. Only GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD and GM_FONT_LARGE can be applied to Chinese characters.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

medNew

Synopsis

```
UIOBJ *medNew(unsigned short id, short x, short y, WORD cx, WORD lineNo,  
              BYTE style, BYTE fontId);
```

Description

Creates a new multiple-line editor object.

id Specifies the object ID of this multiple-line editor.

x Specifies the new x-coordinate of left side of this multiple-line editor object, in pixels.

y Specifies the new y-coordinate of top side of this multiple-line editor object, in pixels.

cx Specifies the width of this toolbar in pixels. This should be more than 3.

<i>lineNo</i>	Specifies the new line number.
<i>style</i>	Specifies the new style. The possible styles are EM_STYLE_HAVE_BORDER, EM_STYLE_NO_BORDER , EM_STYLE_READONLY, EM_STYLE_NO_SHOWENTER, EM_STYLE_NO_UNLINE, EM_STYLE_NO_POPUP_KB and EM_STYLE_NO_AUTOCLIP.
<i>fontId</i>	Specifies the character font. The available fonts so far are GM_FONT_SMALL, GM_FONT_SMALLBOLD, GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD, and GM_FONT_LAGE. Only GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD and GM_FONT_LARGE can be applied to Chinese characters.

Returned Value

The returned value is a pointer of UIOBJ if this function succeeds. Otherwise, the returned value is NULL.

medNewEx

Synopsis

```
UIOBJ *medNewEx(WORD id, WORD x, WORD y, WORD w, BYTE fontId, WORD lineNum, DWORD attrib);
```

Description

Creates a new multipl edit object.

<i>id</i>	Specifies the unique ID number of this multiple-line editor object.
<i>x</i>	Specifies x-coordinate of left side of this multiple-line editor object, in pixels.
<i>y</i>	Specifies y-coordinate of top side of this multiple-line editor object, in pixels.
<i>w</i>	Specifies the width of this multiple-line editor object.
<i>fontId</i>	Specifies the character font. The available fonts so far are GM_FONT_SMALL, GM_FONT_SMALLBOLD, GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD, and GM_FONT_LAGE. Only GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD and GM_FONT_LARGE can be applied to Chinese characters.
<i>lineNum</i>	Specifies the number of lines in the multiple-line editor object.
<i>attrib</i>	Specifies the attributes of the multiple-line editor object.

Returned Value

The returned value is a pointer of UIOBJ.

medScrollDown

Synopsis

```
void medScrollDown(UIOBJ *meditor);
```

Description

Moves down one line in multiple-line editor.

**meditor* Specifies the multiple-line editor object.

Returned Value

None.

medScrollPageDown

Synopsis

```
void medScrollPageDown(UIOBJ *meditor);
```

Description

Moves down one page in multiple-line editor.

**meditor* Specifies the multiple-line editor object.

Returned Value

None.

medScrollPageUp

Synopsis

```
void medScrollPageUp(UIOBJ *meditor);
```

Description

Moves up one page in multiple-line editor.

**meditor* Specifies the multiple-line editor object.

Returned Value

None.

medScrollUp

Synopsis

```
void medScrollUp(UIOBJ *meditor);
```

Description

Moves up one line in multiple-line editor.

**meditor* Specifies the multiple-line editor object.

Returned Value

None.

medSelectAll

Synopsis

```
void medSelectAll(UIOBJ *meditor);
```

Description

Sets the object to be all selected.

**meditor* Specifies the multiple-line editor object.

Returned Value

None.

medSetCurPos

Synopsis

```
void medSetCurPos(UIOBJ *meditor, WORD pos);
```

Description

Sets the position of the cursor.

**meditor* Specifies the multiple-line editor object.
pos Specifies the new position of the cursor.

Returned Value

None.

medSetEditFont

Synopsis

```
BYTE medSetEditFont(UIOBJ *meditor, BYTE fontId);
```

Description

Sets the font of this object.

**meditor* Specifies the multiple-line editor object.
fontId Specifies the character font. The available fonts so far are GM_FONT_SMALL, GM_FONT_SMALLBOLD, GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD, and GM_FONT_LARGE. Only GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD and GM_FONT_LARGE can be applied to Chinese characters.

Returned Value

The returned value is the previous font if this function succeeds. Otherwise, the returned value is 0xFF.

medSetEditText

Synopsis

```
int medSetEditText(UIOBJ *meditor, const char *string);
```

Description

Changes the text in the multiple-line editor.

**meditor* Specifies the multiple-line editor object.
**string* Specifies the new text string to be set.

Returned Value

The returned value is 1 if this function succeeds. Otherwise, the returned value is 0.

medSetTextEx

Synopsis

```
int medSetTextEx(UIOBJ *meditor, const char *string, BOOL isRefresh);
```

Description

Changes the text in the multiple-line editor with an option of refreshing screen.

<i>*meditor</i>	Specifies the multiple-line editor object.
<i>*string</i>	Specifies the new text string to be set.
<i>isRefresh</i>	Specifies the flag of refreshing screen.
TRUE	Refresh the screen.
FALSE	Do not refresh the screen.

Returned Value

The returned value is 1 if this function succeeds. Otherwise, the returned value is 0.

medSetFocus

Synopsis

```
void medSetFocus(UIOBJ *meditor);
```

Description

Sets the object focus to this multiple-line editor.

<i>*meditor</i>	Specifies the multiple-line editor object.
-----------------	--

Returned Value

None.

medSetMaxCnt

Synopsis

```
void medSetMaxCnt(UIOBJ *meditor, WORD count);
```

Description

Sets the maximum number of characters in the multiple-line editor.

<i>*meditor</i>	Specifies the multiple-line editor object.
<i>count</i>	the maximum number of characters in bytes.

Returned Value

None.

medSetModify

Synopsis

```
void medSetModify(UIOBJ *meditor, BOOL isEditable);
```

Description

Sets the text of the multiple-line editor to be editable or not.

<i>*meditor</i>	Specifies the multiple-line editor object.
<i>isEditable</i>	Specifies the flag of editing.
TRUE	Editable
FALSE	Non-editable

Returned Value

None.

medSetSelect

Synopsis

```
void medSetSelect(UIOBJ *meditor, WORD start, WORD end);
```

Description

Sets the status of current selection.

<i>*meditor</i>	Specifies the multiple-line editor object.
<i>start</i>	Specifies the starting position of selection is copied.
<i>end</i>	Specifies the ending position of selection is copied.

Returned Value

None.

medSetViewPos

Synopsis

```
BYTE medSetViewPos(UIOBJ *meditor, WORD y);
```

Description

Sets the view position.

<i>*meditor</i>	Specifies the multiple-line editor object.
<i>y</i>	Specifies the y-coordinate of the view position.

Returned Value

The returned value is 1 if this function succeeds. Otherwise, the returned value is 0.

medSetViewPosEx

Synopsis

```
BYTE medSetViewPosEx(UIOBJ *meditor, WORD y, BOOL isRefresh);
```

Description

Sets the view position with a refreshing option.

<i>*meditor</i>	Specifies the multiple-line editor object.
<i>y</i>	Specifies the y-coordinate of the view position.
<i>isRefresh</i>	Specifies the flag of refreshing screen.
	TRUE Refresh the screen.
	FALSE Do not refresh the screen.

Returned Value

The returned value is 1 if this function succeeds. Otherwise, the returned value is 0.

Field

General Description

Creates a field table with flexible width.

Related Data Structure

FldDes Structure

This structure stores all information of a field object.

```
typedef struct tag_DbField {
    WORD SrcType      0 (DB type)
    void *pSrcDes      FldDBDes struct
    FldDBDes structure:
    Variable          Description
    char *Dbfilename  filename
    WORD fldCnt       the field number to display
    WORD *FieldWidthA the field width
    char **pfldNameA  the name of the data field
    BYTE FontID       the font size
    BYTE TitleFontID  title id
    BYTE FrameColor   the shade of frame color
    BYTE Unused       0 (unused)
    WORD Style        Fld type
                     FLD_STYLE_FRAME
                     FLD_STYLE_TITLE
                     FLD_STYLE_LINEMODE
                     FLD_STYLE_HIDE_DELITEM
                     FLC_SYTLE_RESERVE
                     FLC_SYTLE_MULTI_RESERVE
}
```

System Message

MT_FIELD_COMMAND	Select the field object.
pMsg->id	The ID of this field.
pMsg->data	The number of the recoed.
MT_FIELD_SELECT	Double clicks the field object.
pMsg->id	The ID of this field.
pMsg->data.val.s1	The number of the index.
pMsg->data.val.s2	The number of the record.
MT_FIELD_STAND	The pen down on the field object, and don' t move.
pMsg->id	The ID of this field.
pMsg->data.val.s1	The number of the index.
pMsg->data.val.s2	The number of the record.
MT_FIELD_ITEM_COMMAND	Selects the item of the field object.
pMsg->id	The ID of this field.
pMsg->data.val.s1	The number of the index.
pMsg->data.val.s2	The number of the record.
MT_FIELD_FOCUS_IN	Move pen on touch panel, and let the field object get focus.
pMsg->id	The ID of this field.
pMsg->data.val.s1	The number of the index.

pMsg->data.val.s2	The number of the record.
MT_FIELD_FOCUS_OUT	Move pen on touch panel, and let the field object lose focus.
pMsg->id	The ID of this field.
pMsg->data.val.s1	The number of the index.
pMsg->data.val.s2	The number of the record.
MT_FIELD_FOCUS_MOVE	Move pen on touch panel in the field object, and when different record or item get selected.
pMsg->id	The ID of this field.
pMsg->data.val.s1	The number of the index.
pMsg->data.val.s2	The number of the record.
MT_FIELD_CHANGE	Reset the order of sort or the column' s width.
pMsg->id	Indicates the order of sort or the column' s width has been changed.
	The following are the possible value:
	FLD_CHANGE_SORT The order of sort has been changed.
	FLD_CHANGE_WIDTH The column' s width has been changed.
pMsg->data.val.s2	The value for change.
	If pMsg->id == FLD_CHANGE_SORT, the value indicates the number of the column.
	If pMsg->id == FLD_CHANGE_WIDTH, the value indicates the number of the changable line.

fldAddRecord

Synopsis

```
WORD fldAddRecord(UIOBJ *field, WORD recordId, const char *content);
```

Description

Appends a record to the field object.

<i>*field</i>	Specifies the field object.
<i>recordId</i>	Specifies the record ID.
<i>*content</i>	Specifies the content to be appended.

Returned Value

The returned value is 1 if this function succeeds. Otherwise, the returned value is 0.

fldClearReserve

Synopsis

```
void fldClearReserve(UIOBJ *field);
```

Description

Deselects the reserve items in the field.

<i>*field</i>	Specifies the field object.
---------------	-----------------------------

Returned Value

None.

fldDelAllRecord

Synopsis

```
void fldDelAllRecord(UIOBJ *field);
```

Description

Deletes all records int the field object.

<i>*field</i>	Specifies the field object.
---------------	-----------------------------

Returned Value

None.

fldDelRecord

Synopsis

```
void fldDelRecord(UIOBJ *field, WORD recordId);
```

Description

Deletes a row from the field object.

**field* Specifies the field object.
recordId Specifies the record ID.

Returned Value

None.

fldEnableRepeat

Synopsis

```
void fldEnableRepeat(UIOBJ *field, int isEnabled, DWORD delay);
```

Description

Enables or disables the repeat function of the field object or not.

**field* Specifies the field object.
isEnabled Specifies the setting for repeat function. The following are possible values:
1 Enable the repeat function.
0 Disable the repeat function.
delay Specifies the setting for the time interval of the repeat function in milliseconds.

Returned Value

None.

fldGetRecord

Synopsis

```
WORD fldGetRecord(UIOBJ *field, WORD recordId, WORD idx, void *buffer, WORD size);
```

Description

Obtains the data from the item in the field object.

**field* Specifies the field object.
recordId Specifies the record ID.
idx Specifies the item index.
**buffer* Specifies the memory buffer to which the data is copied.
size Specifies the length of the memory buffer.

Returned Value

The returned value is the data number in bytes if this function succeeds. Otherwise, the returned value is 0.

fldGetRecordReserve

Synopsis

```
WORD fldGetRecordReserve(UIOBJ *field, WORD reocrdNum, WORD *recordId);
```

Description

Obtains all reserved rows in the file object.

<i>*field</i>	Specifies the field object.
<i>reocrdNum</i>	Specifies the total reserved records.
<i>*recordId</i>	Specifies the buffer to which the ID of the selected records is copied.

Returned Value

The returned value is 1 if this function succeeds. Otherwise, the returned value is 0.

fldGetRecordReserveNum

Synopsis

```
WORD fldGetRecordReserveNum(UIOBJ *field) ;
```

Description

Obtains the number of the selected records in the field objec.

<i>*field</i>	Specifies the field object.
---------------	-----------------------------

Returned Value

The returned value is the number of the selected rows in the field objec. if this function succeeds. Otherwise, the returned value is 0.

fldGetTop

Synopsis

```
WORD fldGetTop(UIOBJ *field) ;
```

Description

Obtains the number of the record which shown at the top of the visual windows.

<i>*field</i>	Specifies the a UIOBJ.
---------------	------------------------

Returned Value

The returned value is field number if this function succeeds. Otherwise, the returned value is 0.

fldGetWidth

Synopsis

```
WORD fldGetWidth(UIOBJ *field, WORD columnNo, WORD *width) ;
```

Description

Obtains the width of the column in the field object.

<i>*field</i>	Specifies the field object.
<i>columnNo</i>	Specifies the column number.
<i>*width</i>	Specifies the number to which the width of the column is copied.

Returned Value

The returned value is 1 if this function succeeds. Otherwise, the returned value is 0.

fldMove

Synopsis

```
BOOL fldMove(UIOBJ *field, FldDes *fieldDes, WORD x, WORD y, WORD w, WORD h);
```

Description

Resets some setting of a field object.

<i>*field</i>	Specifies the field object.
<i>*fieldDes</i>	Specifies the FldDes structure that provide the data source for field object.
<i>x</i>	Specifies x-coordinate of left side of this field object, in pixels.
<i>y</i>	Specifies y-coordinate of top side of this field object, in pixels.
<i>w</i>	Specifies the width of this field object.
<i>h</i>	Specifies the height of this field object.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

fldMoveLine

Synopsis

```
WORD fldMoveLine(UIOBJ *field, BOOL isDown);
```

Description

Moves up or down one line.

<i>*field</i>	Specifies the a UIOBJ.
<i>isDown</i>	Specifies the direction flag of movement. TRUE for moving down; FALSE for moving up.

Returned Value

The returned value is field number.

fldMovePage

Synopsis

```
WORD fldMovePage(UIOBJ *field, BOOL isDown);
```

Description

Moves up or down one page.

<i>*field</i>	Specifies the a UIOBJ.
<i>isDown</i>	Specifies the direction flag of movement. TRUE for moving down; FALSE for moving up.

Returned Value

The returned value is field number.

fldNew

Synopsis

```
UIOBJ *fldNew(WORD id, FldDes *fieldDes, WORD x, WORD y, WORD w, WORD h);
```

Description

Creates a field object.

<i>id</i>	Specifies the handle of the database.
* <i>fieldDes</i>	Specifies the FldDes structure.
<i>x</i>	Specifies the x-coordinate of the screen position.
<i>y</i>	Specifies the y-coordinate of the screen position.
<i>w</i>	Specifies the width of the field.
<i>h</i>	Specifies the height of the field.

Returned Value

The returned value is a pointer of FldDes structure if this function succeeds. Otherwise, the returned value is NULL.

fldNewEx

Synopsis

```
UIOBJ *fldNew(WORD id, FldDes *fieldDes, WORD x, WORD y, WORD w, WORD h, DWORD style);
```

Description

Creates a field object.

<i>id</i>	Specifies the unique ID number of this field object.
* <i>fieldDes</i>	Specifies the FldDes structure that provide the data source for the field object.
<i>x</i>	Specifies x-coordinate of left side of this field object, in pixels.
<i>y</i>	Specifies y-coordinate of top side of this field object, in pixels.
<i>w</i>	Specifies the width of this field object.
<i>h</i>	Specifies the height of this field object.
<i>style</i>	Specifies the style of this field object.

Returned Value

The returned value is a pointer of UIOBJ.

fldSetRecordReserve

Synopsis

```
WORD fldSetRecordReserve(UIOBJ *field, WORD recordId);
```

Description

Sets a record in the field object to be selected.

* <i>field</i>	Specifies the field object.
<i>recordId</i>	Specifies the record ID.

Returned Value

The returned value is 1 if this function succeeds. Otherwise, the returned value is 0.

fldSetWidth

Synopsis

```
WORD fldSetWidth(UIOBJ *field, WORD columnNum, WORD *columnWidth);
```

Description

Sets the width of the column.

<i>*field</i>	Specifies the field object.
<i>columnNum</i>	Specifies the column number to be set the width.
<i>*columnWidth</i>	Specifies the width for the column.

Returned Value

The returned value is 1 if this function succeeds. Otherwise, the returned value is 0.

fldSetReserve

Synopsis

```
void fldSetReserve(UIOBJ *field, WORD dispPos);
```

Description

Sets one record in the field object to be reserved.

<i>*field</i>	Specifies the a UIOBJ.
<i>dispPos</i>	Specifies the number of the record to be reserved.

Returned Value

None.

fldSetSort

Synopsis

```
void fldSetSort(UIOBJ *field, WORD idx, WORD option);
```

Description

Sets the sort setting of the column.

<i>*field</i>	Specifies the field object.
<i>idx</i>	Specifies the index number of the column in the field to be sorted.
<i>option</i>	Specifies the setting to be sort or redraw. The following are possible values: FLD_SORT_DECREASE The sort order is decrease. FLD_SORT_REDRAW Redraw automatically.

Returned Value

None.

fldSetSpecialReserve

Synopsis

```
WORD fldSetSpecialReserve(UIOBJ *field, WORD type);
```

Description

Sets the reserve area

<i>*field</i>	Specifies the field object.
<i>type</i>	Specifies the reserve type. The following are possible values:
	FLD_RESERVE_ALL All items in the field be selected.
	FLD_RESERVE_INVERCE Inverse the selected state of the items in the field.

Returned Value

The returned value is 1 if this function succeeds. Otherwise, the returned value is 0.

fldSetTop

Synopsis

```
WORD fldSetTop(UIOBJ *field, WORD dispPos);
```

Description

Sets the record to be shown at the top of the visual windows.

<i>*field</i>	Specifies the a UIOBJ.
<i>dispPos</i>	Specifies the number of the record to be shown.

Returned Value

The returned value is field number if this function succeeds. Otherwise, the returned value is 0.

fldTotalRecord

Synopsis

```
WORD fldTotalRecord(UIOBJ *field);
```

Description

Obtains the a row to be selected.

<i>*field</i>	Specifies the field object.
---------------	-----------------------------

Returned Value

The returned value is the total records if this function succeeds. Otherwise, the returned value is 0.

CUI

General Description

In this section, some graphical API' s are provided to access system resources.

Related Data Structure

SearchParam Structure

This structure stores all information of a field of database.

System Message

MT_KEYBOARD_COMMAND pMsg->id	Select a key of the keyboard. The ID of this keyboard.
MT_KEYBOARD_FUNC pMsg->id pMsg->data.d	Selects a function key of the keyboard The ID of this keyboard. The ID of the function key.
MT_KEYBOARD_FOCUS_IN pMsg->id	Pen position is entering the keyboard. The ID of this keyboard.
MT_KEYBOARD_FOCUS_OUT pMsg->id	Pen position is leaving the keyboard. The ID of this keyboard.

GetFilename

Synopsis

```
BOOL GetFilename(const char *pattern, const char *caption, char *filename, int size);
```

Description

Obtains the filename chosen by user. This function will pop up a container which displays the directories and files under the path **pattern*. When the button "OK" is clicked, it will back to the previous container and the file name will be copied to the memory pointer **filename* with length less than *size*.

**pattern* Specifies the pattern of file name.
**caption* Specifies the container title.
**filename* Specifies the file name (including path) being selected.
size Specifies the length of buffer **filename*.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

GetFilenameEx

Synopsis

```
BOOL GetFilenameEx(const char *pattern, const char *caption, char *filename, int size,  
WORD wflag);
```

Description

Obtains the filename chosen by user. This function will pop up a container which displays the directories and files under the path **pattern*. When the button "OK" is clicked, it will back to the previous container and the file name will be copied to the memory pointer **filename* with length less than *size*.

**pattern* Specifies the pattern of file name.
**caption* Specifies the container title.
**filename* Specifies the file name (including path) being selected.
size Specifies the length of buffer **filename*.
flag Specifies the setting. The following are possible values:

FMGR_FLAG_NORMAL	0x0000	Display (sub-)directories and files.
FMGR_FLAG_SHOW_DIR	0x0001	Display (sub-)directories only.
FMGR_FLAG_SHOW_CAPTION	0x0002	Display title.
FMGR_FLAG_ONLY_FILE	0x0004	Only files but not directories can be selected.
FMGR_FLAG_MULTI_SELECT	0x0008	More than one files can be selected.
FMGR_FLAG_FIELD_OBJ	0x0100	Use field object.
FMGR_FLAG_SMALL_PICTURE	0x0200	Use small icon.
FMGR_FLAG_USAGE_LEFT	0x4000	Tool bar is on left side.
FMGR_FLAG_3D_OBJ	0x8000	Use 3 D object.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

SearchComplete

Synopsis

```
void SearchComplete(SearchParam *params);
```

Description

Ends the global search (indicating that AP had tranversed all the records).

params Specifies the SearchParam structure.

Returned Value

None.

SearchDrawTitle

Synopsis

```
BOOL SearchDrawTitle(SearchParam *params, const char *title);
```

Description

Displays the title of the AP to be searched on the search screen.(normally the name of the AP).

params Specifies the SearchParam structure.
title Specifies The title to be displayed on the global search screen.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

SearchSaveMatched

Synopsis

```
BOOL SearchSaveMatched(SearchParam *params, DWORD recIdx, DWORD fldIdx, DWORD pos,  
                        DWORD custom, const char *str);
```

Description

Saves the matching record to the search structure and display it on the result screen.

params Specifies the SearchParam structure.
recIdx Specifies the record index of the string to be searched in AP.
fldIdx Specifies the field index of the string to be searched in AP.
pos Specifies the position of the string to be searched in AP.
custom Specifies the AP custom value.
str Specifies the information to be displayed on the result screen.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

SearchSetDefMode

Synopsis

```
int SearchSetDefMode(int mode, DWORD param);
```

Description

Sets the mode of search.

<i>mode</i>	Specifies the setting of search window. The following are possible values: SEARCH_MODE_NORMAL Activate the search windows without keyboard SEARCH_MODE_WITHKEYBOARD Activate the search windows with keyboard
<i>param</i>	Reserved parameter.

Returned Value

Return the latest mode.

SearchStrInStr

Synopsis

```
BOOL SearchStrInStr(const char *string, const char *charSet, DWORD *pos);
```

Description

Matches the pattern string StrcharSet in String, case insensitive.

<i>string</i>	Specifies the string to be searched.
<i>charSet</i>	Specifies the pattern string to be matched.
<i>pos</i>	Specifies the offset of the pattern in the string to be searched.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

SysGetDate

Synopsis

```
BOOL SysGetDate(int *year, int *month, int *day, BOOL isInit);
```

Description

Set the last error codes for UI objects.

<i>*year</i>	Specifies the memory buffer to which year will be copied from Setting-Date application.
<i>*month</i>	Specifies the memory buffer to which month will be copied from Setting-Date application.
<i>*day</i>	Specifies the memory buffer to which day will be copied from Setting-Date application.
<i>isInit</i>	Specifies the flag of initialization. If <i>isInit</i> is 1, the initial screen will be displayed but you have to set it up before the function is called. If <i>isInit</i> is 0, the current system date will be displayed.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

SysGetTime

Synopsis

```
BOOL SysGetTime(int *hour, int *minute, int *second, BOOL isDepent, BOOL isInit);
```

Description

Set the last error codes for UI objects.

<i>*hour</i>	Specifies the memory buffer to which hour will be copied from Setting-Time application.
<i>*minute</i>	Specifies the memory buffer to which minute will be copied from Setting-Time application.
<i>*second</i>	Specifies the memory buffer to which second will be copied from Setting-Time application.
<i>isDepent</i>	Specifies the dependent flag on hour. If this flag is set to TRUE, the digit of hour will be increased one when the digit of minute is increased from 59 to 00. TRUE Set to be dependent. FALSE Set to be independent.
<i>isInit</i>	Specifies the flag of initialization. If <i>isInit</i> is 1, the initial screen will be displayed but you have to set it up before the function is called. If <i>isInit</i> is 0, the current system date will be displayed.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

SysSearch

Synopsis

```
BOOL SysSearch(void);
```

Description

Activates global system global search.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

File and Database

In this chapter, some functions are introduced to access file and database.

File

General Description

Penbex OS supports file system. The file name must be less than 8 characters with extension name less than 3 characters. That is in “8.3” format. The file name will be truncated if longer than 8 characters. In this file system, all letters of file names, path names and drive names will be taken as capital letters even if you create or open them with small letters. The maximum number of opened file at the same time is 30. As for data management objects in Penbex OS, it stores data in files and this limitation has to be taken into consideration when you open database and files at the same time. This file system is compatible with FAT16 in PC.

Penbex OS provides a way like file retrieving to access memory directly. The file created via DR_NewP() is taken as general read-only file. It can be opened, read and closed via FileOpen(), FileRead() and FileClose(). Otherwise, user has to use DR_xx API's.

Related Data Structure

DSINFO Structure

This structure stores all information of a file.

```
typedef struct tagDSINFO {
    BYTE          fattribute; // File attribute. Some of the attributes can be combined by
                                // OR operation. The following are possible values:
                                // FILE_ATTR_NORMAL          Normal file
                                // FILE_ATTR_RDONLY          For reading only
                                // FILE_ATTR_HIDDEN          Hidden file
                                // FILE_ATTR_SYSTEM          System file
                                // FILE_ATTR_VOLUME          Reserved for long file name
                                // FILE_ATTR_DIRECTORY       Directory but not a file
                                // FILE_ATTR_ARCHIVE         Archived file

    BYTE          res; // Reserved
    const char    *fname; // File name, up to 8 characters
    const char    *fext; // Extension name, up to 3 characters
    const char    *fpath; // File path used in search pattern
    DWORD        fsize; // File size
    void         *p; // Reserved
}
```

FMTEASY Structure

```
typedef struct fmteasy {
    BYTE          bFatType; // FAT type. The following are possible values:
                                // 0x0c          FAT12
                                // 0x10          FAT16
                                // 0x20          Reserved for future FAT32

    BYTE          bSecpalloc; // Reserved for number of sector per cluster
    WORD          wNumroot; // Maximum of files on root path "A:\"
    char          text_volume_label[12]; // Disk label
}
```

System Message

No system message for file retrieving.

FileAttrib

Synopsis

```
BOOL FileAttrib(const char *fileName, BYTE *attr, BOOL isRead);
```

Description

Obtains or sets file attributes. Sets the input parameter *isRead* to TRUE for obtaining attributes or FALSE for setting attributes.

<i>*fileName</i>	Specifies the file name. The <i>fileName</i> can be a complete file name with drive and relative name with part of path or file name only, such as "A:A.TXT", "A:\A.TXT", "\DOWNLOAD\A.TXT" or "A.TXT".
<i>*attr</i>	Specifies the attribute to which the attribute is copied or set. The following are possible values: FILE_ATTR_NORMAL No specified attribute FILE_ATTR_RDONLY Read only file FILE_ATTR_HIDDEN Hidden file FILE_ATTR_SYSTEM System file FILE_ATTR_VOLUME Disk volume FILE_ATTR_CIRECTORY Directory but not a file FILE_ATTR_ARCHIVE Archived file
<i>*isRead</i>	Specifies the action of attribute. TRUE for obtaining attributes and FALSE for setting.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

FileClose

Synopsis

```
int FileClose(FHANDLE fd);
```

Description

Writes all unwritten data to the file and closes the file. This file descriptor is freed by FileClose() and the associated file cannot be operated afterward.

fd Specifies the descriptor of the file.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is -1.

FileDelete

Synopsis

```
BOOL FileDelete(const char *fileName);
```

Description

Deletes the file and releases all the memory it occupied. This function fails if the specified file does not exist or is not a file.

**fileName* Specifies the file name. The *fileName* can be a complete file name with drive and relative name with part of path or file name only, such as "A:A.TXT", "A:\A.TXT",

“\DOWNLOAD\A.TXT” or “A.TXT”.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

FileDeleteDir

Synopsis

```
BOOL FileDeleteDir(const char *dirName);
```

Description

Deletes the directory. This function fails if the specified directory does not exist or is not a directory.

**dirName* Specifies the directory name. The *dirName* can be a complete directory name with drive or relative name with part of path, such as “A:TEST”, “A:\TEST”, “\TEST” or “TEST”.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

FileDiskAbort

Synopsis

```
void FileDiskAbort(const char *diskName);
```

Description

Closes and then opens the disk drive.

**diskName* Specifies the name of the disk drive. This name can only be a character followed by a colon like “A:”.

Returned Value

None.

FileDiskClose

Synopsis

```
BOOL FileDiskClose(const char *diskName);
```

Description

Decreases the counter of opening this disk by one.

**diskName* Specifies the name of the disk drive. This name can only be a character followed by a colon like “A:”.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

FileDiskDefrag

Synopsis

```
BOOL FileDiskDefrag(const char *diskName, int mode);
```

Description

Defrags the disk drive. This function moves data in RAM disk per cluster and does not move subdirectory structure, locked files and system files.

**diskName* Specifies the name of the disk drive. This name can only be a character followed by a colon like "A:".
mode Specifies the mode of the disk defragmentation. The following are possible values:
DEFRAGMENT_FAST_MODE Reserved for future use.
DEFRAGMENT_MOST_MODE The only one mode so far.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

FileDiskFormat

Synopsis

```
BOOL FileDiskFormat(const char *diskName, FMTEASY *format);
```

Description

Formats the disk drive with specified parameters. All previous information of file system are erased, and the new file system is created. NAND gate memory cannot be formatted.

**diskName* Specifies the name of the disk drive. This name can only be a character followed by a colon like "A:".
**format* Specifies the FMTEASY pointer for formatting. The details of FMTEASY structure are as above.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

FileDiskOpen

Synopsis

```
int FileDiskOpen(const char *diskName);
```

Description

Increases the counter of opening this disk by one, and obtains the type of disk drive.

**diskName* Specifies the name of the disk drive. This name can only be a character followed by a colon like "A:".

Returned Value

The returned value is type of disk drive. The following are possible value returned by FileDiskOpen():
NO_DISK No such disk

RAM	Formatted RAM disk
RAM_UNFORMAT	Unformatted RAM disk
NAND	Formatted NAND data disk
NAND_UNFORMAT	Unformatted NAND data disk
SMC	Formatted smart media card
SMC_UNFORMAT	Unformatted smart media card
CF	Formatted compact flash memory
CF_UNFORMAT	Unformatted compact flash memory
SD	Formatted secure digital memory
SD_UNFORMAT	Unformatted secure digital memory
MS	Formatted memory stick
MS_UNFORMAT	Unformatted memory stick

FileFlush

Synopsis

```
BOOL FileFlush(FHANDLE fd);
```

Description

Writes all the unwritten data in memory buffer to the file and refreshes the FAT table of file system.

fd Specifies the file descriptor.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

FileGetCurrentDir

Synopsis

```
BOOL FileGetCurrentDir(const char *diskName, char *dirName);
```

Description

Gets current working or last worked directory name of a drive.

**diskName* Specifies the name of the disk drive. This name can only be a character followed by a colon like "A:".

**dirName* Specifies the memory buffer to which directory path name is copied. This path name starts and ends with back slashes like "\DOWNLOAD\123\".

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

FileGetDefaultDriver

Synopsis

```
short FileGetDefaultDriver(void);
```

Description

Gets the default disk drive. The default disk drive can be used when user opens, for example, a file only specifying a path name without disk name.

Returned Value

The returned value is the disk drive number. The following are possible values returned by

```
FileGetDefaultDriver():
```

0 Drive A

1 Drive B

2 Drive C

and so on.

FileGetDone

Synopsis

```
void FileGetDone(DSINFO *dsInfo);
```

Description

Releases system resource of the structure DSINFO created when calling FileGetFirst(). The input DSINFO structure is for system to release.

**dsInfo* Specifies the DSINFO structure to which the matched file is already copied.

Returned Value

None.

FileGetFirst

Synopsis

```
BOOL FileGetFirst(DSINFO *dsInfo, char *pattern);
```

Description

Gets the file information with file name matching indicated pattern. This function obtains matched files if any and moves the DSINFO structure to the first matched file.

**dsInfo* Specifies the DSINFO structure to which the first matched file is copied.

**pattern* Specifies the pattern string. This can be a wildcard pattern, such as "A:*.TXT", "\DOWNLOAD\A.???" or "*. *".

Returned Value

The returned value is TRUE if this function finds any matched file. Otherwise, the returned value is FALSE.

FileGetFreeSpace

Synopsis

```
long FileGetFreeSpace(char *diskName);
```

Description

Counts the number of free memory of the disk drive in bytes.

**diskName* Specifies the name of the disk drive. This name can only be a character followed by a

colon like "A:".

Returned Value

The returned value is the number of free memory in bytes if this function succeeds. Otherwise, the returned value is 0.

FileGetFreeSpaceEx

Synopsis

```
BOOL FileGetFreeSpaceEx(const char *diskName, WORD *sectPClust, WORD *bytePSect,  
                        DWORD *freeClust, DWORD *totalClust);
```

Description

Counts the number of free memory of the disk drive in bytes. To obtain the free memory in bytes, user has to multiply free cluster *FreeClust* by sectors per cluster *SectPClust* and bytes per sector *BytePSect*.

- *diskName* Specifies the name of the disk drive. This name can only be a character followed by a colon like "A:".
- *sectPClust* Specifies the number to which sectors per cluster of current system architecture is copied. You can skip this parameter by inputting NULL if you want to.
- *bytePSect* Specifies the number to which bytes per sector of current system architecture is copied. You can skip this parameter by inputting NULL if you want to.
- *freeClust* Specifies the number to which free cluster is copied. You can skip this parameter by inputting NULL if you want to.
- *totalClust* Specifies the number to which total cluster is copied. You can skip this parameter by inputting NULL if you want to.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

FileGetNext

Synopsis

```
BOOL FileGetNext(DSINFO *dsInfo);
```

Description

Moves the DSINFO structure to the next file matching the pattern of FileGetFirst().

- *dsInfo* Specifies the DSINFO structure to which the matched file is already copied.

Returned Value

The returned value is TRUE if there is at least one matched file. Otherwise, the returned value is FALSE.

FileGetTime

Synopsis

```
BOOL FileGetTime(const char *fileName, RtcDateTime *time);
```

Description

Obtains the date and time of a file.

- *fileName* Specifies the file name. This file must be open under A drive, so the file name need not and must not be composed of drive name, such as "A.TXT", "\A.TXT" or "\DOWNLOAD\A.TXT".
- *time* Specifies the RtcDateTime structure of the new time. The time of the file will be set to current time if this input is NULL.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

FileNewDir

Synopsis

```
BOOL FileNewDir(const char *dirName);
```

Description

Creates a new directory in the path specified. This function fails if the specified path does not exist.

- *dirName* Specifies the directory name. The *dirName* can be a complete directory name with drive or relative name with part of path, such as "A:TEST", "A:\TEST", "\TEST" or "TEST".

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

FileOpen

Synopsis

```
int FileOpen(const char *fileName, WORD flag, WORD mode);
```

Description

Opens the file with a flag. If this flag is composed of FILE_FLAG_CREAT, it will use mode parameter to set the access permission on that file.

- *fileName* Specifies the file name. The *fileName* can be a complete file name with drive and relative name with part of path or file name only, such as "A:A.TXT", "A:\A.TXT", "\DOWNLOAD\A.TXT" or "A.TXT".
- flag* Specifies the flag to open file. Some of these flags can be combined by using OR (|) operation. For example, if you want to open a file for editing no matter this file exists or not, you have to include the flag FILE_FLAG_CREAT or it will fail if this file does not exist. The following are possible values of flag:
- | | |
|---------------------|---|
| FILE_FLAG_READONLY | Open a file for reading. |
| FILE_FLAG_READWRITE | Open a file for writing. If this file exists, the contents will be overwritten. |
| FILE_FLAG_APPEND | Open a file for writing at the end of this file. |
| FILE_FLAG_CREAT | Create a file if it does not exist. |
| FILE_FLAG_TRUNCATE | Open a file for writing. If this file exists, the contents will be destroyed. |
| FILE_FLAG_EXIST | Terminate the opening if this file already exists. |

	FILE_NOSHAREWRITE	Reserve for future use only. This flag allows multiple opens on the same file at the same time, but at most one of these opens is open for writing.
	FILE_NOSHAREANY	Reserve for future use only. This flag allows only one open on the same file at the same time.
<i>mode</i>	Specifies the permission mode for creating file.	
	FILE_FLAG_READ	The new file will be created for reading only. This file can be edited only when the first time it was created.
	FILE_FLAG_READ_WRITE	The new file will be created for reading and writing.

Returned Value

The returned value is a non-negative integer if this function succeeds. Otherwise, the returned value is -1.

FileRead

Synopsis

```
short FileRead(FHANDLE fd, BYTE *buffer, short count);
```

Description

Reads up to *count* bytes from the input file *fd* and store them in the given buffer. If the file position is at the end of the file, it reads nothing.

<i>fd</i>	Specifies the file descriptor.
* <i>buffer</i>	Specifies the memory buffer to which data is copied.
<i>count</i>	Specifies the maximum bytes to read.

Returned Value

The returned value is the same with the input parameter *count* if this function succeeds. Otherwise, the returned value is -1.

FileReadEx

Synopsis

```
long FileReadEx(FHANDLE fd, BYTE *buffer, long count);
```

Description

Reads up to *count* bytes from the input file *fd* and store them in the given buffer *buffer*. The only difference between FileReadEx() and FileRead() is the number of bytes it can read. FileRead() can only read a short integer number of bytes but FileReadEx() can read a long integer number.

<i>fd</i>	Specifies the file descriptor.
* <i>buffer</i>	Specifies the memory buffer to which data is copied.
<i>count</i>	Specifies the maximum bytes to read.

Returned Value

The returned value is the same with the input parameter *count* if this function succeeds. Otherwise, the returned value is -1.

FileRename

Synopsis

```
BOOL FileRename(const char *oldName, const char *newName);
```

Description

Changes the file name to a new one.

- *oldName* Specifies the file name. The *fileName* can be a complete file name with drive and path, relative name with part of path or file name only, such as "A:A.TXT", "A:\A.TXT", "\DOWNLOAD\A.TXT" or "A.TXT".
- *newName* Specifies the new file name. This new name must be in "8.3" format without any drive or path name, such as "B.TXT".

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

FileRewind

Synopsis

```
void FileRewind(FHANDLE fd);
```

Description

Sets or changes file position to the beginning.

fd Specifies the file descriptor.

Returned Value

None.

FileSeek

Synopsis

```
long FileSeek(FHANDLE fd, long offset, short origin);
```

Description

Sets or changes the position of the file to a new one.

fd Specifies the file descriptor.

offset Specifies the new position from an indicated base position.

origin Specifies the base position. The following are possible values:

SEEK_FROM_BEGIN	Beginning of file.
SEEK_FROM_CURRENT	Current position of file.
SEEK_FROM_END	End of file.

Returned Value

The returned value is the new position if this function succeeds. Otherwise, the returned value is -1.

FileSetCurrentDir

Synopsis

```
BOOL FileSetCurrentDir(const char *dirName);
```

Description

Sets or changes the current working directory to a new one for current drive. This function will not effect the current (default) drive.

**dirName* Specifies the directory name. The *dirName* can be a complete directory name with drive or relative name with part of path, such as "A:TEST", "A:\TEST", "\TEST" or "TEST".

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

FileSetDefaultDriver

Synopsis

```
BOOL FileSetDefaultDriver(short driverNo);
```

Description

Sets or changes the current disk driver to a new one.

driverNo Specifies the disk drive number. The following are possible values of disk drive number:

0	Drive A
1	Drive B
2	Drive C

and so on.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

FileSetTime

Synopsis

```
BOOL FileSetTime(const char *fileName, RtcDateTime *time);
```

Description

Modifies the date and time of a file.

**fileName* Specifies the file name. The *fileName* can be a complete file name with drive and relative name with part of path or file name only, such as "A:A.TXT", "A:\A.TXT", "\DOWNLOAD\A.TXT" or "A.TXT".

**time* Specifies the RtcDateTime structure of the new time. The time of the file will be set to current time if this input is NULL.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

FileTell

Synopsis

```
long FileTell(FHANDLE fd);
```

Description

Obtains the file position.

fd Specifies the file descriptor.

Returned Value

The returned value is the file position if this function succeeds. Otherwise, the returned value is -1.

FileTruncate

Synopsis

```
BOOL FileTruncate(FHANDLE fd, long offset);
```

Description

Truncates the file to the indicated length *offset*. This file must be open for truncating. If the file size exceeds *offset*, the extra data will be discarded. If the file size is smaller than *offset*, this function will fail.

fd Specifies the file descriptor.
offset Specifies the length of content in bytes you want to keep.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

FileWrite

Synopsis

```
short FileWrite(FHANDLE fd, BYTE *buffer, short count);
```

Description

Writes up to *count* bytes from the given buffer to the file *fd*. This function does nothing if error occurs or out of memory.

fd Specifies the file descriptor.
**buffer* Specifies the memory buffer from which data is copied.
count Specifies the maximum bytes to write.

Returned Value

The returned value is the same with the input parameter *count* if this function succeeds. Otherwise, the returned value is -1.

FileWriteEx

Synopsis

```
long FileWriteEx(FHANDLE fd, BYTE *buffer, long count);
```

Description

Writes up to *count* bytes from the given buffer to the file *fd*. The only difference between FileWriteEx() and FileWrite() is the number of bytes it can write. FileWrite() can only write a short integer number of bytes but FileWriteEx() can write a long integer number.

<i>fd</i>	Specifies the file descriptor.
* <i>buffer</i>	Specifies the memory buffer from which data is copied.
<i>count</i>	Specifies the maximum bytes to write.

Returned Value

The returned value is the same with the input parameter *count* if this function succeeds. Otherwise, the returned value is -1.

GetDiskVolumeInfo

Synopsis

```
BOOL GetDiskVolumeInfo(const char *diskName, char *volumeBuf, WORD vBufLen,  
                        DWORD *volumeSerialNumber, DWORD *maxNameLen,  
                        char *fatNameBuf, WORD fBufLen);
```

Description

Obtains the information of disk, including disk volume name, system number, maximum length of file name and type of file system.

* <i>diskName</i>	Specifies the name of the disk drive. This name can only be a character followed by a colon like "A:".
* <i>volumeBuf</i>	Specifies the buffer to which disk volume is copied.
<i>vBufLen</i>	Specifies the string length of returned disk volume.
* <i>volumeSerialNumber</i>	Specifies the buffer to which system serial number is copied.
* <i>maxNameLen</i>	Specifies the buffer to which the maximum length of file name is copied.
* <i>fatNameBuf</i>	Specifies the buffer to which type of file system is copied, such as "FAT12".
<i>fBufLen</i>	Specifies the string length of returned disk type.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

GetFileLastError

Synopsis

```
int GetFileLastError(void);
```

Description

Obtains the returned status of the last file library being called. Penbex OS stores the returned status for all functions about file, and user can obtain this status easily by calling this function. Some of the file functions returns zero no matter those functions succeed or not. For this condition,

user has to call this function to make sure those functions succeed or not.

Returned Value

The returned value is the returned status of the last file library being called. The following are possible values:

FERR_NO_ERROR	The function succeeds.
FERR_FILE_NOT_FOUND	Cannot find the indicated file.
FERR_INVALID_FILE_HANDLE	The file associated with the indicated handle does not exist.
FERR_DIR_RDONLY_CANT_OPEN	
FERR_FILE_ALREADY_EXISTS	The indicated file already exists.
FERR_SEEK_NEGATIVE_VALUE	The input parameter offset cannot be negative.
FERR_TOO_MANY_FILES_OPEND	The number of open files exceeds the system limitation.
FERR_WRITE_FAILED	Cannot write data to file.
FERR_TRUNCATE_ON_SHARING	

NumberOfDir

Synopsis

```
int NumberOfDir(void);
```

Description

(This function is reserved for future implementation.)

NumberOfFile

Synopsis

```
int NumberOfFile(void);
```

Description

(This function is reserved for future implementation.)

SetDiskVolumeLabel

Synopsis

```
BOOL SetDiskVolumeLabel(const char *diskName, const char *label);
```

Description

Modifies the disk label. The label is composed of 12 characters at most including null terminated.

<i>*diskName</i>	Specifies the name of the disk drive. This name can only be a character followed by a colon like "A:".
<i>*label</i>	Specifies the disk label to set to disk.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

DR_DeLP

Synopsis

```
BOOL DR_DelP(const char *fileName);
```

Description

Removes a file created via DR_NewP().

**fileName* Specifies the file name. This file must be open under A drive, so the file name need not and must not be composed of drive name, such as "A.TXT", "\A.TXT" or "\DOWNLOAD\A.TXT".

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

DR_GetP

Synopsis

```
void *DR_GetP(const char *fileName);
```

Description

Obtains the file buffer created via DR_NewP().

**fileName* Specifies the file name. This file must be open under A drive, so the file name need not and must not be composed of drive name, such as "A.TXT", "\A.TXT" or "\DOWNLOAD\A.TXT".

Returned Value

The returned value is the memory buffer if this function succeeds. Otherwise, the returned value is NULL.

DR_LockP

Synopsis

```
BOOL DR_LockP(const char *fileName, BOOL isLock);
```

Description

Locks or unlocks a file created via DR_NewP(). A locked file cannot be moved, including enlarged or reduced via DR_ResizeP() or defragged via FileDiskDefrag(). All the files created via DR_NewP() are locked by default.

**fileName* Specifies the file name. This file must be open under A drive, so the file name need not and must not be composed of drive name, such as "A.TXT", "\A.TXT" or "\DOWNLOAD\A.TXT".

isLock Specifies the status of locking. TRUE for locking and FALSE for unlocking.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

DR_MaxBlock

Synopsis

```
DWORD DR_MaxBlock(void);
```

Description

Obtains the available consecutive memory size in bytes for creating a file via DR_NewP().

Returned Value

The returned value is the memory size in bytes if this function succeeds. Otherwise, the returned value is 0.

DR_NewP

Synopsis

```
void *DR_NewP(const char *fileName, DWORD fileSize);
```

Description

Allocates a consecutive memory buffer of size *FileSize*. This function fails if the file already existed and created via DR_NewP().

**fileName* Specifies the file name. This file must be open under A drive, so the file name need not and must not be composed of drive name, such as "A.TXT", "\A.TXT" or "\DOWNLOAD\A.TXT".

fileSize Specifies the size of the memory buffer.

Returned Value

The returned value is the memory buffer if this function succeeds. Otherwise, the returned value is NULL.

DR_RenameP

Synopsis

```
BOOL DR_RenameP(const char *oldName, const char *newName);
```

Description

Changes the file name created via DR_NewP() to a new one.

**oldName* Specifies the file name. This file must be open under A drive, so the file name need not and must not be composed of drive name, such as "A.TXT", "\A.TXT" or "\DOWNLOAD\A.TXT".

**newName* Specifies the file name. This file must be open under A drive, so the file name need not and must not be composed of drive name, such as "A.TXT", "\A.TXT" or "\DOWNLOAD\A.TXT".

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

DR_ResizeP

Synopsis

```
void *DR_ResizeP(const char *fileName, DWORD fileSize);
```

Description

Re-allocates a consecutive memory buffer of size *FileSize* for an existing file created via DR_NewP(). This function fails if the file does not exist or is not created via DR_NewP().

**fileName* Specifies the existing file name. This file must be open under A drive, so the file name need not and must not be composed of drive name, such as "A.TXT", "\A.TXT" or "\DOWNLOAD\A.TXT".

fileSize Specifies the new size of the memory buffer.

Returned Value

The returned value is the memory buffer if this function succeeds. Otherwise, the returned value is NULL.

DR_SizeP

Synopsis

```
DWORD DR_SizeP(const char *fileName);
```

Description

Obtains the memory size in bytes for an existing file created via DR_NewP(). This function fails if the file does not exist or is not created via DR_NewP().

**fileName* Specifies the existing file name. This file must be open under A drive, so the file name need not and must not be composed of drive name, such as "A.TXT", "\A.TXT" or "\DOWNLOAD\A.TXT".

Returned Value

The returned value is the memory size in bytes if this function succeeds. Otherwise, the returned value is 0.

Database

General Description

Penbex OS supports 2 database systems. One is database and the other is data manager. As for database manager, all data are stored in files with extension name “DBF”, in dBase III compatible format.

Database provides only character data type. Every data manager has a unique handle which is a positive number greater than 0 to identify itself. The field name cannot consist of double-byte word such Chinese character. When using database, the following limitations have to be taken into consideration.

65535	Maximal size of data per field
28	Maximal length of field name
64	Maximal number of fields per database
3	Maximal number of fields of an index
64	Maximal opened index files at the same time
65535	Maximal number of records per database

The following are possible returned values for some of the database API' s:

DBERR_SUCCESS	Command succeeds.
DBERR_INVALID_PARAMETER	Input the wrong or invalid parameter(s).
DBERR_OUT_OF_RANGE	The input position exceeds the total number of records
DBERR_DELETED_RECORD	The record is already deleted.
DBERR_NO_MEMORY	Out of memory.
DBERR_MAX_RECORD	The number of records in a database exceeds the limitation.
DBERR_CANNOT_OPEN	Cannot open database.
DMERR_ERROR	Unknown error.

Related Data Structure

DBField (also *PdbField) Structure

This structure stores all information of a field of database.

```
#define FIELD_NAME_SIZE    10
typedef struct tag_DbField {
    char cDbFieldNameA[FIELD_NAME_SIZE + 1];
                                // Field name, not case sensitive and cannot be Chinese characters
    char cDbFieldType;         // Field type
    BYTE bDbFieldLength;       // Field length
    BYTE bDbDecimalLength;     // Length of digits for decimal
}
```

System Message

No system message for database retrieving.

DbAppend

Synopsis

```
int DbAppend(DWORD handle);
```

Description

Allocates empty buffer of a record after the last record. DbWrite() has to be called to write the data into that buffer. The record position moves to the appended record after calling DbAppend().

handle Specifies the handle of the database.

Returned Value

The returned value is the position of appended record, which is a non-negative value, if this function succeeds. Otherwise, the returned value is a negative value.

DbCloseDb

Synopsis

```
int DbCloseDb(DWORD handle);
```

Description

Closes a database. Database will automatically close when the application calling this database terminates.

handle Specifies the handle of the database.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value.

DbCreateDb

Synopsis

```
BOOL DbCreateDb(const char *dbName, PDbField dbFld, WORD fldCnt);
```

Description

Creates a new database.

<i>*dbName</i>	Specifies the complete file name of the database with extension name "DBF".
<i>dbFld</i>	Specifies the field structure array of the database.
<i>fldCnt</i>	Specifies the number of fields.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

DbCreateNdx

Synopsis

```
BOOL DbCreateNdx(const char *dbName, const char *idxName, WORD fldNo);
```

Description

Creates an index file for the specified field of database. Database need not to open when calling this function.

**dbName* Specifies the complete file name of the database with extension name "DBF".
**idxName* Specifies the file name of index.
fldNo Specifies the field number to be index.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

DbDelete

Synopsis

```
int DbDelete(DWORD handle);
```

Description

Deletes the current record.

handle Specifies the handle of the database.

Returned Value

The returned value is the number of records if this function succeeds. Otherwise, the returned value is a negative value.

DbDeleteAndPack

Synopsis

```
BOOL DbDeleteAndPack(DWORD handle, WORD ptype);
```

Description

Deletes the current record and executes DbPack() after deleting.

handle Specifies the handle of the database.
ptype Specifies the way to pack. The only valid value is 0 so far.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

DbDeleteDb

Synopsis

```
int DbDeleteDb(const char *dbName);
```

Description

Deletes a database. Database has to be closed before being deleted.

**dbName* Specifies the complete file name of the database with extension name "DBF".

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value.

DbDeleteNdx

Synopsis

```
int dbDeleteNdx(const char *idxName);
```

Description

Deletes the index file. Index file has to be closed before being deleted.

**idxName* Specifies the file name of index.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value.

DbGotoBottom

Synopsis

```
long DbGotoBottom(DWORD handle);
```

Description

Moves the record position of database to the last record with all records ordered by the index. Index file has to be opened by DbOpenDbWithNdx() or DbOpenDbWithMulNdx() before calling this function. The record position starts from 0; that is, the position of the first record is 0.

handle Specifies the handle of the database.

Returned Value

The returned value is record position, which is a non-negative value, if this function succeeds. Otherwise, the returned value is a negative value.

DbGotoTop

Synopsis

```
long DbGotoTop(DWORD handle);
```

Description

Moves the record position of database to the first record with all records ordered by the index. Index file has to be opened by DbOpenDbWithNdx() or DbOpenDbWithMulNdx() before calling this function. The record position starts from 0; that is, the position of the first record is 0.

handle Specifies the handle of the database.

Returned Value

The returned value is record position, which is a non-negative value, if this function succeeds. Otherwise, the returned value is a negative value.

DbInitial

Synopsis

```
void DbInitial(void);
```

Description

Allocates memory for database. User has to initialize database before open it.

Returned Value

None.

DbIsDeleted

Synopsis

```
BOOL DbIsDeleted(DWORD handle);
```

Description

Checks if the current record is deleted or not.

handle Specifies the handle of the database.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

DbNameOfField

Synopsis

```
BOOL DbNameOfField(DWORD handle, WORD fldNo, const char *fldName);
```

Description

Obtains the name of the specified field.

handle Specifies the handle of the database.

fldNo Specifies the field number.

**fldName* Specifies the memory buffer to which field name will be copied.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

DbNext

Synopsis

```
long DbNext(DWORD handle);
```

Description

Moves the record position of database to the next record with all records ordered by the index. Index file has to be opened by DbOpenDbWithNdx() or DbOpenDbWithMulNdx() before calling

this function. The record position starts from 0; that is, the position of the first record is 0.

handle Specifies the handle of the database.

Returned Value

The returned value is record position, which is a non-negative value, if this function succeeds. Otherwise, the returned value is a negative value.

DbNumberOfField

Synopsis

```
WORD DbNumberOfField(DWORD handle);
```

Description

Obtains the number of fields per record.

handle Specifies the handle of the database.

Returned Value

The returned value is the number of fields if this function succeeds. Otherwise, the returned value is 0.

DbNumberOfRecord

Synopsis

```
WORD DbNumberOfRecord(DWORD handle);
```

Description

Obtains the number of records in database.

handle Specifies the handle of the database.

Returned Value

The returned value is the number of records if this function succeeds. Otherwise, the returned value is 0.

DbNumberOfRecordByKey

Synopsis

```
WORD DbNumberOfRecordByKey(DWORD handle, const char *key, long keyLen);
```

Description

Obtains the number of record with the index field composed of the specified data. Index file has to be opened by DbOpenDbWithNdx() or DbOpenDbWithMulNdx() before calling this function.

handle Specifies the handle of the database.

**key* Specifies the data to search for. It can be a pointer to character or integer.

keyLen Specifies the length of the specified key in bytes.

Returned Value

The returned value is the number of records if this function succeeds. Otherwise, the returned value is 0.

DbOffsetBykey

Synopsis

```
WORD DbOffsetBykey(DWORD handle, long recNo, const char *key, long keyLen);
```

Description

Obtains the ranking of *recNo*th record in the records with the index field composed of the specified data. Index file has to be opened by DbOpenDbWithNdx() or DbOpenDbWithMulNdx() before calling this function.

<i>handle</i>	Specifies the handle of the database.
<i>recNo</i>	Specifies the record number.
* <i>key</i>	Specifies the data to search for. It can be a pointer to character or integer.
<i>keyLen</i>	Specifies the length of the specified key in bytes.

Returned Value

The returned value is the record position, which is a non-negative value, if this function succeeds. Otherwise, the returned value is 0.

DbOpenDb

Synopsis

```
DWORD DbOpenDb(const char *dbName);
```

Description

Opens a database.

* <i>dbName</i>	Specifies the complete file name of the database with extension name "DBF".
-----------------	---

Returned Value

The returned value is database handle if this function succeeds. Otherwise, the returned value is 0.

DbOpenDbWithMulNdx

Synopsis

```
DWORD DbOpenDbWithMulNdx(const char *dbName, const char **idxName, WORD idxCnt);
```

Description

Opens database and the associated index files. This function can open more than one index file.

* <i>dbName</i>	Specifies the complete file name of the database with extension name "DBF".
** <i>idxName</i>	Specifies the array of the index file names.
<i>idxCnt</i>	Specifies the number of the index files.

Returned Value

The returned value is a handle of database if this function succeeds. Otherwise, the returned value is 0.

DbOpenDbWithNdx

Synopsis

```
DWORD DbOpenDbWithNdx(const char *dbName, const char *idxName);
```

Description

Opens database and the associated index file. This function can open only one index file.

**dbName* Specifies the complete file name of the database with extension name "DBF".
**idxName* Specifies the index file name.

Returned Value

The returned value is a handle of database if this function succeeds. Otherwise, the returned value is 0.

DbPack

Synopsis

```
BOOL DbPack(DWORD handle);
```

Description

Compacts a database.

handle Specifies the handle of the database.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

DbPackEx

Synopsis

```
BOOL DbPackEx(DWORD handle);
```

Description

Compacts a database.

handle Specifies the handle of the database.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

DbPrev

Synopsis

```
long DbPrev(DWORD handle) ;
```

Description

Moves the record position of database to the previous record with all records ordered by the index. Index file has to be opened by DbOpenDbWithNdx() or DbOpenDbWithMulNdx() before calling this function. The record position starts from 0; that is, the position of the first record is 0.

handle Specifies the handle of the database.

Returned Value

The returned value is the record number, which is a non-negative value, if this function succeeds. Otherwise, the returned value is a negative value.

DbRead

Synopsis

```
int DbRead(DWORD handle, WORD fldNo, char *buffer, WORD *bufSize) ;
```

Description

Reads data from specified field of current record.

handle Specifies the handle of the database.
fldNo Specifies the field number.
**buffer* Specifies the buffer to which data is copied.
**bufSize* Specifies the length of data. This parameter has to be initialized to the length of memory buffer, and the length of data being copied after calling this function will be assigned to it.

Returned Value

The returned value is the record number of the record being read if this function succeeds. Otherwise, the returned value is a negative value.

DbRelease

Synopsis

```
void DbRelease(void) ;
```

Description

Releases resources hold by a database. Database will be automatically released when the application calling database terminates.

Returned Value

None.

DbSearch

Synopsis

```
long DbSearch(DWORD handle, const char *key, long keyLen) ;
```

Description

Obtains the record number of the next record from current position with the index composed of the key and all records ordered by the index. This function only obtains the record number but not moves to that record. Index file has to be opened by DbOpenDbWithNdx() or DbOpenDbWithMulNdx() before calling this function.

<i>handle</i>	Specifies the handle of the database.
<i>*key</i>	Specifies the data to search for. It can be a pointer to character or integer.
<i>keyLen</i>	Specifies the length of the specified key in bytes.

Returned Value

The returned value is the record number, which is a non-negative value, if this function succeeds. Otherwise, the returned value is a negative value.

DbSearchFirst

Synopsis

```
long DbSearchFirst(DWORD handle, const char *key, long keyLen);
```

Description

Obtains the record number of the first record with the index composed of the key and all records ordered by the index. This function only obtains the record number but not moves to that record. Index file has to be opened by DbOpenDbWithNdx() or DbOpenDbWithMulNdx() before calling this function.

<i>handle</i>	Specifies the handle of the database.
<i>*key</i>	Specifies the data to search for. It can be a pointer to character or integer.
<i>keyLen</i>	Specifies the length of the specified key in bytes.

Returned Value

The returned value is the record number, which is a non-negative value, if this function succeeds. Otherwise, the returned value is a negative value.

DbSearchLast

Synopsis

```
long DbSearchLast(DWORD handle, const char *key, long keyLen);
```

Description

Obtains the record number of the last record with the index composed of the key and all records ordered by the index. This function only obtains the record number but not moves to that record. Index file has to be opened by DbOpenDbWithNdx() or DbOpenDbWithMulNdx() before calling this function.

<i>handle</i>	Specifies the handle of the database.
<i>*key</i>	Specifies the data to search for. It can be a pointer to character or integer.
<i>keyLen</i>	Specifies the length of the specified key in bytes.

Returned Value

The returned value is the record number, which is a non-negative value, if this function succeeds.

Otherwise, the returned value is a negative value.

DbSeek

Synopsis

```
long DbSeek(DWORD handle, long offset, BYTE origin);
```

Description

Moves the record position to the specified position. To seek backward from the end of the database, offset must be a negative value and the base point be DBSEEK_END.

<i>handle</i>	Specifies the handle of the database.
<i>offset</i>	Specifies the offset value from a base point.
<i>origin</i>	Specifies the base point of seek cursor. DBSEEK_CUR Seek from current record DBSEEK_SET Seek from the first record DBSEEK_END Seek from the last record

Returned Value

The returned value is the record number, which is a non-negative value, if this function succeeds. Otherwise, the returned value is a negative value.

DbSeekBykey

Synopsis

```
long DbSeekbBykey(DWORD handle, long offset, BYTE origin, const char *key, long keyLen);
```

Description

Move the database pointer to the specific position.

<i>handle</i>	Specifies the handle of the database.
<i>offset</i>	Specifies the offset value from a base point.
<i>origin</i>	Specifies the base point of seek cursor. DBSEEK_CUR Seek from current record DBSEEK_SET Seek from the first record DBSEEK_END Seek from the last record
* <i>key</i>	Specifies the data to search for. It can be a pointer to character or integer.
<i>keyLen</i>	Specifies the length of the specified key in bytes.

Returned Value

The returned value is the record number, which is a non-negative value, if this function succeeds. Otherwise, the returned value is a negative value.

DbSetMasterNdx

Synopsis

```
long DbSetMasterNdx(DWORD handle, WORD fileIdx);
```

Description

Changes the index from some index files. If database is opened via DbOpenDbWithMulNdx(), the

index can be switched by this function. For example, there is an array of index files in DbOpenDbWithMulNdx(), says idx[0], idx[1] and idx[2]. User can reorder records by changing *fileIdx* to 0, 1 or 2. The default index depends on the first element of index file array, says idx[0]. Index file has to be opened by DbOpenDbWithNdx() or DbOpenDbWithMulNdx() before calling this function.

<i>handle</i>	Specifies the handle of the database.
<i>fileIdx</i>	Specifies the ranking of the designated index file in the index file array of DbOpenDbWithMulNdx().

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value.

DbSizeOfField

Synopsis

```
WORD DbSizeOfField(DWORD handle, WORD fldNo);
```

Description

Obtains the size of the field. The size is determined when database is created.

<i>handle</i>	Specifies the handle of the database.
<i>fldNo</i>	Specifies the field number.

Returned Value

The returned value is the field size if this function succeeds. Otherwise, the returned value is 0.

DbSizeOfRecord

Synopsis

```
WORD DbSizeOfRecord(DWORD handle);
```

Description

Obtains the size of current record.

<i>handle</i>	Specifies the handle of the database.
---------------	---------------------------------------

Returned Value

The returned value is the size of current record if this function succeeds. Otherwise, the returned value is a negative value.

DbUnDelete

Synopsis

```
int DbUnDelete(DWORD handle);
```

Description

Undeletes the last deleted record.

handle Specifies the handle of the database.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value.

DbWrite

Synopsis

```
int DbWrite(DWORD handle, WORD fldNo, const char *buffer, WORD *bufSize);
```

Description

Writes data to specified field at current record.

<i>handle</i>	Specifies the handle of the database.
<i>fldNo</i>	Specifies the field number.
* <i>buffer</i>	Specifies the data written to field.
* <i>bufSize</i>	Specifies the length of data written to field.

Returned Value

The returned value is the record number of current one if this function succeeds. Otherwise, the returned value is a negative value.

Data Manager

General Description

Penbex OS supports 2 database systems. One is database and the other is data manager. As for data manager, all data are stored in files with extension name "DM", in Penbex proprietary format. In this format, every field is stored in a unit of 2 bytes. For example, there is a record with 3 fields, and the associated data are as "AA", "BBB" and "CCCC". This record is stored in memory like:

2	3	5	A	A	E	E	E		C	C	C	C	C	
---	---	---	---	---	---	---	---	--	---	---	---	---	---	--

Data length of each field is sequentially stored at the beginning, called length table. The data type of length is WORD (4 bytes). 2, 3 and 5 in the above diagram are the data length of the first, second and third field in bytes. Data are stored following data length area in memory. The second and the third field are followed by a byte because data are stored in a unit of 2 bytes as specified above.

Data manager provides data type more than character, such as integer. Each data manager has a unique handle which is a positive number greater than 0 to identify itself. The field name cannot consist of double-byte word such Chinese character. When using data manager, the following limitations have to be taken into consideration.

DM_FIELD_MAX_LENGTH	65535	Maximal size of data per field
DM_FIELD_NAME_SIZE	28	Maximal length of field name
DM_MAX_DBOPENED	8	Maximal opened data manager at the same time
DM_MAX_FIELD	64	Maximal number of fields per data manager
DM_MAX_NDX_FIELD	3	Maximal number of fields of an index
DM_MAX_NDXOPENED	64	Maximal opened index files at the same time
DM_MAX_RECORD	65535	Maximal number of records per data manager
DM_RECORD_MAX_SIZE	DM_FIELD_MAX_LENGTH * DM_MAX_FIELD	Maximal size of data per record

The following are possible returned values for some of the data manager API' s:

DMERR_SUCCESS	Command succeeds.
DMERR_INVALID_PARAMETER	Input the wrong or invalid parameter(s).
DMERR_OUT_OF_RANGE	The input position exceeds the total number of records
DMERR_NO_RECORD	No record in data manager.
DMERR_NO_MEMORY	Out of memory.
DMERR_MAX_RECORD	The number of record in a data manager exceeds the limitation DM_MAX_RECORD.
DMERR_MAX_DBOPENED	The number of opened data manager exceeds the limitation DM_MAX_DBOPENED.
DMERR_DB_CANNOT_OPEN	Cannot open data manager.
DMERR_DB_CANNOT_READ	Cannot read data from data manager.
DMERR_DB_CANNOT_CLOSE	Cannot close data manager.
DMERR_DB_CANNOT_WRITE	Cannot write data to data manager.
DMERR_INDEX_CANNOT_OPEN	Cannot open index file.
DMERR_INDEX_CANNOT_READ	Cannot read data from index file.
DMERR_INDEX_CANNOT_CLOSE	Cannot close index file.
DMERR_INDEX_CANNOT_WRITE	Cannot write data to index file.
DMERR_INDEX_DIRTY	Index is not consistent with data. Index file has to be deleted and created again when this error occurs.
DMERR_NOT_FOUND	Cannot find any record.
DMERR_DISK_NO_SPACE	Out of disk space.
DMERR_TEMP_FILE_CANNOT_OPEN	Cannot open temporary file.
DMERR_NOT_INITIALIZE	Data manager is not initialized.
DMERR_INVALID_FIELD	Input field is not valid.

DMERR_DATA_TOO_LONG	Data length is more than the limitation.
DMERR_BUFFER_NOT_ENOUGH	Out of buffer.
DMERR_BOTTOM	Record pointer is at the bottom.
DMERR_TOP	Record pointer is on the top.
DMERR_FILE_EXISTS	File already exists.

Related Data Structure

DMField Structure

This structure stores all information of a field of data manger.

```
#define DM_FIELD_NAME_SIZE 28
typedef struct tag_DMField {
    char cfldNameA[DM_FIELD_NAME_SIZE + 1];
    // Field name, not case sensitive and cannot be Chinese characters
    BYTE bFieldType;
    // Field type
    // DM_FIELD_TYPE_USHORT   Unsigned short integer
    // DM_FIELD_TYPE_SHORT    Short integer
    // DM_FIELD_TYPE_ULONG    Unsigned long integer
    // DM_FIELD_TYPE_LONG     Long integer
    // DM_FIELD_RESERVED1     Reserved for future use
    // DM_FIELD_TYPE_CHAR     Character
    // DM_FIELD_TYPE_BINARY   Binary
    // DM_FIELD_TYPE_VARCHAR  DM_FIELD_TYPE_CHAR
    // DM_FIELD_TYPE_WORD     DM_FIELD_TYPE_USHORT
    // DM_FIELD_TYPE_DWORD    DM_FIELD_TYPE_ULONG
    WORD wFieldLength;
    // Field length
}
```

System Message

No system message for data manager retrieving.

DMAppend

Synopsis

```
long DMAAppend(DWORD handle);
```

Description

Allocates empty buffer of a record after the last record. To complete appending a record, DMWrite() has to be called to write the data into that buffer, and followed by DMFlush() to update the related files. The record position moves to the appended record after calling DMAAppend().

handle Specifies the handle of the data manager.

Returned Value

The returned value is the position of appended record, which is a non-negative value, if this function succeeds. Otherwise, the returned value is a negative value, which can be DMERR_INVALID_PARAMETER or DMERR_MAX_RECORD.

DMCreateNDX

Synopsis

```
long DMCreateNDX(const char *dbName, const char *idxName, WORD fldNo);
```

Description

Creates an index file for the specified field of data manager. Data manager need not be open when calling this function.

**dbName* Specifies the complete file name of the data manager with extension name "DM".
**idxName* Specifies the file name of index.
fldNo Specifies the field number to be index.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be DMERR_INVALID_PARAMETER, DMERR_NO_MEMORY, DMERR_INVALID_FIELD or DMERR_FILE_EXISTS.

DMCreateNDX2

Synopsis

```
long DMCreateNDX2(const char *dbName, const char *idxName, WORD fldNo1,  
WORD fldNo2);
```

Description

Creates an index file for the specified two fields of data manager. Data manager need not be open when calling this function.

**dbName* Specifies the complete file name of the data manager with extension name "DM".
**idxName* Specifies the file name of index.
fldNo1 Specifies the field number to be the primary index.
fldNo2 Specifies the field number to be the secondary index.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be DMERR_INVALID_PARAMETER, DMERR_NO_MEMORY, DMERR_INVALID_FIELD or DMERR_FILE_EXISTS.

DMCreateNDX3

Synopsis

```
long DMCreateNDX3(const char *dbName, const char *idxName, WORD fldNo1,
                  WORD fldNo2, WORD fldNo3);
```

Description

Creates an index file for the specified three fields of data manager. Data manager need not be open when calling this function.

<i>*dbName</i>	Specifies the complete file name of the data manager with extension name "DM".
<i>*idxName</i>	Specifies the file name of index.
<i>fldNo1</i>	Specifies the field number to be the primary index.
<i>fldNo2</i>	Specifies the field number to be the secondary index.
<i>fldNo3</i>	Specifies the field number to be the tertiary index.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be DMERR_INVALID_PARAMETER, DMERR_NO_MEMORY, DMERR_INVALID_FIELD or DMERR_FILE_EXISTS.

DMDBCclose

Synopsis

```
long DMDBCclose(DWORD handle);
```

Description

Closes a data manager. Data manager will automatically close when the application calling data manager terminates.

<i>handle</i>	Specifies the handle of the data manager.
---------------	---

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be DMERR_INVALID_PARAMETER, DMERR_INDEX_CANNOT_OPEN or DMERR_INDEX_CANNOT_WRITE.

DMDBCreate

Synopsis

```
long DMDBCreate(const char *dbName, const DMField *dmFld, WORD fldCnt);
```

Description

Creates a new data manager.

<i>*dbName</i>	Specifies the complete file name of the data manager with extension name “DM”.
<i>*dmFld</i>	Specifies the field structure array of the data manager.
<i>fldCnt</i>	Specifies the number of fields.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be DMERR_INVALID_PARAMETER, DMERR_NO_MEMORY, DMERR_DB_CANNOT_OPEN, DMERR_DB_CANNOT_WRITE or DMERR_INVALID_FIELD.

DMDBDelete

Synopsis

```
long DMDBDelete(const char *dbName);
```

Description

Deletes a data manager. Data manager has to be closed before being deleted.

**dbName* Specifies the complete file name of the data manager with extension name “DM”.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be DMERR_INVALID_PARAMETER.

DMDBOpen

Synopsis

```
DWORD DMDBOpen(const char *dbName);
```

Description

Opens a data manager. The maximum of opened data manager at the same time is DM_MAX_DBOPENED.

**dbName* Specifies the complete file name of the data manager with extension name “DM”.

Returned Value

The returned value is data manager handle if this function succeeds. Otherwise, the returned value is 0. DMGetLastErrorCode() obtains detailed failure information which can be DMERR_INVALID_PARAMETER, DMERR_NO_MEMORY, DMERR_MAX_DBOPENED, DMERR_DB_CANNOT_OPEN, DMERR_DB_CANNOT_READ or DMERR_INVALID_FIELD.

DMDelete

Synopsis

```
long DMDelete(DWORD handle);
```

Description

Deletes the current record.

handle Specifies the handle of the data manager.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be DMERR_INVALID_PARAMETER.

DMDeleteNDX

Synopsis

```
long DMDeleteNDX(const char *idxName);
```

Description

Deletes the index file. Index file has to be closed before being deleted.

**idxName* Specifies the file name of index file.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be DMERR_INVALID_PARAMETER.

DMFlush

Synopsis

```
long DMFlush(DWORD handle);
```

Description

Refreshes the records, which is already modified, into file system.

handle Specifies the handle of the data manager.

Returned Value

The returned value is the number of records, which is a non-negative value, if this function succeeds. Otherwise, the returned value is a negative value, which can be DMERR_INVALID_PARAMETER, DMERR_NO_MEMORY or DMERR_DB_CANNOT_WRITE.

DMGetFieldNoByName

Synopsis

```
long DMGetFieldNoByName(DWORD handle, const char *fldName, WORD *fldNo);
```

Description

Obtains the field number of the specified field.

handle Specifies the handle of the data manager.
**fldName* Specifies the field name.
**fldNo* Specifies the memory buffer to which field number is copied.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value,

which can be DMERR_INVALID_PARAMETER or DMERR_NOT_FOUND.

DMGetLastErrorCode

Synopsis

```
long DMGetLastErrorCode(void);
```

Description

Obtains the error code caused by previous data manager function.

Returned Value

The returned value is the error code caused by previous data manager function.

DMGotoBottom

Synopsis

```
long DMGotoBottom(DWORD handle);
```

Description

Moves the record position of data manager to the last record with all records ordered by the index. Index file has to be opened by DMOpenDBWithNDX() or DMOpenDBWithMulNDX() before calling this function. The record position starts from 0; that is, the position of the first record is 0. It is the same with DMSeekWithNDX(*handle*, 0, DMSEEK_END).

handle Specifies the handle of the data manager.

Returned Value

The returned value is record position, which is a non-negative value, if this function succeeds. Otherwise, the returned value is a negative value, which can be DMERR_INVALID_PARAMETER, DMERR_OUT_OF_RANGE, DMERR_NO_MEMORY or DMERR_INDEX_CANNOT_OPEN.

DMGotoTop

Synopsis

```
long DMGotoTop(DWORD handle);
```

Description

Moves the record position of data manager to the first record with all records ordered by the index. Index file has to be opened by DMOpenDBWithNDX() or DMOpenDBWithMulNDX() before calling this function. The record position starts from 0; that is, the position of the first record is 0. It is the same with DMSeekWithNDX(*handle*, 0, DMSEEK_SET).

handle Specifies the handle of the data manager.

Returned Value

The returned value is record position, which is a non-negative value, if this function succeeds. Otherwise, the returned value is a negative value, which can be DMERR_INVALID_PARAMETER, DMERR_OUT_OF_RANGE, DMERR_NO_MEMORY or DMERR_INDEX_CANNOT_OPEN.

DMInitial

Synopsis

```
long DMInitial(void);
```

Description

Allocates memory for data manager. User has to initialize data manager before open it.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be DMERR_NO_MEMORY.

DMNameOfField

Synopsis

```
long DMNameOfField(DWORD handle, WORD fldNo, char *fldName);
```

Description

Obtains the name of the specified field.

<i>handle</i>	Specifies the handle of the data manager.
<i>fldNo</i>	Specifies the field number.
<i>*fldName</i>	Specifies the memory buffer to which field name is copied.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be DMERR_INVALID_PARAMETER or DMERR_INVALID_FIELD.

DMNextWithNDX

Synopsis

```
long DMNextWithNDX(DWORD handle);
```

Description

Moves the record position of data manager to the next record with all records ordered by the index. Index file has to be opened by DMOpenDBWithNDX() or DMOpenDBWithMulNDX() before calling this function. The record position starts from 0; that is, the position of the first record is 0.

<i>handle</i>	Specifies the handle of the data manager.
---------------	---

Returned Value

The returned value is the record number, which is a non-negative value, if this function succeeds. Otherwise, the returned value is a negative value, which can be DMERR_INVALID_PARAMETER, DMERR_OUT_OF_RANGE, DMERR_NO_RECORD, DMERR_NO_MEMORY, DMERR_INDEX_CANNOT_OPEN or DMERR_BOTTOM.

DMNumberOfField

Synopsis

```
WORD DMNumberOfField(DWORD handle);
```

Description

Obtains the number of fields per record.

handle Specifies the handle of the data manager.

Returned Value

The returned value is the number of fields if this function succeeds. Otherwise, the returned value is 0. DMGetLastErrorCode() obtains detailed failure information which can be DMERR_INVALID_PARAMETER.

DMNumberOfRecord

Synopsis

```
WORD DMNumberOfRecord(DWORD handle);
```

Description

Obtains the number of records in data manager.

handle Specifies the handle of the data manager.

Returned Value

The returned value is the number of records, which is a non-negative value, if this function succeeds. Otherwise, the returned value is 0. Since the returned value is also 0 if there is no record in data manager, DMGetLastErrorCode() has to be called to obtain detailed information if returned value is 0. The following are possible values obtained by DMGetLastErrorCode(): DMERR_SUCCESS and DMERR_INVALID_PARAMETER

DMNumberOfRecordByKey

Synopsis

```
WORD DMNumberOfRecordByKey(DWORD handle, const void *key, long keyLen);
```

Description

Obtains the number of records with the index field composed of the specified data. Index file has to be opened by DMOpenDBWithNDX() or DMOpenDBWithMulNDX() before calling this function.

handle Specifies the handle of the data manager.
**key* Specifies the data to search for. It can be a pointer to character or integer.
keyLen Specifies the length of the specified key in bytes.

Returned Value

The returned value is the number of records, which is a non-negative value, if this function succeeds. Otherwise, the returned value is 0. Since the returned value is also 0 if there is no record in data manager, DMGetLastErrorCode() has to be called to obtain detailed information if returned value is 0. The following are possible values obtained by DMGetLastErrorCode(): DMERR_SUCCESS and DMERR_INVALID_PARAMETER.

DMOffsetBykey

Synopsis

```
long DMOffsetBykey(DWORD handle, long recNo, const void *key, long keyLen);
```

Description

Obtains the ranking of *recNo*th record in the records with the index field composed of the specified data. If the index is composed by more than one field, this function only search in the primary field. Index file has to be opened by DMOpenDBWithNDX() or DMOpenDBWithMulNDX() before calling this function.

<i>handle</i>	Specifies the handle of the data manager.
<i>recNo</i>	Specifies the record number.
<i>*key</i>	Specifies the data to search for. It can be a pointer to character or integer.
<i>keyLen</i>	Specifies the length of the specified key in bytes.

Returned Value

The returned value is the record position, which is a non-negative value, if this function succeeds. Otherwise, the returned value is 0. Since the returned value is also 0 if there is no record in data manager, DMGetLastErrorCode() has to be called to obtain detailed information if returned value is 0. The following are possible values obtained by DMGetLastErrorCode(): DMERR_SUCCESS and DMERR_INVALID_PARAMETER.

DMOpenDBWithMulNDX

Synopsis

```
DWORD DMOpenDBWithMulNDX(const char *dbName, const char **idxName, WORD idxCnt);
```

Description

Opens a data manager and the associated index files. This function can open more than one index file.

<i>*dbName</i>	Specifies the complete file name of the data manager with extension name "DM".
<i>**idxName</i>	Specifies the array of the index file names.
<i>idxCnt</i>	Specifies the number of the index files.

Returned Value

The returned value is a handle of the data manager if this function succeeds. Otherwise, the returned value is 0. DMGetLastErrorCode() has to be called to obtain detailed information if returned value is 0. The following are possible values obtained by DMGetLastErrorCode(): DMERR_SUCCESS and DMERR_INVALID_PARAMETER.

DMOpenDBWithNDX

Synopsis

```
DWORD DMOpenDBWithNDX(const char *dbName, const char *idxName);
```

Description

Opens a data manager and the associated index file. This function can only open one index file.

**dbName* Specifies the complete file name of the data manager with extension name "DM".
**idxName* Specifies the index file name.

Returned Value

The returned value is a handle of the data manager if this function succeeds. Otherwise, the returned value is 0. DMGetLastErrorCode() has to be called to obtain detailed information if returned value is 0. The following are possible values obtained by DMGetLastErrorCode(): DMERR_SUCCESS and DMERR_INVALID_PARAMETER.

DMPack

Synopsis

```
long DMPack(DWORD handle);
```

Description

Compacts a data manager.

handle Specifies the handle of the data manager.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be DMERR_INVALID_PARAMETER or DMERR_NO_MEMORY.

DMPositionOfNDX

Synopsis

```
long DMPositionOfNDX(DWORD handle, long recNo);
```

Description

Obtains the ranking of the specified record with all records ordered by the index. Index file has to be opened by DMOpenDBWithNDX() or DMOpenDBWithMulNDX() before calling this function.

handle Specifies the handle of the data manager.
recNo Specifies the position of record.

Returned Value

The returned value is the record position if this function succeeds. Otherwise, the returned value is a negative value, which can be DMERR_INVALID_PARAMETER.

DMPrevWithNDX

Synopsis

```
long DMPrevWithNDX(DWORD handle);
```

Description

Moves the record position of data manager to the previous record with all records ordered by the index. Index file has to be opened by DMOpenDBWithNDX() or DMOpenDBWithMulNDX() before calling this function. The record position starts from 0; that is, the position of the first record is 0.

handle Specifies the handle of the data manager.

Returned Value

The returned value is the record number, which is a non-negative value, if this function succeeds. Otherwise, the returned value is a negative value, which can be DMERR_INVALID_PARAMETER, DMERR_OUT_OF_RANGE, DMERR_NO_RECORD, DMERR_NO_MEMORY, DMERR_INDEX_CANNOT_OPEN or DMERR_BOTTOM.

DMRead

Synopsis

```
long DMRead(DWORD handle, WORD fldNo, void *buffer, DWORD *bufSize);
```

Description

Reads data from specified field of current record. The whole record is copied to buffer if *fldNo* is assigned DM_NO_FIELD. The whole record is in the format as specified in “General Description” of “Data Manager”.

<i>handle</i>	Specifies the handle of the data manager.
<i>fldNo</i>	Specifies the field number.
<i>*buffer</i>	Specifies the memory buffer to which data is copied.
<i>*bufSize</i>	Specifies the length of data. This parameter has to be initialized to the length of memory buffer, and the length of data being copied after calling this function will be assigned to it.

Returned Value

The returned value is the record number of this record being read if this function succeeds. Otherwise, the returned value is a negative value, which can be DMERR_INVALID_PARAMETER, DMERR_OUT_OF_RANGE, DMERR_INVALID_FIELD or DMERR_BUFFER_NOT_ENOUGH.

DMRelease

Synopsis

```
void DMRelease(void);
```

Description

Releases resources hold by a data manager. Data manager will be automatically released when the application calling data manager terminates.

Returned Value

None.

DMSearchFirstWithNDX

Synopsis

```
long DMSearchFirstWithNDX(DWORD handle, const void *key, long keyLen);
```

Description

Obtains the record number of the first record with the primary index composed of the key and all

records ordered by the index. This function only obtains the record number but not moves to that record. Index file has to be opened by `DMOpenDBWithNDX()` or `DMOpenDBWithMulNDX()` before calling this function.

<i>handle</i>	Specifies the handle of the data manager.
<i>*key</i>	Specifies the data to search for. It can be a pointer to character or integer.
<i>keyLen</i>	Specifies the length of the specified key in bytes.

Returned Value

The returned value is the record number, which is a non-negative value, if this function succeeds. Otherwise, the returned value is a negative value, which can be `DMERR_INVALID_PARAMETER` or `DMERR_NOT_FOUND`.

DMSearchLastWithNDX

Synopsis

```
long DMSearchLastWithNDX(DWORD handle, const void *key, long keyLen);
```

Description

Obtains the record number of the last record with the primary index composed of the key and all records ordered by the index. This function only obtains the record number but not moves to that record. Index file has to be opened by `DMOpenDBWithNDX()` or `DMOpenDBWithMulNDX()` before calling this function.

<i>handle</i>	Specifies the handle of the data manager.
<i>*key</i>	Specifies the data to search for. It can be a pointer to character or integer.
<i>keyLen</i>	Specifies the length of the specified key in bytes.

Returned Value

The returned value is the record number, which is a non-negative value, if this function succeeds. Otherwise, the returned value is a negative value, which can be `DMERR_INVALID_PARAMETER` or `DMERR_NOT_FOUND`.

DMSearchWithNDX

Synopsis

```
long DMSearchWithNDX(DWORD handle, const void *key, long keyLen);
```

Description

Obtains the record number of the next record from current position with the primary index composed of the key and all records ordered by the index. This function only obtains the record number but not moves to that record. Index file has to be opened by `DMOpenDBWithNDX()` or `DMOpenDBWithMulNDX()` before calling this function.

<i>handle</i>	Specifies the handle of the data manager.
<i>*key</i>	Specifies the data to search for. It can be a pointer to character or integer.
<i>keyLen</i>	Specifies the length of the specified key in bytes.

Returned Value

The returned value is the record number, which is a non-negative value, if this function succeeds. Otherwise, the returned value is a negative value, which can be `DMERR_INVALID_PARAMETER`

or DMERR_NOT_FOUND.

DMSeek

Synopsis

```
long DMSeek(DWORD handle, long offset, BYTE origin);
```

Description

Moves the record position to the specified position. To seek backward from the end of the data manager, offset must be a negative value and the base point be DMSEEK_END.

<i>handle</i>	Specifies the handle of the data manager.
<i>offset</i>	Specifies the offset value from a base point.
<i>origin</i>	Specifies the base point of seek cursor.
DMSEEK_CUR	Seek from current record.
DMSEEK_SET	Seek from the first record.
DMSEEK_END	Seek from the last record.

Returned Value

The returned value is the record number, which is a non-negative value, if this function succeeds. Otherwise, the returned value is a negative value, which can be DMERR_INVALID_PARAMETER, DMERR_NO_MEMORY or DMERR_OUT_OF_RANGE.

DMSeekWithNDX

Synopsis

```
long DMSeekWithNDX(DWORD handle, long offset, BYTE origin);
```

Description

Moves the record position to the specific position with all records ordered by the index. To seek backward from the end of the data manager, offset must be a negative value and the base point be DMSEEK_END. Index file has to be opened by DMOpenDBWithNDX() or DMOpenDBWithMulNDX() before calling this function.

<i>handle</i>	Specifies the handle of the data manager.
<i>offset</i>	Specifies the offset value from a base point.
<i>origin</i>	Specifies the base point of pointer.
DMSEEK_CUR	Seek from current record
DMSEEK_SET	Seek from the first record
DMSEEK_END	Seek from the last record

Returned Value

The returned value is the record number, which is a non-negative value, if this function succeeds. Otherwise, the returned value is a negative value, which can be DMERR_INVALID_PARAMETER, DMERR_NO_MEMORY, DMERR_OUT_OF_RANGE or DMERR_INDEX_CANNOT_OPEN.

DMSeekWithNDXBykey

Synopsis

```
long DMSeekWithNDXBykey(DWORD handle, long offset, BYTE origin, const void *key,
```


`long keyLen) ;`

Description

Moves the record position to the specific position with the primary index composed of the key and all records ordered by the index. To seek backward from the end of the data manager, offset must be a negative value and the base point be DMSEEK_END. Index file has to be opened by DMOpenDBWithNDX() or DMOpenDBWithMulNDX() before calling this function.

<i>handle</i>	Specifies the handle of the data manager.
<i>offset</i>	Specifies the offset value from a base point.
<i>origin</i>	Specifies the base point of pointer. DMSEEK_CUR Seek from current record DMSEEK_SET Seek from the first record DMSEEK_END Seek from the last record
<i>*key</i>	Specifies the data to search for. It can be a pointer to character or integer.
<i>keyLen</i>	Specifies the length of the specified key in bytes.

Returned Value

The returned value is the record number, which is a non-negative value, if this function succeeds. Otherwise, the returned value is a negative value, which can be DMERR_INVALID_PARAMETER or DMERR_OUT_OF_RANGE.

DMSetMasterNDX

Synopsis

`long DMSetMasterNDX(DWORD handle, WORD fileIdx) ;`

Description

Changes the index from some index files. If data manager is opened via DMOpenDBWithMulNDX(), the index can be switched by this function. For example, there is an array of index files in DMOpenDBWithMulNDX(), says idx[0], idx[1] and idx[2]. User can reorder records by changing *fileIdx* to 0, 1 or 2. The default index depends on the first element of index file array, says idx[0]. Index file has to be opened by DMOpenDBWithNDX() or DMOpenDBWithMulNDX() before calling this function.

<i>handle</i>	Specifies the handle of the data manager.
<i>fileIdx</i>	Specifies the ranking of the designated index file in the index file array of DMOpenDBWithMulNDX().

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be DMERR_INVALID_PARAMETER or DMERR_INDEX_CANNOT_OPEN.

DMSizeOfField

Synopsis

`WORD DMSizeOfField(DWORD handle, WORD fldNo) ;`

Description

Obtains the size of the field. The size is determined when data manager is created.

handle Specifies the handle of the data manager.
fldNo Specifies the field number.

Returned Value

The returned value is the size of the field if this function succeeds. Otherwise, the returned value is 0. DMGetLastErrorCode() obtains detailed failure information which can be DMERR_INVALID_PARAMETER or DMERR_INVALID_FIELD.

DMSizeOfLengthTable

Synopsis

```
WORD DMSizeOfLengthTable(WORD fldNum);
```

Description

Obtains the size of the memory storing the field length. In the record format, length table leads the data and each length is stored in 4 bytes (WORD). That is, this function returns *fldNum* times 4 in bytes.

fldNum Specifies the number of fields.

Returned Value

The returned value is the size of length table.

DMSizeOfRecord

Synopsis

```
long DMSizeOfRecord(DWORD handle);
```

Description

Obtains the size of current record.

handle Specifies the handle of the data manager.

Returned Value

The returned value is the size of current record if this function succeeds. Otherwise, the returned value is a negative value, which can be DMERR_INVALID_PARAMETER, DMERR_OUT_OF_RANGE or DMERR_NO_RECORD.

DMTypeOfField

Synopsis

```
long DMTypeOfField(DWORD handle, WORD fldNo, BYTE *type);
```

Description

Obtains the data type of the specified field.

handle Specifies the handle of the data manager.
fldNo Specifies the field number.
**type* Specifies the field type.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be DMERR_INVALID_PARAMETER.

DMWrite

Synopsis

```
long DMWrite(DWORD handle, WORD fldNo, const void *buffer, DWORD bufSize);
```

Description

Writes data to specified field at current record.

<i>handle</i>	Specifies the handle of the data manager.
<i>fldNo</i>	Specifies the field number.
<i>*buffer</i>	Specifies the data written to field.
<i>bufSize</i>	Specifies the length of data written to field.

Returned Value

The returned value is the record number of current one if this function succeeds. Otherwise, the returned value is a negative value, which can be DMERR_NO_MEMORY, DMERR_OUT_OF_RANGE, DMERR_INVALID_FIELD or DMERR_DATA_TOO_LONG.

Multimedia

In this chapter, multimedia functions are introduced.

Graphics

General Description

In Penbex SDK 2.0, some API's are for color screen up to 16 bits (65536) colors. Palette is introduced in color API's. The palette is capable of storing 256 sets of color, and each is assigned a palette number from 0x00 to 0xFF. The first 20 sets (0x00 – 0x13) and the last 16 sets (0xF0 – 0xFF) are reserved for system. That is, the valid palette number for user is from 0x14 to 0xEF.

Related Data Structure

LBmp Structure

```
typedef struct tagLBmp {
    char IDA[2];           // "bm"
    unsigned char Color;   // 2 or 4
    unsigned char Flag;    //
    unsigned short w;      // Bitmap width
    unsigned short h;      // Bitmap height
}
```

CBmp Structure

```
typedef struct tagCBMP {
    unsigned short bfType;           //
    unsigned long bfSize;            //
    unsigned short bfReserved1, bfReserved2;
    unsigned long bfOffset;          //
    unsigned long biSize;            //
    long biWidth;                    //
    long biHeight;                   //
    unsigned short biPlanes;         //
    unsigned short biBitCount;       //
    unsigned long biCompression;     //
    unsigned long biSizeImage;       //
    long biXPelsPerMeter;            //
    long biYPelsPerMeter;            //
    unsigned long biClrUsed;         //
    unsigned long biClrImportant;    //
}
```

BmpData Structure

```
typedef struct tagHBmp {
    DWORD *dwSignature;  //
    void *pBmpData;      //
    WORD wType;          //
    void *pHandle;        //
    void *pTmpBmp;        //
    BYTE bReserved[8];    //
}
```

```
}
```

RGB24 Structure

```
typedef struct tagRGB24 {  
    unsigned char rgbBlue;    //  
    unsigned char rgbGreen;  //  
    unsigned char rgbRed;    //  
}
```

CoverRect Structure

```
typedef struct tagCoverRect {  
    int x;        //  
    int y;        //  
    int x2;       //  
    int y2;       //  
}
```

PageHandle Structure

```
typedef struct tagPageHandle {  
    DWORD dwChker;    //  
    Void *pFrameBuf;  //  
    WORD width, height; //  
    WORD wWidthByte;  //  
    WORD wClipX;      //  
    WORD wBitPerPixel; //  
    char Reserved[14]; //  
}
```

System Message

No system message for graphics.

GmBmpCopy

Synopsis

```
BOOL GmVPageCopy(BmpData *bmpData, WORD xbmp, WORD ybmp, WORD w, WORD h, int xscrn,
                 int yscrn);
```

Description

Copies an area from bitmap image to LCD screen.

<i>*bmpData</i>	Specifies the bitmap image to be copied.
<i>xbmp</i>	Specifies x-coordinate of left side of the area in bitmap image, in pixels.
<i>ybmp</i>	Specifies y-coordinate of top side of the area in bitmap image, in pixels.
<i>w</i>	Specifies the width of the area.
<i>h</i>	Specifies the height of the area.
<i>xscrn</i>	Specifies x-coordinate of left side of the area in screen, in pixels.
<i>yscrn</i>	Specifies y-coordinate of top side of the area in screen, in pixels.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

GmBmpCopyEx

Synopsis

```
BOOL GmBmpCopyEx(BmpData *bmpData, WORD xbmp, WORD ybmp, WORD w, WORD h, int xscrn,
                 int yscrn, WORD wscrn, WORD hscrn, DWORD zoomopt);
```

Description

Copies an area from bitmap image to LCD screen. The size of the area can be reduced or magnified.

<i>*bmpData</i>	Specifies the bitmap image to be copied.
<i>xbmp</i>	Specifies x-coordinate of left side of the area in bitmap image, in pixels.
<i>ybmp</i>	Specifies y-coordinate of top side of the area in bitmap image, in pixels.
<i>w</i>	Specifies the width of the area.
<i>h</i>	Specifies the height of the area.
<i>xscrn</i>	Specifies x-coordinate of left side of the area in screen, in pixels.
<i>yscrn</i>	Specifies y-coordinate of top side of the area in screen, in pixels.
<i>wscrn</i>	Specifies the width of the area in screen.
<i>hscrn</i>	Specifies the height of the area in screen.
<i>zoomopt</i>	Specifies the way to zoom the area.
	ZOOMOPT_QUICK Quick mode
	ZOOMOPT_DARK_FOCUS Heavy color first
	ZOOMOPT_WHITE_FOCUS Light color first

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

GmCalcStrWidth

Synopsis

```
int GmCalcStrWidth(const char *string, int maxWidth, int *realWidth);
```

Description

Calculates the string's width.

<i>*string</i>	Specifies the string's pointer.
<i>maxWidth</i>	Specifies the maximum length (in pixel) of string to be displayed.
<i>*realWidth</i>	Specifies the memory buffer to which the real width of the string to be displayed is copied. This value must be less than <i>maxWidth</i> .

Returned Value

The returned value is the number of characters to be displayed if this function succeeds. Otherwise, the returned value is 0.

GmCaptureBmp

Synopsis

```
BOOL GmCaptureBmp(WORD x, WORD y, WORD w, WORD h, LBmp *buffer);
```

Description

Copies an area from current draw-page to LBmp structure.

<i>x</i>	Specifies x-coordinate of left side of the area in pixels.
<i>y</i>	Specifies y-coordinate of top side of the area in pixels.
<i>w</i>	Specifies the width of the area.
<i>h</i>	Specifies the height of the area.
<i>*buffer</i>	Specifies the memory buffer to which the image is copied.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

GmCaptureGray4Bmp

Synopsis

```
BOOL GmCaptureGray4Bmp(WORD x, WORD y, WORD w, WORD h, LBmp *buffer);
```

Description

Copies an area from LCD screen to LBmp structure.

<i>x</i>	Specifies x-coordinate of left side of the area in screen, in pixels.
<i>y</i>	Specifies y-coordinate of top side of the area in screen, in pixels.
<i>w</i>	Specifies the width of the area.
<i>h</i>	Specifies the height of the area.
<i>*buffer</i>	Specifies the memory buffer to which the image is copied.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

GmCheckGray4BmpSize

Synopsis

```
DWORD GmCheckGray4BmpSize(WORD w, WORD h);
```

Description

Obtains the size of the memory a 4-gray-scale image occupied.

<i>w</i>	Specifies the width of the 4-gray-scale image.
<i>h</i>	Specifies the height of the 4-gray-scale image.

Returned Value

The returned value is size of the memory if this function succeeds. Otherwise, the returned value is 0.

GmCheckBmpSize

Synopsis

```
WORD GmCheckBmpSize(WORD w, WORD h);
```

Description

Obtains the size of the memory a bitmap image occupied.

<i>w</i>	Specifies the width of the bitmap image.
<i>h</i>	Specifies the height of the bitmap image.

Returned Value

The returned value is size of the memory if this function succeeds. Otherwise, the returned value is 0.

GmContrastDecrease

Synopsis

```
void GmContrastDecrease(void);
```

Description

Decrease the contrast rate of LCD by one level.

Returned Value

None.

GmContrastGetValue

Synopsis

```
WORD GmContrastGetValue(void);
```

Description

Obtains the value of the contrast rate of LCD screen.

Returned Value

The returned value is the contrast rate of LCD screen.

GmContrastIncrease

Synopsis

```
void GmContrastIncrease(void);
```

Description

Increases the contrast rate of LCD by one level.

Returned Value

None.

GmCreatePenQuickVal

Synopsis

```
DWORD GmCreatePenQuickVal(BYTE color, BYTE bkColor, BYTE mode, BYTE pattern);
```

Description

Creates the attributes of the pen in a value. All these attributes are set to a value and can be used as the input parameter of the function GmSetPenQuick().

<i>color</i>	Specifies the color of the pen.
<i>bkColor</i>	Specifies the background color of the pen.
<i>mode</i>	Specifies the mode of the pen.
<i>pattern</i>	Specifies the pattern of the pen.

Returned Value

The returned value is the value composed of all the attributes.

GmDrawBmp

Synopsis

```
void GmDrawBmp(unsigned int x, unsigned int y, LBmp *bmp);
```

Description

Draws a bitmap image at specifies position.

<i>x</i>	Specifies x-coordinate of left side of the screen to put the image, in pixels.
<i>y</i>	Specifies y-coordinate of top side of the screen to put the image, in pixels.
<i>*bmp</i>	Specifies the LBmp structure to be drawn.

Returned Value

None.

GmDrawBmpEx

Synopsis

```
void GmDrawBmpEx(int x, int y, int bmpX, int bmpY, WORD bmpW, WORD bmpH,  
                 const LBmp *bmp);
```

Description

Draws part of a bitmap image at specifies position.

<i>x</i>	Specifies x-coordinate of left side of the screen to put the image, in pixels.
<i>y</i>	Specifies y-coordinate of top side of the screen to put the image, in pixels.
<i>bmpX</i>	Specifies x-coordinate of left side of the area in the image to put to screen, in pixels.
<i>bmpY</i>	Specifies y-coordinate of top side of the area in the image to put to screen, in pixels.
<i>bmpW</i>	Specifies width of the area in the image to put to screen, in pixels.
<i>bmpH</i>	Specifies height of the area in the image to put to screen, in pixels.
<i>*bmp</i>	Specifies the LBmp structure to be drawn.

Returned Value

None.

GmDrawDot

Synopsis

```
void GmDrawDot(int x, int y);
```

Description

Draws a dot at specifies position.

<i>x</i>	Specifies x-coordinate of the dot in pixels.
<i>y</i>	Specifies y-coordinate of the dot in pixels.

Returned Value

None.

GmDrawEllip

Synopsis

```
void GmDrawEllip(int x, int y, int w, int h);
```

Description

Draws an ellipse outline with the current pen color and width.

<i>x</i>	Specifies x-coordinate of the left side of the ellipse, in pixels.
<i>y</i>	Specifies y-coordinate of the top side of the ellipse, in pixels.
<i>w</i>	Specifies the width of the ellipse in pixels.
<i>h</i>	Specifies the height of the ellipse in pixels.

Returned Value

None.

GmDrawLine

Synopsis

```
void GmDrawLine(int x1, int y1, int x2, int y2);
```

Description

Draws a line.

<i>x1</i>	Specifies x-coordinate of the starting dot of the line in pixels.
<i>y1</i>	Specifies y-coordinate of the starting dot of the line in pixels.
<i>x2</i>	Specifies x-coordinate of the ending dot of the line in pixels.
<i>y2</i>	Specifies y-coordinate of the ending dot of the line in pixels.

Returned Value

None.

GmDrawRect

Synopsis

```
void GmDrawRect(int x, int y, int w, int h, int isCorner);
```

Description

Draws a rectangle outline with the current pen color and width.

<i>x</i>	Specifies x-coordinate of the left side of the rectangle, in pixels.
<i>y</i>	Specifies y-coordinate of the top side of the rectangle, in pixels.
<i>w</i>	Specifies the width of the rectangle in pixels.
<i>h</i>	Specifies the height of the rectangle in pixels.
<i>isCorner</i>	Specifies the style of corner of the rectangle. The possible styles are GM_RECT_ROUND_CORNER and GM_RECT_SQUARE_CORNER.

Returned Value

None.

GmDrawSymbol

Synopsis

```
void GmDrawSymbol(int x, int y, const char *symbol);
```

Description

Draws the specifies symbol.

<i>x</i>	Specifies x-coordinate of the symbol in pixels.
<i>y</i>	Specifies y-coordinate of the symbol in pixels.
<i>*symbol</i>	Specifies the symbol name.

Returned Value

None.

GmDrawSymbolCnt

Synopsis

```
void GmDrawSymbolCnt(int x, int y, const char *symbol, int len);
```

Description

Draws the specifies symbol.

<i>x</i>	Specifies x-coordinate of the symbol in pixels.
<i>y</i>	Specifies y-coordinate of the symbol in pixels.
<i>*symbol</i>	Specifies the symbol name.
<i>len</i>	Specifies the maximal length of the symbol.

Returned Value

None.

GmFillArea

Synopsis

```
void GmFillArea(int x, int y, int w, int h, BYTE color);
```

Description

Fills an area with a specified color.

<i>x</i>	Specifies x-coordinate of the left side of the area to be filled, in pixels.
<i>y</i>	Specifies y-coordinate of the top side of the area to be filled, in pixels.
<i>w</i>	Specifies the width of the area to be filled in pixels.
<i>h</i>	Specifies the height of the area to be filled in pixels.
<i>color</i>	Specifies the color to be filled with.

Returned Value

None.

GmFillEllip

Synopsis

```
void GmFillEllip(int x, int y, int w, int h);
```

Description

Fills an ellipse area with the current brush pattern.

<i>x</i>	Specifies x-coordinate of the left side of the ellipse to be filled, in pixels.
<i>y</i>	Specifies y-coordinate of the top side of the ellipse to be filled, in pixels.
<i>w</i>	Specifies the width of the ellipse to be filled, in pixels.
<i>h</i>	Specifies the height of the ellipse to be filled, in pixels.

Returned Value

None.

GmFillRect

Synopsis

```
void GmFillRect(int x, int y, int w, int h, int isCorner);
```

Description

Fills a rectangle area with the current brush pattern.

<i>x</i>	Specifies x-coordinate of the left side of the rectangle to be filled, in pixels.
<i>y</i>	Specifies y-coordinate of the top side of the rectangle to be filled, in pixels.
<i>w</i>	Specifies the width of the rectangle to be filled, in pixels.
<i>h</i>	Specifies the height of the rectangle to be filled, in pixels.
<i>isCorner</i>	Specifies the style of corner of the rectangle. The possible styles are GM_RECT_ROUND_CORNER and GM_RECT_SQUARE_CORNER.

Returned Value

None.

GmFillScreen

Synopsis

```
void GmFillScreen(BYTE cLevel);
```

Description

Fills in the screen.

<i>cLevel</i>	Specifies the color level to be filled. The possible values are GM_COLOR_LEVEL0, GM_COLOR_LEVEL1, GM_COLOR_LEVEL2, GM_COLOR_TOTAL.
---------------	--

Returned Value

None.

GmGetCharWidth

Synopsis

```
int GmGetCharWidth(BYTE char);
```

Description

Obtains the width of the specified character in pixels.

<i>char</i>	Specifies the character.
-------------	--------------------------

Returned Value

The returned value is the width of the specified character.

GmGetCurBlinkRef

Synopsis

```
BYTE GmGetCurBlinkRef(void);
```

Description

Obtains the reflash value of the blinking hardware cursor.

Returned Value

The returned value is the reflash value.

GmGetCurPos

Synopsis

```
void GmGetCurPos(int *x, int *y);
```

Description

Obtains the hardware cursor position.

**x* Specifies the memory buffer to which the x-coordinate of the cursor position is copied.
**y* Specifies the memory buffer to which the y-coordinate of the cursor position is copied.

Returned Value

None.

GmGetDrawPage

Synopsis

```
BOOL GmGetDrawPage(const PageHandle **vPage);
```

Description

Obtains the active draw page.

***vPage* Specifies the returned address of the virtual page or 0 for the LCD screen.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

GmGetFontHeight

Synopsis

```
int GmGetFontHeight(BYTE fontId);
```

Description

Obtains the height of the specified font in pixels.

fontId Specifies the font used in Gm API' s. The possible values are GM_FONT_SMALL, GM_FONT_SMALLBOLD, GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD, GM_FONT_LARGE, GM_FONT_SYMBOL, GM_FONT_TOTAL.

Returned Value

The returned value is the height of the specified font.

GmGetPalette

Synopsis

```
void GmGetPalette(BYTE index, BYTE *red, BYTE *green, BYTE *blue);
```

Description

Obtains the color of the specified palette in RGB value.

<i>index</i>	Specifies the number of the palette.
<i>*red</i>	Specifies the red value of the color in RGB.
<i>*green</i>	Specifies the green value of the color in RGB.
<i>*blue</i>	Specifies the blue value of the color in RGB.

Returned Value

None.

GmGetPalette4

Synopsis

```
BOOL GmGetPalette4(BYTE *dGray, BYTE *lGray);
```

Description

Obtains the hardware palette.

<i>*dGray</i>	Specifies the memory buffer to which the current dark-gray value is copied.
<i>*lGray</i>	Specifies the memory buffer to which the current light-gray value is copied.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

GmGetPenPos

Synopsis

```
void GmGetPenPos(int *x, int *y);
```

Description

Obtains the position of the pen.

<i>*x</i>	Specifies the memory buffer to which the x-coordinate of the pen position is copied.
<i>*y</i>	Specifies the memory buffer to which the y-coordinate of the pen position is copied.

Returned Value

None.

GmGetScreen

Synopsis

```
void GmGetScreen(void *content);
```

Description

Obtains the content of the screen. The buffer size is LCD_WIDTH*LCD_HEIGHT/4.

**content* Specifies the memory buffer to which the content is copied.

Returned Value

None.

GmGetScreenSize

Synopsis

```
void GmGetScreenSize(int *width, int *height);
```

Description

Obtains the width and height of the LCD screen.

**width* Specifies the memory buffer to which the screen width is copied.
**height* Specifies the memory buffer to which the screen height is copied.

Returned Value

None.

GmGetStartAddr

Synopsis

```
void *GmGetStartAddr(void);
```

Description

Obtains the start address of LCD monitor in memory.

Returned Value

The returned value is the address.

GmGetStrWidth

Synopsis

```
int GmGetStrWidth(const char *string);
```

Description

Obtains the width of a string.

**string* Specifies the text of the string.

Returned Value

The returned value is the width of the specified string.

GmGetStrWidthCnt

Synopsis

```
int GmGetStrWidthCnt(const char *string, int len);
```

Description

Obtains the width of a string and the truncates this string to be limited length in pixels.

<i>*string</i>	Specifies the text of the string.
<i>len</i>	Specifies the length to be truncated.

Returned Value

The returned value is the number of characters after truncating to fit the length.

GmGrayArea

Synopsis

```
void GmGrayArea(WORD x, WORD y, WORD w, WORD h);
```

Description

Grays down a specify area.

<i>x</i>	Specifies x-coordinate of the left side of the area to be grayed down, in pixels.
<i>y</i>	Specifies y-coordinate of the top side of the area to be grayed down, in pixels.
<i>w</i>	Specifies the width of the area to be grayed down, in pixels.
<i>h</i>	Specifies the height of the area to be grayed down, in pixels.

Returned Value

None.

GmGrayScreen

Synopsis

```
void GmGrayScreen(int level);
```

Description

Sets the whole LCD screen in gray. To resume the original screen, GmCaptureBmp() and GmDrawBmp() have to be called in sequence if necessary.

<i>level</i>	Specifies the gray level. The only valid valud is 0 so far.
--------------	---

Returned Value

None.

GmIsCurActive

Synopsis

```
BOOL GmIsCurActive(void);
```

Description

Obtains the status of the hardware cursor.

Returned Value

The returned value is TRUE if the cursor is active. Otherwise, the returned value is FALSE.

GmLoadBmp

Synopsis

```
BmpData *GmLoadBmp(WORD type, void *tdata);
```

Description

Loads bitmap image to BmpData structure.

<i>type</i>	Specifies the type of the bitmap image.
<i>*tdata</i>	Specifies the data associated with the type of the bitmap image.
TYPE	TDATA
0	String of file name
1	String of file name which is created via DR_NewP()
2	Reserved
3	Memory address

Returned Value

The returned value is BmpData structure if this function succeeds. Otherwise, the returned value is NULL.

GmLineTo

Synopsis

```
void GmLineTo(int x, int y);
```

Description

Draws a line from current position to a new point. After drawing, the position moves to the new point.

<i>x</i>	Specifies x-coordinate of the new point in pixels.
<i>y</i>	Specifies y-coordinate of the new point in pixels.

Returned Value

None.

GmMagnifyBmp

Synopsis

```
void GmMagnifyBmp(LBmp *bmpDest, LBmp *bmpSrc);
```

Description

Magnifies a bitmap image in LBmp structure.

**bmpDest* Specifies the memory buffer of the destination image.
**bmpSrc* Specifies the memory buffer of the source image.

Returned Value

None.

GmMoveTo

Synopsis

```
void GmMoveTo(int x, int y);
```

Description

Moves the position to the designated one.

x Specifies x-coordinate of the new position in pixels.
y Specifies y-coordinate of the new position in pixels.

Returned Value

None.

GmPutScreen

Synopsis

```
void GmPutScreen(const void *content);
```

Description

Restores the screen contents.

**content* Specifies the content of the screen.

Returned Value

None.

GmReduceBmp

Synopsis

```
void GmReduceBmp(LBmp *bmpDest, LBmp *bmpSrc);
```

Description

Reduces a bitmap image in LBmp structure.

**bmpDest* Specifies the memory buffer of the destination image.
**bmpSrc* Specifies the memory buffer of the source image.

Returned Value

None.

GmReleaseBmp

Synopsis

```
BOOL GmReleaseBmp(BmpData *bmpData);
```

Description

Releases the memory of the BmpData structure.

**bmpData* Specifies the BmpData structure to be released.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

GmRevertRect

Synopsis

```
void GmRevertRect(int x, int y, int w, int h, int isCorner);
```

Description

Reverts of a rectangle.

<i>x</i>	Specifies x-coordinate of the left side of the rectangle to be reverted, in pixels.
<i>y</i>	Specifies y-coordinate of the top side of the rectangle to be reverted, in pixels.
<i>w</i>	Specifies the width of the rectangle to be reverted, in pixels.
<i>h</i>	Specifies the height of the rectangle to be reverted, in pixels.
<i>isCorner</i>	Specifies the style of corner of the rectangle. The possible styles are GM_RECT_ROUND_CORNER and GM_RECT_SQUARE_CORNER.

Returned Value

None.

GmSetBrushPattern

Synopsis

```
BYTE GmSetBrushPattern(BYTE pattern);
```

Description

Sets the pattern of the brush.

pattern Specifies the pattern of the brush. The possible patterns of the brush are GM_BRUSH_PATTERN_SOLID, GM_BRUSH_PATTERN_SLIDER, GM_BRUSH_PATTERN_NULL, GM_BRUSH_PATTERN_TOTAL.

Returned Value

The returned value is the pattern of the brush before being set.

GmSetCurBlinkRef

Synopsis

```
void GmSetCurBlinkRef(BYTE value);
```

Description

Sets the reflash value of the blinking hardware cursor.

value Specifies the cursor value which is from 0x00 to 0xFF.

Returned Value

None.

GmSetCurBlinkTime

Synopsis

```
void GmSetCurBlinkTime(WORD interval);
```

Description

Sets the blinking time of the hardware cursor.

interval Specifies the blinking time interval in milliseconds

Returned Value

None.

GmSetCurPos

Synopsis

```
void GmSetCurPos(int x, int y);
```

Description

Sets the position of the hardware cursor.

x Specifies x-coordinate of the cursor, in pixels.

y Specifies y-coordinate of the cursor, in pixels.

Returned Value

None.

GmSetCurWH

Synopsis

```
void GmSetCurWH(WORD width, WORD height);
```

Description

Sets the width and height of the hardware cursor.

width Specifies the width of the cursor in pixels.
height Specifies the height of the cursor in pixels.

Returned Value

None.

GmSetCustomPattern

Synopsis

```
BYTE GmSetCustomPattern (BYTE *ptmA8) ;
```

Description

Sets a customized pattern.

**ptmA8* Specifies the 8-byte array to be sent.

Returned Value

The returned value is the previous pattern.

GmSetDrawPage

Synopsis

```
BOOL GmSetDrawPage (PageHandle *vPage) ;
```

Description

Sets the active draw page. All the Gm functions activate only on the draw page.

**vPage* Specifies the address of the virtual page or 0 for the LCD screen.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

GmSetFont

Synopsis

```
BYTE GmSetFont (BYTE fontId) ;
```

Description

Sets the font ID.

fontId Specifies the font used in Gm API' s. The possible values are GM_FONT_SMALL, GM_FONT_SMALLBOLD, GM_FONT_MIDDLE, GM_FONT_MIDDLEBOLD, GM_FONT_LARGE, GM_FONT_SYMBOL, GM_FONT_TOTAL.

Returned Value

The returned value is the font ID before being set.

GmSetPalette

Synopsis

```
BOOL GmSetPalette(BYTE index, BYTE red, BYTE green, BYTE blue);
```

Description

Sets the color of the specified palette in RGB value.

<i>index</i>	Specifies the number of the palette.
<i>red</i>	Specifies the red value of the color in RGB.
<i>green</i>	Specifies the green value of the color in RGB.
<i>blue</i>	Specifies the blue value of the color in RGB.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

GmSetPalettes

Synopsis

```
BOOL GmSetPalettes(BYTE index, BYTE size, RGB24 *palettes);
```

Description

Sets the color of the specified palette in RGB value.

<i>index</i>	Specifies the palette number of the first one to be set.
<i>size</i>	Specifies the number of palettes to be set.
<i>*palettes</i>	Specifies the array of palettes in RGB24 structures. The element number of this array should be the same with the <i>size</i> .

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

GmSetPalette4

Synopsis

```
void GmSetPalette4(BYTE dGray, BYTE lGray);
```

Description

Sets the hardware palette.

<i>dGray</i>	Specifies the dark-gray value of the palette. The valid value is from 0 to 15.
<i>lGray</i>	Specifies the light-gray value of the palette. The valid value is from 0 to 15.

Returned Value

None.

GmSetPenBkColor

Synopsis

```
BYTE GmSetPenBkColor (BYTE color) ;
```

Description

Sets the background color of the pen.

color Specifies the color of the pen. The possible values are GM_COLOR_LEVEL0, GM_COLOR_LEVEL1, GM_COLOR_LEVEL2, GM_COLOR_TOTAL.

Returned Value

The returned value is the background color of the pen before being set.

GmSetPenBkColorRGB

Synopsis

```
BYTE GmSetPenBkColorRGB (BYTE red, BYTE green, BYTE blue) ;
```

Description

Sets the background color. The new color is put to the palette with number 0xF1.

red Specifies the red value of the color in RGB.
green Specifies the green value of the color in RGB.
blue Specifies the blue value of the color in RGB.

Returned Value

The returned value is the palette number of the previous background color.

GmSetPenColor

Synopsis

```
BYTE GmSetPenColor (BYTE color) ;
```

Description

Sets the foreground color of the pen.

color Specifies the color of the pen. The possible values are GM_COLOR_LEVEL0, GM_COLOR_LEVEL1, GM_COLOR_LEVEL2, GM_COLOR_TOTAL.

Returned Value

The returned value is the foreground color of the pen before being set.

GmSetPenColorRGB

Synopsis

```
BYTE GmSetPenColorRGB (BYTE red, BYTE green, BYTE blue) ;
```

Description

Sets the color of the pen. The new color is put to the palette with number 0xF0.

<i>red</i>	Specifies the red value of the color in RGB.
<i>green</i>	Specifies the green value of the color in RGB.
<i>blue</i>	Specifies the blue value of the color in RGB.

Returned Value

The returned value is the palette number of the previous pen color.

GmSetPenMode

Synopsis

```
BYTE GmSetPenMode (BYTE mode) ;
```

Description

Sets the mode of the pen.

<i>mode</i>	Specifies the mode of the pen. The possible values are GM_MODE_REPLACE, GM_MODE_XOR, GM_MODE_TRANSPARENT, GM_MODE_TOTAL.
-------------	--

Returned Value

The returned value is the mode of the pen before being set.

GmSetPenPattern

Synopsis

```
BYTE GmSetPenPattern (BYTE pattern) ;
```

Description

Sets the pattern of the pen.

<i>pattern</i>	Specifies the pattern of the pen. The possible values are GM_PEN_PATTERN_SOLID, GM_PEN_PATTERN_DOT, GM_PEN_PATTERN_DASH
----------------	---

Returned Value

The returned value is the pattern of the pen before being set.

GmSetPenPos

Synopsis

```
void GmSetPenPos (int x, int y) ;
```

Description

Sets the new position of the pen.

<i>x</i>	Specifies x-coordinate of the new position, in pixels.
<i>y</i>	Specifies y-coordinate of the new position, in pixels.

Returned Value

None.

GmSetPenQuick

Synopsis

```
DWORD GmSetPenQuick (DWORD value) ;
```

Description

Sets the attributes of the pen. The four attributes of the pen (color, background color, mode and pattern) can be set to a value by calling GmCreatePenQuick() which is the input parameter of this function GmSetPenQuick().

value Specifies the value which is composed of the attributes of the pen.

Returned Value

The returned value the attribute value before being set.

GmSetPenQuickEx

Synopsis

```
DWORD GmSetPenQuickEx (DWORD value, WORD *pwValue) ;
```

Description

Sets the attributes of the pen and obtains the attributes pattern and width of the pen.

value Specifies the value which is composed of the attributes of the pen.

**pwValue* Specifies the memory buffer to which the brush pattern and the pen width are copied.

Returned Value

The returned value is the attribute value of the pen before being set.

GmSetPenWidth

Synopsis

```
BYTE GmSetPenWidth (BYTE width) ;
```

Description

Sets the width of the pen.

width Specifies the new width of the pen to be set.

Returned Value

The returned value is the width of the pen before being set.

GmSetTextClip

Synopsis

```
int GmSetTextClip(int x);
```

Description

Limits the screen as fixed width.

x Specifies the x-coordinate.

Returned Value

The returned value is the x-coordinate.

GmSetSymbol

Synopsis

```
void GmSetSymbol(const void *symbol);
```

Description

Sets the symbol.

**symbol* Specifies the symbol.

Returned Value

None.

GmShiftScrn

Synopsis

```
void GmShiftScrn(int x, int y, int w, int h, int shiftPixel, int shiftMode);
```

Description

Shifts a specified screen area.

<i>x</i>	Specifies x-coordinate of the left side of the area to be shifted, in pixels.
<i>y</i>	Specifies y-coordinate of the top side of the area to be shifted, in pixels.
<i>w</i>	Specifies the width of the area to be shifted, in pixels.
<i>h</i>	Specifies the height of the area to be shifted, in pixels.
<i>shiftPixel</i>	Specifies the offset value to be shifted in pixels.
<i>shiftMode</i>	Specifies the direction to be shifted. The following are possible values:
1	The direction is up.
0	The direction is down.

Returned Value

None.

GmShowCur

Synopsis

```
void GmShowCur(BOOL isShow);
```

Description

Sets hardware cursor to be active or not.

<i>isShow</i>	Specifies the status of showing the cursor.
TRUE	The cursor is active.
FALSE	The cursor is not active.

Returned Value

None.

GmTextOut

Synopsis

```
void GmTextOut(int x, int y, const char *string);
```

Description

Displays a string on the screen.

<i>x</i>	Specifies x-coordinate to display the string, in pixels.
<i>y</i>	Specifies y-coordinate to display the string, in pixels.
<i>*string</i>	Specifies the string text to be displayed.

Returned Value

None.

GmTextOutCnt

Synopsis

```
void GmTextOutCnt(int x, int y, const char *string, int len);
```

Description

Displayed a string on the screen and truncates the string with a specified length.

<i>x</i>	Specifies x-coordinate to display the string, in pixels.
<i>y</i>	Specifies y-coordinate to display the string, in pixels.
<i>*string</i>	Specifies the string text to be displayed.
<i>len</i>	Specifies the maximal length of the string to be displayed.

Returned Value

None.

GmTextOutLF

Synopsis

```
void GmTextOutLF(int x, int y, const char *string, int lSpace);
```

Description

Displays a string on the screen with a specified line space.

<i>x</i>	Specifies x-coordinate to display the string, in pixels.
----------	--

<i>y</i>	Specifies y-coordinate to display the string, in pixels.
<i>*string</i>	Specifies the string text to be displayed.
<i>*string</i>	Specifies the line space when displaying, in pixels.

Returned Value

None.

GmVPageCopy

Synopsis

```
BOOL GmVPageCopy(PageHandle *vPage, WORD xsrc, WORD ysrc, WORD w, WORD h, int xdest,
int ydest);
```

Description

Copies an area from virtual page to current draw page without changing the size.

**vPage* Specifies the address of the virtual page or 0 for the LCD screen.

xsrc Specifies x-coordinate of left side of the area in virtual page, in pixels.

ysrc Specifies y-coordinate of top side of the area in virtual page, in pixels.

w Specifies the width of the area.

h Specifies the height of the area.

xdest Specifies x-coordinate of left side of the area in draw page, in pixels.

ydest Specifies y-coordinate of top side of the area in draw page, in pixels.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

GmVPageCopyEx

Synopsis

```
BOOL GmVPageCopyEx(PageHandle *vPage, WORD xsrc, WORD ysrc, WORD wsrc, WORD hsrc,
int xdest, int ydest, WORD xdest, WORD ydest, DWORD zoomopt);
```

Description

Copies an area from virtual page to current draw page. The size of the area can be reduced or magnified.

**vPage* Specifies the address of the virtual page or 0 for the LCD screen.

xsrc Specifies x-coordinate of left side of the area in virtual page, in pixels.

ysrc Specifies y-coordinate of top side of the area in virtual page, in pixels.

wsrc Specifies the width of the area in virtual page.

hsrc Specifies the height of the area in virtual page.

xdest Specifies x-coordinate of left side of the area in draw page, in pixels.

ydest Specifies y-coordinate of top side of the area in draw page, in pixels.

wdest Specifies the width of the area in draw page.

hdest Specifies the height of the area in draw page.

zoomopt Specifies the way to zoom the area.

ZOOMOPT_QUICK	Quick mode
ZOOMOPT_DARK_FOCUS	Heavy color first
ZOOMOPT_WHITE_FOCUS	Light color first

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

GmVPageCreate

Synopsis

```
PageHandle *GmVPageCreate(WORD w, WORD h);
```

Description

Creates a virtual page. The width of virtual page must be a multiple of 8.

<i>w</i>	Specifies the width of the virtual page.
<i>h</i>	Specifies the height of the virtual page.

Returned Value

The returned value is the address of virtual page if this function succeeds. Otherwise, the returned value is NULL.

GmVPageDelete

Synopsis

```
BOOL GmVPageDelete(PageHandle *vPage);
```

Description

Deletes a virtual page.

<i>*vPage</i>	Specifies the address of the virtual page or 0 for the LCD screen.
---------------	--

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Voice

General Description

Provides API's for voice, such as MP3 format.

Related Data Structure

MP3Info Structure

This structure stores all information of a MP3 file.

```
typedef struct tagMp3Info {  
    BYTE    Version;           // mpeg version, 0=2.5, 1=0.0, 2=2.0, 3=1.0;  
    BYTE    Layer;             // Value;  
    BYTE    ChannelMode;       // 0=Stereo, 1=Joint Stereo, 2=Dual Channel, 3=Single  
                                // Channel;  
    BYTE    bReserved1;        // Reserved  
    DWORD   BitRate;           // Bit Rate  
    DWORD   SamplingRate;      // Sampling rate.  
    WORD    SecSize;           // Section size.  
    BYTE    bReserved2[18];    // Reserved.  
} Mp3Info;
```

System Message

No system message for voice retrieving.

sndClose

Synopsis

```
BOOL sndClose(void);
```

Description

Closes the sound module

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

sndControl

Synopsis

```
BOOL sndControl(int command, void *value, DWORD exValue);
```

Description

Sets some setting for control when playing the sound.

<i>command</i>	Specifies the the setting. The following are possible values:	
	SND_CTRL_SET_PRESCALE_VALUE	Sets the prescale value.
	SND_CTRL_GET_PRESCALE_VALUE	Gets the prescale value.
	SND_CTRL_SET_BASSBOOST_ENABLE_CTRL	Sets the setting for bassboost control.
	SND_CTRL_GET_BASSBOOST_ENABLE_CTRL	Gets the setting for bassboost control.
	SND_CTRL_SET_BASSBOOST_GAIN_VALUE	Sets the gain value of bass.
	SND_CTRL_GET_BASSBOOST_GAIN_VALUE	Gets the gain value of bass.
	SND_CTRL_SET_BASSBOOST_FREQ_VALUE	Sets the freq value of bass.
	SND_CTRL_GET_BASSBOOST_FREQ_VALUE	Gets the freq value of bass.
	SND_CTRL_SET_TONE_ENABLE_CTRL	Sets the setting for tone control.
	SND_CTRL_GET_TONE_ENABLE_CTRL	Gets the setting for tone control.
	SND_CTRL_SET_TONE_BASS_GAIN_VALUE	Sets the gain value of tone bass.
	SND_CTRL_GET_TONE_BASS_GAIN_VALUE	Gets the gain value of tone bass.
	SND_CTRL_SET_TONE_BASS_FREQ_VALUE	Sets the freq value of tone bass.
	SND_CTRL_GET_TONE_BASS_FREQ_VALUE	Gets the freq of tone bass.
	SND_CTRL_SET_TONE_TREBLE_GAIN_VALUE	Sets the gain value of tone bass.
	SND_CTRL_GET_TONE_TREBLE_GAIN_VALUE	Gets the gain value of tone bass.
	SND_CTRL_SET_TONE_TREBLE_FREQ_VALUE	Sets the freq value of tone bass.
	SND_CTRL_GET_TONE_TREBLE_FREQ_VALUE	Gets the freq value of tone bass.
	SND_CTRL_SET_EQ_ENABLE_CTRL	Sets the setting for EQ control.
	SND_CTRL_GET_EQ_ENABLE_CTRL	Gets the setting for EQ control.
	SND_CTRL_SET_EQ_BAND1_VALUE	Sets the band1 value of EQ.
	SND_CTRL_GET_EQ_BAND1_VALUE	Gets the band1 value of EQ.
	SND_CTRL_SET_EQ_BAND2_VALUE	Sets the band2 value of EQ.
	SND_CTRL_GET_EQ_BAND2_VALUE	Gets the band2 value of EQ.

SND_CTRL_SET_EQ_BAND3_VALUE	Sets the band3 value of EQ.
SND_CTRL_GET_EQ_BAND3_VALUE	Gets the band3 value of EQ.
SND_CTRL_SET_EQ_BAND4_VALUE	Sets the band4 value of EQ.
SND_CTRL_GET_EQ_BAND4_VALUE	Gets the band4 value of EQ.
SND_CTRL_SET_EQ_BAND5_VALUE	Sets the band5 value of EQ.
SND_CTRL_GET_EQ_BAND5_VALUE	Gets the band5 value of EQ.
SND_CTRL_SET_EQ_BAND6_VALUE	Sets the band6 value of EQ.
SND_CTRL_GET_EQ_BAND6_VALUE	Gets the band6 value of EQ.
SND_CTRL_GET_MP3_SONG_INFO	Gets the information of MP3 song.
SND_CTRL_SET_MP3_SEC_POSITION	Sets the position of the MP3.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

sndGetFileInfo

Synopsis

```
BOOL sndGetFileInfo(void *fileName, int type, void *fileInfo);
```

Description

(This function is reserved for future implementation.)

sndGetLastError

Synopsis

```
int sndGetLastError(void);
```

Description

Retrieves the error code.

Returned Value

The following are possible returned values:

SND_ERR_NONE	No error.
SND_ERR_HW_NOT_SUPPORT	Not supported by hardware.
SND_ERR_FILE_NOT_EXIST	File not exists.
SND_ERR_FILE_TYPE_NOT_MATCH	Invalid file type.
SND_ERR_SND_NOT_INIT	Module not initialized.
SND_ERR_SND_NEED_CLOSE	Module not closed.
SND_ERR_UNKNOW	Unknown error.
SND_ERR_ACTION_NOT_ACCEPT	Instruction not accepted.
SND_ERR_INTERNAL	Internal error.
SND_ERR_PARAMETER	Invalid parameters.
SND_ERR_NO_MEMORY	Insufficient memory.
SND_ERR_NO_DISK_SPACE	Insufficient disk space.

sndGetVolume

Synopsis

```
BYTE sndGetVolume(void);
```

Description

Obtains the current volume.

Returned Value

The returned value is the current volume (0~255).

sndInit

Synopsis

```
BOOL sndInit(int command, void *buffer, DWORD bufferLen);
```

Description

Initializes the sound module.

<i>command</i>	Specifies the initialization mode.	
	SND_COMM_PLAYBACK	Play wave sound from the information stored in the buffer
	SND_COMM_RECORDING	Record sound to buffer(in wave format)
	SND_COMM_PLAY_WAVE_BUF	Play wave sound from buffer
	SND_COMM_PLAY_WAVE_FILE	Play wave sound from file
	SND_COMM_PLAY_MP3_BUF	Play MP3 from buffer
	SND_COMM_PLAY_MP3_FILE	Play MP3 from file
	SND_COMM_RECORD_BUF	Record sound to buffer(in wave format)
	SND_COMM_RECORD_FILE	Record sound to file(in wave format)
	SND_COMM_PLAY_WAVE_FILE_ARRAY	Play wave sound from list
	SND_COMM_PLAY_MP3_FILE_ARRAY	Play MP3 from list
	SND_COMM_RECORD_ADPCM_BUF	Record adpcm to buffer.
	SND_COMM_RECORD_ADPCM_FILE	Record adpcm to the file
	SND_COMM_PLAY_ADPCM_BUF	Play adpcm from the buffer
	SND_COMM_PLAY_ADPCM_FILE	Play adpcm from the file
<i>*buffer</i>	Specifies the name of the file or the start address of memory.	
<i>bufferLen</i>	Specifies the length of the buffer.	

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

sndPause

Synopsis

```
DWORD sndPause(void);
```

Description

Pauses playing or recording sound.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

SndPlay

Synopsis

```
void SndPlay(void *sndData, unsigned long sndLen);
```

Description

Plays sound.

**sndData* Specifies the sound data array.
sndLen Specifies the length of the sound data array.

Returned Value

None.

sndQueryPosition

Synopsis

```
DWORD sndQueryPosition(void);
```

Description

Queries current position of playing or recording sound.

Returned Value

The returned value is the position of playing or recording sound.

sndSetDirection

Synopsis

```
BOOL sndSetDirection(BOOL isForward);
```

Description

Sets the direction of playing sound.

isForward Specifies the setting to play sound forward or backward.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

sndSetMode

Synopsis

```
BOOL sndSetMode(int mode);
```

Description

Sets the mode of the sound module.

mode Specifies the mode of playing sound.

SND_MODE_ONCE	Play just once
SND_MODE_REPEAT	Play repeatedly
SND_MODE_NEXT	Play next song in play list

SND_MODE_NEXT_SONG	Play next song in play list
SND_MODE_LAST_SONG	Play last song in play list
SND_MODE_SONG_REPEAT	Play a single song repeatedly
SND_MODE_LIST_REPEAT	Play the whole list repeatedly
SND_MODE_SONG_NO_REPEAT	Play without repetition of each song
SND_MODE_LIST_NO_REPEAT	Play without repetition of play list
SND_MODE_LIST_ENTRY_DEMO	Enter demo mode
SND_MODE_LIST_LEAVE_DEMO	Leave demo mode

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

sndSetPosition

Synopsis

```
BOOL sndSetPosition(DWORD position);
```

Description

Sets the offset position of playing sound.

position Specifies the offset position.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

sndSetSpeed

Synopsis

```
BOOL sndSetSpeed(int speed);
```

Description

(This function is reserved for future implementation.)

sndSetVolume

Synopsis

```
BOOL sndSetVolume(BYTE volume);
```

Description

Sets the volume of sound.

volume The value of volume (0~255).

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

sndStart

Synopsis

```
BOOL sndStart(void);
```

Description

Starts playing or recording sound.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

sndStatus

Synopsis

```
DWORD sndStatus(int type);
```

Description

Queries the status of the sound module.

<i>type</i>	Specifies the type to query. The following are possible values:
SND_QUERY_MODE	Query the mode of sound module
SND_QUERY_TIME	Query the time of playing
SND_QUERY_COMMAND	Query the status of sndInit
SND_QUERY_IS_INIT	Query whether initialization is OK
SND_QUERY_IS_PLAYING	Query whether sound is currently being played.
SND_QUERY_IS_RECORD	Query whether in record mode
SND_QUERY_IS_PAUSE	Query whether in pause mode
SND_QUERY_IS_STOP	Query whether in stop mode
SND_QUERY_SONG_LEN	Query the length of song
SND_QUERY_SONG_POS	Query the offset position in current song
SND_QUERY_SONG_NUMBER	Query the position of the song in play list
SND_QUERY_IS_DEMO	Query whether in demo mode
SND_QUERY_IS_SONG_REPEAT	Query whether in single song repetition mode
SND_QUERY_IS_LIST_REPEAT	Query whether in play list repetition mode
SND_QUERY_DEMO_SECOND	Query the time of a single song in demonstration mode.
SND_QUERY_LIST_ID	Query the ID of play list
SND_QUERY_TOTAL_SONG_NUMBER	Query the totality of songs in a play list
SND_QUERY_TOTAL_TIME	Query the total time of a song

Returned Value

The returned value depends on the type of hardware. The following are the hardware type and the associated returned value(s).

Type	Returned Value
SND_QUERY_MODE	Return the status of sndInit
SND_QUERY_TIME	Return the elapse time of the song being played(sec)
SND_QUERY_COMMAND	Return the status of sndInit
SND_QUERY_IS_INIT	Whether initialization is OK(1 for OK, 0 for failure)
SND_QUERY_IS_PLAYING	Whether in play mode(1 for OK,0 for negative)
SND_QUERY_IS_RECORD	Whether in record mode(1 for OK,0 for negative)
SND_QUERY_IS_PAUSE	Whether in pause mode(1 for OK,0 for negative)
SND_QUERY_IS_STOP	Whether in stop mode(1 for OK,0 for negative)

SND_QUERY_SONG_LEN	Length of the song(Bytes)
SND_QUERY_SONG_POS	Offset of the song(Byte)
SND_QUERY_SONG_NUMBER	Offset position of song in play list(number).
SND_QUERY_IS_DEMO	Whether in demo mode(1 for OK,0 for negative)
SND_QUERY_IS_SONG_REPEAT	Whether in single song repetition mode (1 for OK, 0 for negative)
SND_QUERY_IS_LIST_REPEAT	Whether in play list repetition mode (1 for OK,0 for negative)
SND_QUERY_DEMO_SECOND	Total seconds of the song in demo mode
SND_QUERY_LIST_ID	Number of the play list
SND_QUERY_TOTAL_SONG_NUMBER	Totality of all the songs in the play list
SND_QUERY_TOTAL_TIME	Time of the song

sndStop

Synopsis

```
DWORD sndStop(void);
```

Description

Stops playing or recording sound.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Tone

General Description

Provides API' s for single tone, such as DO, RE, ME and so on.

Related Data Structure

None.

System Message

No system message for tone retrieving.

ToneBeep

Synopsis

```
BOOL ToneBeep(void);
```

Description

Plays beep sound.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

ToneClose

Synopsis

```
void ToneClose(void);
```

Description

Releases memory for database.

Returned Value

None.

ToneGetVolume

Synopsis

```
BYTE ToneGetVolume(void);
```

Description

Obtains the volume of tone.

Returned Value

The returned value is the volume of tone. The maximum value is TONE_VOLUME_MAX; the minimum value being TONE_VOLUME_MIN

ToneOpen

Synopsis

```
void ToneOpen(void);
```

Description

Allocates memory for Tone.

Returned Value

None.

TonePlayMIDIFile

Synopsis

```
int TonePlayMIDIFile(char *fileName, BOOL isCircle, WORD second);
```

Description

Plays MIDI file.

<i>*fileName</i>	Specifies the file name of the MIDI file.
<i>isCircle</i>	Specifies the flag of circulating playing. TRUE Circulating playing FALSE Not circulating playing
<i>second</i>	Specifies the time to play in seconds.

Returned Value

The returned value is 1 if the function succeeds. Otherwise, the returned value is 0.

TonePlayMusic

Synopsis

```
int TonePlayMusic(void *file, BOOL isRepeat, UINT second);
```

Description

Plays tone.

<i>*file</i>	Specifies the memory where it starts to play.
<i>isRepeat</i>	Specifies the flag of repeating. TRUE Repeating FALSE Not repeating
<i>second</i>	Specifies the time to play in second.

Returned Value

The returned value is 1 if this function succeeds. Otherwise, the returned value is 0.

ToneSetFrequency

Synopsis

```
void ToneSetFrequency(DWORD freq);
```

Description

Sets frequency of the sound.

<i>freq</i>	Specifies the frequency value. The maximum values are 32768.
-------------	--

Returned Value

None.

ToneSetValue

Synopsis

```
void ToneSetValue(int degree, int keyNo);
```

Description

Sets the degree and key of tone.

degree Specifies the degree of tone. The following are possible values:

Degree	Degree Value
SECOND_DEGREE	1
THIRD_DEGREE	2
FOUR_DEGREE	3
FIVE_DEGREE	4

keyNo Specifies the note of tone. The following are possible values:

KeyNum	Key Number.
DO_VOICE	1
DO_UP	2
RE_VOICE	3
RE_UP	4
MI_VOICE	5
FA_VOICE	6
FA_UP	7
SO_VOICE	8
SO_UP	9
LA_VOICE	10
LA_UP	11
SI_VOICE	12

Returned Value

None.

ToneSetVolume

Synopsis

```
void ToneSetVolume(BYTE scale);
```

Description

Sets the volume of tone

scale Specifies the scale of volume. The maximum value is TONE_VOLUME_MAX; the minimum value is TONE_VOLUME_MIN.

Returned Value

None.

ToneStopFrequency

Synopsis

```
void ToneStopFrequency(void);
```

Description

Stops the frequency.

Returned Value

None.

ToneStopMusic

Synopsis

```
void ToneStopMusic(void);
```

Description

Stops playing tone.

Returned Value

None.

ToneStopValue

Synopsis

```
void ToneStopValue(void);
```

Description

Stops sound.

Returned Value

None.

Communication

In this chapter, some communication functions are introduced, such as TCP/IP network, UART and Infrared.

Uart

General Description

Provides COM port programming.

Related Data Structure

None.

System Message

MT_UART_DATA_COME pMsg->id pMsg->data.p	Receives the data from UART. The data length The pointer of the data
MT_UART_RXBUF_FULL pMsg->id pMsg->data.p	The receive FIFO buffer is full. The data length The pointer of the data
MT_UART_SEND_DONE	Finished send data.
MT_GETXM_BEGIN	Begins to use XMODEM protocol to receive data.
MT_GETXM_INIT_TIMEOUT	Initials hardware time out.
MT_GETXM_FRAME_TIMEOUT	Frame time out
MT_GETXM_CHKSUM_ERR	The checksum error
MT_GETXM_CANCEL	Cancels to receive data.
MT_GETXM_DATA_COME pMsg->id pMsg->data.p	Receives the data from UART using XMODEM protocol. The data length The pointer of the data
MT_GETXM_CONTINUE_TMOUT	Continual receiving time of XMODEM reaches timeout.
MT_GETXM_RETRY_OVER	XMODEM RETRY status is over
MT_GETXM_DONE	Successful receives data from UART using XMODEM protocol.
MT_GETXM_END	Finishes receiving data.
MT_SENDXM_BEGIN	Begins to use XMODEM protocol to send data.
MT_SENDXM_INIT_TIMEOUT	Initials hardware time out.

MT_SENDXM_CLIENT_ACK	Sends ACK to client.
MT_SENDXM_CLIENT_NAK	Sends NAK to client.
MT_SENDXM_CANCEL	Cancels to send data.
MT_SENDXM_DONE	Successful sends data to UART using XMODEM protocol.
MT_SENDXM_END	Finishes sending data.
MT_IRDA_OBEX_CONNECTED pMsg->id	IrOBEX connected. The client or server side of IrOBEX. The following are the possible value: 1 Client connected 2 Server connected
MT_IRDA_OBEX_CONNECT_FAIL	IrOBEX connected failure within the time.
MT_IRDA_OBEX_DISCONNECTED	IrOBEX disconnect.
MT_IRDA_OBEX_SEND_DONE	The data has been send out.
MT_IRDA_OBEX_SEND_FAIL	The data send failed.
MT_IRDA_OBEX_RECEIVED pMsg->data.p	The data has been received. The pointer of the data.
MT_IRDA_OBEX_RECEIV_FAIL	The data receive failed.
MT_IRDA_OBEX_RECEIV_START pMsg->data.dw	Starts to receive data. The data length.
MT_IRDA_OBEX_LOST_LINK	The IrOBEX connection lost.
MT_IRDA_OBEX_RELINKED	The IrOBEX reconnects.
MT_IRDA_OBEX_RECEIVING pMsg->id	Receiving the data. The received count.
MT_IRDA_OBEX_SENDING pMsg->id	Sending the data. The send count.
MT_IRDA_LPT_CONNECTED	IrLpt was connected.
MT_IRDA_LPT_CONNECT_FAIL	IrLpt connects fail.
MT_IRDA_LPT_SEND_DONE	IrLpt data has been send.
MT_IRDA_LPT_SEND_FAIL	Sends IrLpt data fail.

UrtCancelSendData

Synopsis

```
BOOL UrtCancelSendData(void);
```

Description

Cancels transferring data in memory, in interrupt mode.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

UrtCancelXmodemGet

Synopsis

```
BOOL UrtCancelXmodemGet(void);
```

Description

Stops receiving data using XMODEM protocol.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

UrtCancelXmodemSend

Synopsis

```
BOOL UrtCancelXmodemSend(void);
```

Description

Cancels sending data in memory using XMODEM protocol.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

UrtClearRxFIFO

Synopsis

```
void UrtClearRxFIFO(void);
```

Description

Cleans the receiving FIFO buffer of UART.

Returned Value

None.

UrtClose

Synopsis

```
BOOL UrtClose(void);
```

Description

Closes UART.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

UrtCTSStatus

Synopsis

```
BOOL UrtCTSStatus(void);
```

Description

Obtains the state of CTS (Clear To Send) in the Uart.

Returned Value

The returned value is the state of CTS. The following are possible values:

TRUE	Clear CTS, suspending the data transmission from PDA
FALSE	Set CTS, continuing the data transmission from PDA

UrtImmGetData

Synopsis

```
int UrtImmGetData(BYTE *buf, WORD cnt, DWORD timeout);
```

Description

Receives the data from UART in immediate mode.

<i>*buf</i>	Specifies the pointer of memory. buffer to which the data from Uart is copied.
<i>cnt</i>	Specifies the length of the memory. buffer.
<i>timeout</i>	Specifies the time out of receiving data in milliseconds.

Returned Value

The returned value is the length of data which has been send if this function succeeds. Otherwise, the returned value is 0.

UrtImmSendData

Synopsis

```
int UrtImmSendData(BYTE *buf, WORD cnt, DWORD timeout);
```

Description

Sends data through UART in immediate mode

<i>*buf</i>	Specifies the pointer of memory. buffer to which the data from Uart is copied.
<i>cnt</i>	Specifies the length of the memory. buffer.

timeout Specifies the time out of receiving data.

Returned Value

The returned value is the length of data which has received if this function succeeds. Otherwise, the returned value is 0.

UrtIsRxReady

Synopsis

```
BOOL UrtIsRxReady(void);
```

Description

Checks the state of UART FIFO Receiver.

Returned Value

The returned value is TRUE if UART FIFO Receiver is ready. Otherwise, the returned value is FALSE.

UrtIsTxAvailable

Synopsis

```
BOOL UrtIsTxAvailable(void);
```

Description

Checks the state of UART FIFO transmitter.

Returned Value

The returned value is TRUE if UART FIFO transmitter is ready. Otherwise, the returned value is FALSE.

UrtIsTxEmpty

Synopsis

```
BOOL UrtIsTxEmpty (void);
```

Description

Checks the state of UART FIFO transmitter.

Returned Value

The returned value is TRUE if UART FIFO Transmitter is empty. Otherwise, the returned value is FALSE.

UrtOpen

Synopsis

```
BOOL UrtOpen(void);
```


Description

Opens and initialize UART.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

UrtRcvModeBegin

Synopsis

```
BOOL UrtRcvModeBegin(BYTE *buf, WORD bufLen);
```

Description

Registers a memory buffer, and OS stores the data from Uart into this memory buffer.

**buf* Specifies the pointer of memory buffer to which the data from Uart is copied.
bufLen Specifies the length of memory. buffer.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

UrtRcvModeEnd

Synopsis

```
BOOL UrtRcvModeEnd(void);
```

Description

Release the memory which occupied by memory buffer.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

UrtRcvReset

Synopsis

```
BOOL UrtRcvReset(BYTE *buf, WORD bufLen);
```

Description

Re-registres a new memory buffer, and OS stores the data from Uart into this new memory buffer.

**buf* Specifies the new pointer of memory. buffer to which the data from Uart is copied.
bufLen Specifies the new length of memory. buffer.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

UrtSendData

Synopsis

```
BOOL UrtSendData (BYTE *buffer, WORD dataLen) ;
```

Description

Sends data which store in memory in interrupt mode.

**buffer* Specifies the pointer of memory. buffer which contain the data.
dataLen Specifies the length of memory. buffer.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

UrtSet

Synopsis

```
BOOL UrtSet (WORD mode, WORD way, DWORD baud, WORD dataBit, WORD stopBit) ;
```

Description

Initializes UART setting.

mode Specifies the transfer mode. The following are possible values:
 UART_MODE_NORMAL
 UART_MODE_IRDA
 UART_MODE_IRCOMM_MDM
 UART_MODE_CF_CARD
way Specifies the transfer method. The following are possible values:
 UART_WAY_ON_RX
 UART_WAY_ON_TX
 UART_WAY_ON_BOTH (IRDA is disabled)
baud Specifies the transfer speed.
dataBit Specifies the data length to transfer. (There are 8 bits or 7 bits options to choose.)
stopBit Specifies the stop bit(0 or 1).

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

UrtSetRTS

Synopsis

```
void UrtSetRTS (BOOL toggle) ;
```

Description

Sets the state of RTS (Request to Send) in the Uart.

toggle Specifies the setting for RTS. The following are possible values:
 TRUE Clear RTS, suspending the data transmission to PDA
 FALSE Set RTS, continuing the data transmission to PDA

Returned Value

None.

UrtSetWay

Synopsis

```
BOOL UrtSetWay(WORD way);
```

Description

Sets the available data transfer direction.

<i>way</i>	Specifies the data transfer direction. The following are possible values:
UART_WAY_ON_RX	Receiving data only.
UART_WAY_ON_TX	Transmitting data only.
UART_WAY_ON_BOTH	Transmitting and receiving data.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

UrtXGetContinue

Synopsis

```
void UrtXGetContinue(void);
```

Description

Continues receiving data using XMODEM protocol.

Returned Value

None.

UrtXmodemGet

Synopsis

```
BOOL UrtXmodemGet(BOOL isCRCMode);
```

Description

Starts receiving data using XMODEM protocol.

<i>isCRCMode</i>	Specifies to use a Cyclic Redundancy Check (CRC) method for error detection. The following are possible values:
TRUE	Use a Cyclic Redundancy Check (CRC) method.
FALSE	Don't use a Cyclic Redundancy Check (CRC) method.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

UrtXmodemSend

Synopsis

```
BOOL UrtXmodemSend(BYTE *buf, DWORD cnt, BOOL is1KMode);
```

Description

Sends data through UART using XMODEM protocol.

<i>*buf</i>	Specifies the pointer of memory. buffer to which the data from Uart is copied.
<i>cnt</i>	Specifies the length of the memory. buffer.
<i>is1Kmode</i>	Specifies the XMODEM block size. The following are possible values:
TRUE	The block size is 1024 byte.
FALSE	The block size is 128 byte.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Infrareds

General Description

Penbex OS supports two ways for infrared programming. One is via Uart API' s with UART_MODE_IRDA (see the previous section) and the other is via Infrared API' s. Uart API' s only provide data stream transfer but not file; Infrared API' s provide file transfer via OBEX.

To explain the data transmission through infrareds OBEX, we first define server and client. Server is the receiving side and client the transmitting side.

Server Side

The steps of server to receive data are as following.

1. Call IrObexStart(). Server is on and ready to receive data.
2. Check system messages MT_IRDA_OBEX_RECEIV_START, MT_IRDA_OBEX_RECEIVING and MT_IRDA_OBEX_RECEIVED to get the data. There could be more than one MT_IRDA_OBEX_RECEIVING message before receiving is done.
3. Call IrObexEnd() to release OBEX.

Client Side

The complete steps of client to transmit data are as following.

1. Call IrObexStart(). Client is on.
2. Call IrObexConnect(). Client connects to server and is ready to transmit data.
3. Call IrObexSendData(). Start to transmit data.
4. Call IrObexDisconnect(). Close the connection to server.
5. Call IrObexEnd() to release OBEX.

Or there is an API called IrObexSendObject(), which will do all the 5 steps above for you. Besides, IrObexSendObject() also pops up a container indicating data is transmitting.

The API' s with the name preceded by Ir_ are for the users who have strong knowledge of infrared transmission.

Related Data Structure

IrReceived Structure

This structure stores all information received from client.

```
typedef struct tagIrReceived {
    WORD        wIrReceivedType; // Not used
    char        *pFileName;      // A pointer to received file name
    void        *pReceivedData;  // A pointer to received data
    DWORD       dwReceivedWord;  // Length of received data
    int         iGotoRecordNumber;
                                // Used for goto active command
                                // User can define a record number for the receiving application
                                // to move to the specified record of data
    BOOL        fDataProcessed;  // The flag if the application needs to process the data
                                // TRUE   This application processes the data and the received
                                //        data structure will not be passed to next registered
                                //        application
                                // FALSE  This application does not process the data and the
                                //        received data structure will be passed to next
                                //        registered application
    DWORD       dwGotoFieldIdx;  // Used for goto active command
    DWORD       dwGotoPosition;  // Used for goto active command
    DWORD       dwGotoCustom;    // Used for goto active command
    BOOL        fAskUser;        // The flag to pop up a container to ask user if he/she wants to
```

```

// go to a specific record/field after receiving completes
// TRUE   Pop up a container
// FALSE  Not pop up a container
}

```

ObStoreEntry (also *IrObjectHandle) Structure

This structure stores all information to be sent.

```

typedef struct tagObStoreEntry {
    VHANDLE  hServData;    // Internal use only
    BYTE      *pbData;      // A pointer to data to be sent
    BYTE      *pbFileName;  // A pointer to file name of the file to be sent
    BYTE      *pbAppName;   // A pointer to the application sending out data
    WORD      wNameLen;     // Not used
    DWORD     dwDataLen;    // Length of data to be sent
    DWORD     dwIndex;      // Internal use only
    BYTE      bFlags;       // Internal use only
    BYTE      bReserved[7]; // Reserved for future use
}

```

System Message

Message of Printer

MT_IRDA_LPT_CONNECTED	Connect to printer in a time interval successful.
MT_IRDA_LPT_CONNECT_FAIL	Cannot connect to printer in a time interval.
MT_IRDA_LPT_SEND_DONE	Data is sent successfully.
MT_IRDA_LPT_SEND_FAIL	Data cannot be sent successfully.

Message of IrOBEX Server

MT_IRDA_OBEX_RECEIV_START pMsg->data	Data are coming and the length of data is received. data.dw is the data length
MT_IRDA_OBEX_RECEIVING pMsg->id	The received data here means that the data received between last MT_IRDA_OBEX_RECEIVING message and this one. Length of received data
MT_IRDA_OBEX_RECEIVED pMsg->data	All data are received. data.p is the pointer of IrObjectHandle.
MT_IRDA_OBEX_RECEIV_FAIL	
MT_IRDA_OBEX_LOST_LINK	The connect is interrupted over 40 seconds.
MT_IRDA_OBEX_RELINKED	The connection is interrupted less than 40 seconds and is re-connected again.

Message of IrOBEX Server

MT_IRDA_OBEX_CONNECTED	Connect to server.
MT_IRDA_OBEX_CONNECT_FAIL	Cannot connect to server.
MT_IRDA_OBEX_DISCONNECTED	Stop the connection to server.
MT_IRDA_OBEX_SEND_DONE	Data transmission is done.

MT_IRDA_OBEX_SEND_FAIL	Data transmission is failed.
MT_IRDA_OBEX_SENDING pMsg->id	Data is being transmitted. Length of sent data

IrLptBusy

Synopsis

```
BOOL IrLptBusy(void);
```

Description

Checks if the printer is available or not.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

IrLptEnd

Synopsis

```
void IrLptEnd(void);
```

Description

Closes the connection to printer.

Returned Value

None.

IrLptSend

Synopsis

```
BOOL IrLptSend(BYTE *buffer, WORD length);
```

Description

Sends data to printer through infrared rays. Printer has to be connected before calling this function. The data to be printed out must in bit values following the format of printer.

<i>*buffer</i>	Specifies the data to be printed out.
<i>length</i>	Specifies the length of printed data.

Returned Value

The returned value is TRUE if the printer is available. Otherwise, the returned value is FALSE.

IrLptStart

Synopsis

```
BOOL IrLptStart(DWORD timeout, char mode);
```

Description

Connects to printer through infrared rays. If it does not connect to printer in a timeout interval, it stops to try again and fails.

<i>timeout</i>	Specifies the timeout interval in milliseconds.
<i>mode</i>	Specifies the mode to connect to printer.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

IrObexAutoRecvSwitch

Synopsis

```
void IrObexAutoRecvSwitch(BOOL switch);
```

Description

Turns the hardware function of automatically receiving on or off. The difference between IrObexAutoRecvSwitch() and IrObexSetAutoRecvMode() is the hardware and software. Since IrObexAutoRecvSwitch() only sets hardware but not software which may cause inconsistency, IrObexSetAutoRecvMode() is recommended for setting automatically receiving.

<i>switch</i>	Specifies the flag of turning infrared on/off.
TRUE	Turn on infrared
FALSE	Turn off infrared

Returned Value

None.

IrObexConnect

Synopsis

```
BOOL IrObexConnect(DWORD timeout);
```

Description

Connects to server through OBEX. If it does not connect to server in a timeout interval, it stops to try again and fails.

<i>timeout</i>	Specifies the timeout interval in seconds.
----------------	--

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

IrObexDisconnect

Synopsis

```
void IrObexDisconnect(void);
```

Description

Closes the connection to server.

Returned Value

None.

IrObexEnd

Synopsis

```
BOOL IrObexEnd(void);
```

Description

Releases the OBEX.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

IrObexGetAutoRecvMode

Synopsis

```
BOOL IrObexGetAutoRecvMode(void);
```

Description

Obtains the status of infrared automatically receiving.

Returned Value

The returned value is TRUE if is in automatically receiving mode. Otherwise, the returned value is FALSE.

IrObexGetData

Synopsis

```
void IrObexGetData(IrObjectHandle irObex);
```

Description

(This function is reserved for future implementation.)

IrObexIsConnected

Synopsis

```
WORD IrObexIsConnected(void);
```

Description

Checks if the infrared connection is ready or not. This function has to be called after client requests a connection to server, or it is always not connected.

Returned Value

The returned value is status of connection. The following are possible values:

- 0 Not connected
- 1 Connected as a client
- 2 Connected as a server

IrObexRegister

Synopsis

```
void IrObexRegister(const char *extName);
```

Description

Registers the file type by the file extension name. If an application registers one specific file, this application will get the system message APP_RUN_IRDA_NOTIFY when such file is received. Each application can register one type at most. If more than one application register the same file type, only one of them can process that file when file is coming.

**extName* Specifies the file extension name.

Returned Value

None.

IrObexSendData

Synopsis

```
void IrObexSendData(IrObjectHandle irObex);
```

Description

Transmits data via infrared.

irObex Specifies the ObStoreEntry structure to be sent.

Returned Value

None.

IrObexSendObject

Synopsis

```
BOOL IrObexSendData(const char *fileName, void *data, DWORD length);
```

Description

Transmits data via infrared.

**fileName* Specifies the file name for the remote side to see.
**data* Specifies the data to be transmitted. This can be in any data type.
length Specifies the length of the data.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

IrObexSetAutoRecvMode

Synopsis

```
void IrObexSetAutoRecvMode(BOOL switch);
```

Description

Turns the hardware function of automatically receiving on or off.

<i>switch</i>	Specifies the flag of turning infrared on/off.
TRUE	Turn on infrared
FALSE	Turn off infrared

Returned Value

None.

IrObexStart

Synopsis

```
BOOL IrObexStart(void);
```

Description

Prepares the OBEX to be ready to receive or transmit data.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Ir_AdvanceCredit

Synopsis

```
void Ir_AdvanceCredit(IrConnect *con, BYTE credit);
```

Description

Advances credit to the other side. The total amount of credit should not exceed 127. The credit passed by this function is added to existing available credit, and this number must not exceed 127.

<i>*con</i>	The pointer of the connection.
<i>credit</i>	The amount of the credit to advance.

Returned Value

None.

Ir_Bind

Synopsis

```
IrStatus Ir_Bind(IrConnect *con, IrCallBack callBack);
```

Description

Obtains a local LSAP selector and register the connection.

<i>*con</i>	Specifies the connection. Other variables are used internally. User shouldn't modify them.
<i>callBack</i>	Specifies the call back function.

Returned Value

The returned value is `IR_STATUS_SUCCESS` if this function succeeds. Otherwise, the returned value is `IR_STATUS_FAILED`.

Ir_ConnectIrLap

Synopsis

```
IrStatus Ir_ConnectIrLap(IrDeviceAddr deviceAddr);
```

Description

Starts an IrLAP connection.

deviceAddr Specifies the 32-bit address of device to which the connection is made.

Returned Value

The returned value is `IR_STATUS_PENDING` if this function succeeds. Otherwise, the returned value can be `IR_STATUS_MEDIA_BUSY` because the media is busy. The result is returned via the callback with the event `LEVENT_LAP_CON_CNF` or `LEVENT_LAP_DISCON_IND`.

Ir_ConnectReq

Synopsis

```
IrStatus Ir_ConnectReq(IrConnect *con, IrPacket *packet, BYTE credit);
```

Description

Requests an IrLMP or Tiny TP connection.

<i>*con</i>	Specifies the connection.
<i>*packet</i>	Specifies the IrPacket structure that contains connection data.
<i>credit</i>	Specifies the initial amount of credit advanced to the other side. Must be less than 127.

Returned Value

The returned value is `IR_STATUS_PENDING` if this function succeeds. Otherwise, the returned value can be `IR_STATUS_NO_IRLAP` because no IrLAP connection exists, or `IR_STATUS_FAILED` because operation failed. The result is returned via the callback with the event `LEVENT_LM_CON_CNF`, `LEVENT_LM_DISCON_IND` or `LEVENT_PACKET_HANDLED`.

Ir_ConnectRsp

Synopsis

```
IrStatus Ir_ConnectRsp(IrConnect *con, IrPacket *packet, BYTE credit);
```

Description

Accepts an incoming connection that has been signaled via the callback with the event `LEVENT_LM_CON_IND`.

**con* Specifies the connection.

<i>*packet</i>	Specifies the IrPacket structure that contains connection data.
<i>credit</i>	Specifies the initial amount of credit advanced to the other side. Must be less than 127.

Returned Value

The returned value is IR_STATUS_PENDING if this function succeeds. Otherwise, the returned value can be IR_STATUS_NO_IRLAP because no IrLAP connection exists, or IR_STATUS_FAILED because operation failed. The result is returned via the callback with the event LEVENT_PACKET_HANDLED.

Ir_DataReq

Synopsis

```
IrStatus Ir_DataReq(IrConnect *con, IrPacket *packet);
```

Description

Sends a data packet.

<i>*con</i>	Specifies the connection.
<i>*packet</i>	Specifies the IrPacket structure that contains connection data.

Returned Value

The returned value is IR_STATUS_PENDING if this function succeeds. Otherwise, the returned value can be IR_STATUS_FAILED because operation failed. The result is returned via the callback with the event LEVENT_PACKET_HANDLED.

Ir_DisconnectIrLap

Synopsis

```
IrStatus Ir_DisconnectIrLap(void);
```

Description

Disconnects the IrLAP connection. The callback of all bound IrConnect structures is called with event LEVENT_LAP_DISCON_IND. Also, all IrPacket structures owned by the stack are returned to their owners via LEVENT_PACKET_HANDLED events.

Returned Value

The returned value is IR_STATUS_PENDING if this function succeeds. Otherwise, the returned value can be IR_STATUS_NO_IRLAP because no IrLAP connection exists. All bound IrConnect structures will be called back when complete.

Ir_DiscoverReq

Synopsis

```
IrStatus Ir_DiscoverReq(IrConnect *con);
```

Description

Starts an IrLMP discovery process.

**con* Specifies the connection.

Returned Value

The returned value is IR_STATUS_PENDING if this function succeeds. Otherwise, the returned value can be IR_STATUS_MEDIA_BUSY because the media is busy, or IR_STATUS_FAILED because operation failed. The result is returned via the callback with the event LEVENT_DISCOVERY_CNF.

Ir_End

Synopsis

```
BOOL Ir_End(void);
```

Description

Stops IrDA service.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Ir_IAS_Add

Synopsis

```
IrStatus Ir_IAS_Add(IrIasObject *obj);
```

Description

Adds an IAS object to the IAS database. The memory for the object must exist for as long as the object is in the database.

**obj* Specifies the IAS object.

Returned Value

The returned value is IR_STATUS_SUCCESS if this function succeeds. Otherwise, the returned value is IR_STATUS_FAILED.

Ir_IAS_GetInteger

Synopsis

```
DWORD Ir_IAS_GetInteger(IrIasQuery *token);
```

Description

Obtains the integer value of the current result item, assuming current result item type is IAS_ATTRIB_INTEGER.

**token* Specifies the IAS query.

Returned Value

The returned value is an integer value of current result item.

Ir_IAS_GetIntLsap

Synopsis

```
BYTE Ir_IAS_GetIntLsap(IrIasQuery *token);
```

Description

Obtains the 8-bit integer value of the current result item, assuming current result item type is IAS_ATTRIB_INTEGER. Usually integer values returned in a query are LSAP selectors.

**token* Specifies the IAS query.

Returned Value

The returned value is a 8-bit integer value of current result item.

Ir_IAS_GetObjectID

Synopsis

```
WORD Ir_IAS_GetObjectID(IrIasQuery *token);
```

Description

Obtains the unique object ID of the current result item.

**token* Specifies the IAS query.

Returned Value

The returned value is a unique object ID of current result item.

Ir_IAS_GetOctetString

Synopsis

```
BYTE* Ir_IAS_GetOctetString(IrIasQuery *token);
```

Description

Obtains an octet string, assuming that the current result item is a type of IAS_ATTRIB_OCTET_STRING.

**token* Specifies the IAS query.

Returned Value

The returned value is a pointer to an octet string.

Ir_IAS_GetOctetStringLength

Synopsis

```
WORD Ir_IAS_GetOctetStringLength(IrIasQuery *token);
```

Description

Obtains the length of an octet string. Assuming that the current result item is a type of

IAS_ATTRIB_OCTET_STRING.

**token* Specifies the IAS query.

Returned Value

The returned value is the length of an octet string.

Ir_IAS_GetType

Synopsis

```
BYTE Ir_IAS_GetType(IrIasQuery *token);
```

Description

Obtains the type of the current result item.

**token* Specifies the IAS query.

Returned Value

The returned value is the type of the current result item.

Ir_IAS_GetUserString

Synopsis

```
BYTE* Ir_IAS_GetUserString(IrIasQuery *token);
```

Description

Obtains the user string. Assuming that the current result item is for the type of IAS_ATTRIB_USER_STRING.

**token* Specifies the IAS query.

Returned Value

The returned value is a pointer of a user string.

Ir_IAS_GetUserStringCharSet

Synopsis

```
IrCharSet Ir_IAS_GetUserStringCharSet(IrIasQuery *token);
```

Description

Obtains the character set of the user string. Assuming that the current result item is a type of IAS_ATTRIB_USER_STRING.

**token* Specifies the IAS query.

Returned Value

The returned value is a character set of the user string.

Ir_IAS_GetUserStringLen

Synopsis

```
BYTE Ir_IAS_GetUserStringLen(IrIasQuery *token);
```

Description

Obtains the length of the user string. Assuming that the current result item is a type of IAS_ATTRIB_USER_STRING.

**token* Specifies the IAS query.

Returned Value

The returned value is the length of the user string.

Ir_IAS_Next

Synopsis

```
BYTE *Ir_IAS_Next(IrIasQuery *token);
```

Description

Moves the internal pointer to next result item.

**token* Specifies the IAS query.

Returned Value

The returned value is a pointer of next result item; 0 means no more result item.

Ir_IAS_Query

Synopsis

```
IrStatus Ir_IAS_Query(IrIasQuery *token);
```

Description

Queries other device IAS database.

**token* Specifies the IAS query.

Returned Value

The returned value is IR_STATUS_PENDING if this function succeeds. Otherwise, the returned value can be IR_STATUS_NO_IRLAP because no IrLAP connection exists, or IR_STATUS_FAILED because operation failed. The result is returned via the callback.

Ir_IAS_SetDeviceName

Synopsis

```
IrStatus Ir_IAS_SetDeviceName(BYTE *name, BYTE len);
```

Description

Sets the value field of the device name attribute of the device object in the IAS database.

**name* Specifies the IAS value for the device.
len Specifies the length of the IAS value.

Returned Value

The returned value is IR_STATUS_SUCCESS if this function succeeds. Otherwise, the returned value is IR_STATUS_FAILED.

Ir_IAS_StartResult

Synopsis

```
void Ir_IAS_StartResult(IrIasQuery *token);
```

Description

Puts the internal pointer at the start of the result buffer.

**token* Specifies the IAS query.

Returned Value

None.

Ir_IsIrLapConnected

Synopsis

```
BOOL Ir_IsIrLapConnected(void);
```

Description

Checks whether IrLAP connection exists.

Returned Value

The returned value is TRUE if IrLAP connection exists. Otherwise, the returned value is FALSE.

Ir_IsMediaBusy

Synopsis

```
BOOL Ir_IsMediaBusy(void);
```

Description

Checks if the media is busy.

Returned Value

The returned value is TRUE if media is busy. Otherwise, the returned value is FALSE.

Ir_IsNoProgress

Synopsis

```
BOOL Ir_IsNoProgress(void);
```

Description

Checks if IrLAP is not making progress.

Returned Value

The returned value is TRUE if IrLAP is progressing normal. Otherwise, the returned value is FALSE.

Ir_IsRemoteBusy

Synopsis

```
BOOL Ir_IsRemoteBusy(void);
```

Description

Checks if other device is IrLAP busy.

Returned Value

The returned value is TRUE if IrLAP for the other device is busy. Otherwise, the returned value is FALSE.

Ir_LocalBusy

Synopsis

```
void Ir_LocalBusy(BOOL flag);
```

Description

Sets the IrLAP local busy flag. When the local busy flag sets to TRUE, the IrLAP layer of the device sends RNR. In response, the remote device will send no more IrLAP data until the local busy flag sets to FALSE.

flag Specifies the flag to set local IrLAP.

Returned Value

None.

Ir_MaxRxSize

Synopsis

```
WORD Ir_MaxRxSize(IrConnect *con);
```

Description

Obtains the maximum size of buffer that can be sent by the other device.

**con* Specifies the connection.

Returned Value

The returned value is the maximum number in bytes that can be received.

Ir_MaxTxSize

Synopsis

```
WORD Ir_MaxTxSize(IrConnect *con);
```

Description

Obtains the maximum size allowed for a transmit packet.

**con* Specifies the connection.

Returned Value

The returned value is the maximum number of bytes for a transmit packet.

Ir_SetConTypeLMP

Synopsis

```
void Ir_SetConTypeLMP(IrConnect *con);
```

Description

Sets the type of the connection to IrLMP.

**con* Specifies the connection.

Returned Value

None.

Ir_SetConTypeTTP

Synopsis

```
void Ir_SetConTypeTTP(IrConnect *con);
```

Description

Sets the type of the connection to Tiny TP.

**con* Specifies the connection.

Returned Value

None.

Ir_SetDeviceInfo

Synopsis

```
IrStatus Ir_SetDeviceInfo(BYTE *info, BYTE len);
```

Description

Sets the XID info string used during discovery to the given string and length. The XID info string

contains hints and the nickname of the device. The size cannot exceed 23 bytes.

**info* Specifies the device information.
len Specifies the length of the device information.

Returned Value

The returned value is IR_STATUS_SUCCESS if this function succeeds. Otherwise, the returned value is IR_STATUS_FAILED.

Ir_Start

Synopsis

```
BOOL Ir_Start(void);
```

Description

Starts infrared service.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Ir_TestReq

Synopsis

```
IrStatus Ir_TestReq(IrDeviceAddr devAddr, IrConnect *conn, IrPacket *packet);
```

Description

Sends a test frame.

devAddr Specifies the address of device where testing frame will be sent. If devAddr equals 0, the testing frame will broadcast to all devices.
**conn* Specifies the connection.
**packet* Specifies the IrPacket structure that contains connection data.

Returned Value

The returned value is IR_STATUS_PENDING if this function succeeds. Otherwise, the returned value can be IR_STATUS_MEDIA_BUSY because the media is busy, or IR_STATUS_FAILED because operation failed. The result is returned via the callback with the event LEVENT_TEST_CNF.

Ir_Unbind

Synopsis

```
IrStatus Ir_Unbind(IrConnect *con);
```

Description

Unbinds the IrConnect structure and releases its LSAP selector.

**con* Specifies the connection.

Returned Value

The returned value is `IR_STATUS_SUCCESS` if this function succeeds. Otherwise, the returned value is `IR_STATUS_FAILED`.

TCP/IP Networking

General Description

This section provides some standard socket API's, and some API's for e-mail retrieving.

Related Data Structure

NETID Structure

This structure stores all information for IP address.

```
typedef struct tagId {
    BYTE        is_ip_addrs[4];    // IP address number
}
```

NETADDR Structure

This structure stores all information for the remote client's address.

```
typedef struct tagNetAddr{
    short        family;            // The type of address. It should always be NU_FAMILY_IP.
    WORD         wPortNo;           // Port number
    NETID        id;                // IP address for the host machine
    char *       pcName;            // machine's name
}
```

NETSOCKETADDR Structure

This structure stores all information for socket.

```
typedef struct tagNetSocketAddr{
    BYTE        ip[4];              // IP address for the host machine
    short       port_num;           // Port number
    short       pad;                // Reserved
}
```

NETFD Structure

This structure stores the bitmap for sockets on which the caller wants to check for data.

```
typedef struct tagNetFd{
    DWORD       dwBitmaps[3];
}
```

NETHOST Structure

This structure stores all information for host.

```
typedef struct tagNetHost{
    char *       pcName;            // Host name
    short        addrType;          // The type of address. It should always be NU_FAMILY_IP
    short        length;            // It should always be 4.
    char *       pcAddr;            // IP address for the host machine
}
```

NETIOCTL Structure

This structure stores IP address associated with a interface of physical layer device.

```
typedef struct tagNetIoCtl{
    BYTE *       s_optval;          // propocol type. The following are possible values:
                                    // PPP_Link
                                    // SLIP_Link

    union
    {

```



```

        BYTE      s_ipaddr[4]; // IP address
    }s_ret;
}

```

NETPOP3 Structure

This structure stores all information for POP3.

```

typedef struct tagPop3Struct{
    short      tv;                // timeout for receives, in seconds
    short      socket;            // handle to created socket
    short      state;             // current state of POP3 conversation
    short      error;
    WORD       flags;             // flags
    BYTE       user[32];          // null terminated user logon name
    BYTE       pass[32];          // null terminated clear text password.
    BYTE       apop[132];         // APOP shared secret password
    BYTE       tstamp[132];       // timestamp from server's banner
    WORD       resp_len;          // response length in bytes.
    BYTE       *resp;             // response if any from pop3 server.
                                    // POP3_Logon will allocate resp memory
                                    // and POP3_Logoff will deallocate it.
    NETADDR    serv;             // address of the POP3 server.
    short (*user_write)(long, char *, long, short); // user's write function,
                                    // when using NU_POP3_Retrieve function
    short (*list_func)(BYTE *, WORD); // user function when getting a list
                                    // of all messages from POP3 server
                                    // using the NU_POP3_List function
}

```

NETPOP3STATE Structure

This structure stores all information for POP3 state

```

typedef struct tagPop3State{
    long      cnt;                // number of messages on POP3 server.
    Long      msgSize;            // number of octets used in 'cnt' messages.
}

```

NETSMTP Structure

This structure stores all information for SMTP state

```

struct tagSmtplibStruct{
    short      tv;                // In client mode it is the timeout for
                                    // receives, in seconds. In server mode
                                    // it should be a larger value. (60+)
    short      cmd;               // current command being worked on.
    short      sock;              // listen socket handle.
    short      state;             // current state of SMTP conversation
    short      error;
    BYTE       banner[NET_SMTP_MAX_BANNER]; // null terminated logon banner
    WORD       resp_len;          // response length in bytes.
    BYTE       resp[NET_SMTP_MAX_REPLY+1]; // response if any from smtp server.
    NETADDR    serv;             // address of the SMTP server.
    WORD       itime;             // inactive timeout in seconds. (180+)
    short      backlog;           // # of backlogs for NU_Listen
    Long       sizelimit;         // largest size of a received message
    BYTE       user[NET_SMTP_MAX_USER]; // null terminated user name.
    BYTE       *domain;          // null terminated domain name.
    BYTE       *configdir;        // directory that contains config files.
    BYTE       *pofficedir;       // post office directory
    BYTE       *tempdir;          // temp file directory
}

```

```

BYTE      *forwardto;           // name of system to forward to
BYTE      *from;                // mail is from
BYTE      *hostname;           // a valid IP hostname, in server mode
BYTE      *reason;              // failure reason.
short      (*mbpath) (NETSMTP *, BYTE *, BYTE **);
short      (*storeto) (NETSMTP *, BYTE *, short);
short      (*storemsg) (NETSMTP *, BYTE *, long, short);
short      (*smtp_send) (NETSMTP *, BYTE *, long, short);
BYTE *      (*verify) (NETSMTP *, BYTE *);           // verify that a user is local
short      (*expand) (NETSMTP *, BYTE *);           // expand a local mailing list
short      (*process_mail) (NETSMTP *);             // process mail after user quits
short      (*mail_notice) (NETSMTP *, BYTE *);      // send undeliverable mail
BYTE *      (*datetime) (void);                     // returns date and time
short      sck;                                     // socket fd to client.
BYTE      id;                                       // used to create temprary files.
DWORD      ct;                                     // logon time
DWORD      lt;                                     // time of last command.
DWORD      num;                                    // used in temp filenames.
short      transfer;                               // data transfer mode
short      mode;                                    // parts that have been done so far.
BYTE *      chostname;
};

```

NETDIALINFO Structure

This structure stores all information for Dial data.

```

typedef struct tagNetDialInfo{
    long lBaudRate;           // Modem baud rate
    int speaker;              // Modem speaker volumn (0...3)
    int flowCtl;              // Modem flow control (0...2)
    BOOL fTone;               // 1: Tone 0: pulse
    char    cInitStrA[NDI_INITSTR_LEN+1]; // Modem initial string
    char    cAccountA[NDI_ACCOUNT_LEN+1]; // ISP account
    char    cPasswordA[NDI_PASSWORD_LEN+1]; // ISP password
    char    cPhoneNoA[NDI_PHONENUM_LEN+1]; // ISP TEL No.
    int mode;                 // Modem Mode (0: RS-232, 1:IRDA)
    char cSlipUNPrompt[NDI_UNPROMPT_LEN+1]; // Slip user name prompt
    char cSlipPSWPrompt[NDI_PSWPROMPT_LEN+1]; // Slip Password prompt
    char cSlipSetup[NDI_SLIPSETUP_LEN+1]; // Slip mode selection
    int fSlipProtocol;        // 0:PPP 1:SLIP
}

```

NETMAILINFO Structure

This structure stores all information for Mail data.

```

typedef struct tagNetDialInfo {
    char cMailAddressA[NMI_ADDR_LEN+1]; // User E-Mail address
    char cAccountA[NMI_ACCOUNT_LEN+1]; // POP3 account
    char cPasswordA[NMI_PASSWORD_LEN+1]; // POP3 password
    char cPop3ServerA[NMI_POP3SERVER_EN+1]; // POP3 server
    char cSmtServerA[NMI_SMTPSERVER_LEN+1]; // SMTP server
    BOOL fKeepInServer; // 1: keep in server
    int fSlipProtocol; // 0:PPP 1:SLIP
}

```

System Message

```

MT_NET_ABORTED      Executing the Net_Abort() api.
    pMsg->data.p->iResult  The result.

```

MT_NET_ACCEPTED pMsg->data.p->iResult	Executing the Net_Accept () api. The result.
MT_NET_SOCKET_CLOSED pMsg->data.p->iResult	Executing the Net_CloseSocket() api. The result.
MT_NET_CONNETED pMsg->data.p->iResult	Executing the Net_Connect() api. The result.
MT_NET_GOT_HOST_NAME pMsg->data.p->iResult	Executing the Net_GetHostByName() api. The result.
MT_NET_LISTEN pMsg->data.p->iResult	Executing the Net_Listen() api. The result.
MT_NET_RECV pMsg->data.p->iResult	Executing the Net_Receive() api. The result.
MT_NET_RECV_FROM pMsg->data.p->iResult	Executing the Net_ReceiveFrom() api. The result.
MT_NET_SELECTED pMsg->data.p->iResult	Executing the Net_Select() api. The result.
MT_NET_SENT pMsg->data.p->iResult	Executing the Net_Send() api. The result.
MT_NET_SENT_TO pMsg->data.p->iResult	Executing the Net_SendTo() api. The result.
MT_NET_PPP_MSG_INIT_MODEM	Executing the Net_Dial() api, and at the initialing modem state.
MT_NET_PPP_MSG_DIALING	Executing the Net_Dial() api, and at the dialing the phone state.
MT_NET_PPP_MSG_CHK_USN_PSW	Executing the Net_Dial() api, and at the checking username and password.
MT_NET_PPP_MSG_HANGUP	Executing the Net_Dial() api, and at the handing up the phone state.
MT_NET_DIAL_DONE pMsg->data.p->iResult	Executing the Net_Dial() api, and the whole dial process has completed. The result.
MT_NET_POP3_LOGGED_ON pMsg->data.p->iResult	Executing the Net_Pop3Logon() api. The result.
MT_NET_POP3_LOGGED_OFF pMsg->data.p->iResult	Executing the Net_Pop3Logoff() api. The result.
MT_NET_POP3_NOOP pMsg->data.p->iResult	Executing the Net_Pop3Noop() api. The result.
MT_NET_POP3_DELET pMsg->data.p->iResult	Executing the Net_Pop3Delete() api. The result.
MT_NET_POP3_RESET pMsg->data.p->iResult	Executing the Net_Pop3Reset() api. The result.
MT_NET_POP3_COMMIT	Executing the Net_Pop3Commit() api.

pMsg->data.p->iResult	The result.
MT_NET_POP3_CLOSE pMsg->data.p->iResult	Executing the Net_Pop3Close() api. The result.
MT_NET_POP3_STATE pMsg->data.p->iResult	Executing the Net_Pop3State() api. The result.
MT_NET_POP3_LIST pMsg->data.p->iResult	Executing the Net_Pop3List() api. The result.
MT_NET_POP3_UIDL pMsg->data.p->iResult	Executing the Net_Pop3Uidl() api. The result.
MT_NET_POP3_RETRIVE_MSG pMsg->data.p->iResult	Executing the Net_Pop3RetrieveMsg() api. The result.
MT_NET_POP3_TOP pMsg->data.p->iResult	Executing the Net_Pop3Top() api. The result.
MT_NET_SMTP_START pMsg->data.p->iResult	Executing the Net_SmtpStart() api. The result.
MT_NET_SMTP_QUIT pMsg->data.p->iResult	Executing the Net_SmtpQuit() api. The result.
MT_NET_SMTP_NOOP pMsg->data.p->iResult	Executing the Net_SmtpNoop() api. The result.
MT_NET_SMTP_RESET pMsg->data.p->iResult	Executing the Net_SmtpReset() api. The result.
MT_NET_SMTP_CLOSE pMsg->data.p->iResult	Executing the Net_SmtpClose() api. The result.
MT_NET_SMTP_HELLO pMsg->data.p->iResult	Executing the Net_SmtpHello() api. The result.
MT_NET_SMTP_HELP pMsg->data.p->iResult	Executing the Net_SmtpHelp() api. The result.
MT_NET_SMTP_MAIL pMsg->data.p->iResult	Executing the Net_SmtpMail() api. The result.
MT_NET_SMTP_RCPT pMsg->data.p->iResult	Executing the Net_SmtpRcpt() api. The result.
MT_NET_SMTP_DATA pMsg->data.p->iResult	Executing the Net_SmtpData() api. The result.
MT_NET_SMTP_MESSAGE pMsg->data.p->iResult	Executing the Net_SmtpMessage() api. The result.
MT_NET_SMTP_VERIFY pMsg->data.p->iResult	Executing the Net_SmtpVerify() api. The result.
MT_NET_SMTP_EXPN pMsg->data.p->iResult	Executing the Net_SmtpExpn() api. The result.

NetAbort

Synopsis

```
int NetAbort(WORD socket);
```

Description

Closes TCP or UDP connection.

socket Specifies the ID of Socket.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be NET_INVALID_SOCKET or NET_NO_PORT_NUMBER.

NetAccept

Synopsis

```
int NetAccept(WORD socket, NETADDR *netAddr, WORD *addrLen);
```

Description

Establishes a new Socket which contains information of server and client.

socket Specifies the ID of the socket.
**netAddr* Specifies the structure NETADDR.
**addrLen* Specifies the length of the structure NETADDR.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be NET_INVALID_SOCKET, NET_NO_TASK_MATCH, NET_INVALID_PROTOCOL, NET_NO SOCK_MEMORY, NET_NO_SOCKET_SPACE, or NET_NO_PORT_NUMBER.

NetAddRoute

Synopsis

```
int NetAddRoute(BYTE *ipAddr, BYTE *subMask, BYTE *gateway);
```

Description

Adds the IP address of gateway into routing table

**ipAddr* Specifies the destination IP address which is to be added to the route. There are three situations:
 1. It can be the IP address of a host.
 2. It can be the IP address of a computer network.
 3. It can be the IP address of all 0's.
**subMask* Specifies the mask used by the route.
**gateway* Specifies the IP address of the gateway or the Next hop.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be NET_INVALID_PARM.

NetBind

Synopsis

```
int NetBind(WORD socket, NETADDR *netAddr);
```

Description

Allocates a new area address to socket. It has to be called by the client when it is in connection-oriented or wireless transfer mode. But the client only needs to call in wireless transfer mode.

socket Specifies the ID of the socket.
**netAddr* Specifies the structure NETADDR containing the remote client's address

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is -1.

NetCheckFD

Synopsis

```
int NetCheckFD(long socket, NETFD *bitMap);
```

Description

Checks if bitmap has been set up. It is usually used in NetSelect to check the returned value of the bitmap.

socket Specifies the ID of Socket.
**bitMap* Specifies the NETFD structure.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value.

NetCloseSocket

Synopsis

```
int NetCloseSocket(WORD socket);
```

Description

Closes the socket. There are two things to remind when closing socket connections:

1. If TCP connection is used, the service of the function call will be paused until the activity of closing connection finishes.
2. If UDP is used, then the Socket will be closed directly.

socket Specifies the ID of socket.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be NET_INVALID_SOCKET or NET_NO_PORT_NUMBER.

NetConnect

Synopsis

```
int NetConnect(WORD socket, NETADDR *netAddr);
```

Description

Establishes connection. Only clients need to call this function. When in connection-oriented transfer mode, client and server have to exchange messages and data; therefore, you have to establish connection. But because connection-oriented transfer needs to be run in TCP, only NetSend and NetReceive can be used.

socket Specifies the ID of Socket.
**netAddr* Specifies the structure NETADDR.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is -1.

NetDial

Synopsis

```
int NetDial(NETDIALINFO *dailInfo, void *(DrawMsg(char msg)));
```

Description

Prompts dialing message.

**dailInfo* Specifies the structure NETDIALINFO.
**(DrawMsg(char msg))*
 Specifies the callback routine of dialing message box. The following are possible values of *msg*:
 NET_PPP_MSG_INIT_MODEM
 NET_PPP_MSG_DIALING
 NET_PPP_MSG_CHK_USN_PSW
 NET_PPP_MSG_HANGUP

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is -1.

NetGetDialInfo

Synopsis

```
BOOL NetGetDialInfo(NETDIALINFO *dailInfo);
```

Description

Obtains the data of the dial setting from system.

**dailInfo* Specifies the structure NETDIALINFO to which the dial setting is coping.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

NetGetHostByAddr

Synopsis

```
int NetGetHostByAddr(char *ipAddr, NETHOST *host);
```

Description

Obtains the structure NETHOST by using IP address.

**ipAddr* Specifies host's IP address to search for.
**host* Specifies the NETHOST structure to which the data of host name is copied.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be NET_INVALID_PARM or NET_NOT_A_HOST.

NetGetHostByName

Synopsis

```
int NetGetHostByName(char *ipAddr, NETHOST *host);
```

Description

Obtains IP address from the other side by using the socket connection.

**ipAddr* Specifies host's IP address to which the data of IP address is copied.
**host* Specifies the NETHOST structure storing the host data to search for.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be NET_BAD_SOCKETD, NET_NOT_CONNECTED, or NET_INVALID_PARM.

NetGetMailInfo

Synopsis

```
BOOL NetGetMailInfo(NETMAILINFO *mailInfo);
```

Description

Obtains the data of the mailbox setting from system.

**mailInfo* Specifies the structure NETMAILINFO to which the mailbox setting is copied.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

NetGetPeerName

Synopsis

```
int NetGetPeerName(WORD socket, NETSOCKETADDR *socketAddr, WORD *addrLen);
```

Description

Obtains the data from the other side by using the socket connection.

socket Specifies the ID of Socket.
**socketAddr* Specifies the NETSOCKETADDR structure to which the remote client's address is copied.
**addrLen* Specifies the length of the NETSOCKETADDR structure.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be NET_BAD_SOCKETD, NET_NOT_CONNECTED, or NET_INVALID_PARM.

NetGetSocketOption

Synopsis

```
int NetGetSocketOption(WORD socket, long level, long optName, void *optVal,  
long *optValLen);
```

Description

Obtains the option of the Socket.

socket Specifies the ID of Socket.
level Specifies protocol level. The following are possible values:
SOL_SOCKET socket option
IPPROTO_IP Ipv 4 option
optName Specifies the designated option.
**optVal* Specifies the new value of the option.
**optValLen* Specifies the size of the area pointed by the pointer *optVal*.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be NET_INVALID_LEVEL, NET_INVALID_OPTION, NET_INVAL, NET_ACCESS, NET_ADDRINUSE, NET_HOST_UNREACHABLE, NET_MSGSIZE, NET_NOBUFS, NET_UNRESOLVED_ADDR, NET_CLOSING, NET_MEM_ALLOC, or NET_RESET.

NetGetIpAddr

Synopsis

```
int NetGetIpAddr(NETIOCTL *netIOCtrl, long netIOCtrlLen);
```

Description

Obtains the IP address of the interface of physical layer.

**netIOCtrl* Specifies the NETIOCTL structure.to which the information of IP address is copied.
IOCtrlLen Specifies the length of the NETIOCTL structure.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value,

which can be NET_INVALID, NET_INVALID_PARM, or NET_INVALID_OPTION.

NetHangUp

Synopsis

```
void NetHangUp(void);
```

Description

Hangs up connection.

Returned Value

None.

NetInitFD

Synopsis

```
void NetInitFD(NETFD *bitMap);
```

Description

Initializes Bitmap.

**bitMap* Specifies the NETFD structure.

Returned Value

None.

NetIsConnected

Synopsis

```
int NetIsConnected(WORD socket);
```

Description

Checks if the connection has been established. Only TCP can be used.

socket Specifies the ID of Socket.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be NET_INVALID_SOCKET.

NetListen

Synopsis

```
int NetListen(WORD netSntp, WORD backLog);
```

Description

Waits for connection request. This function is called by server when connection-oriented transfer is

being established.

netSmtp Specifies the ID of the socket.
backLog Specifies number of requests, which can be queued.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be NET_INVALID_SOCKET, NET_NOT_A_TASK, and NET_NO SOCK_MEMORY.

NetPop3Close

Synopsis

```
int NetPop3Close (NETPOP3 *netPop3) ;
```

Description

Stops POP3 server, but doesn't delete the data on the server.

**netPop3* Specifies the NETPOP3 structure.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value

NetPop3Commit

Synopsis

```
int NetPop3Commit (NETPOP3 *netPop3) ;
```

Description

Executes the deletion according to the flag of NetPop3Delete.

**netPop3* Specifies the NETPOP3 structure.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be NET_POP3_STACK_ERROR, NET_POP3_COMMAND_ERROR or NET_POP3_INVALID_STATE.

NetPop3Delete

Synopsis

```
int NetPop3Delete (NETPOP3 *netPop3, long msgNum) ;
```

Description

Sets the delete flag of a record in POP3 server.

**netPop3* Specifies the NETPOP3 structure.
msgNum Specifies the data number to be deleted.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be NET_POP3_STACK_ERROR, NET_POP3_COMMAND_ERROR, NET_POP3_INVALID_CLIENT, or NET_POP3_INVALID_STATE.

NetPop3List

Synopsis

```
int NetPop3List (NETPOP3 *netPop3, long msgNo);
```

Description

Obtains the length of the mail from the server and list it. The mail to get is according to the msg parameters. If msg equals 0, then it will list the length of all the mails.

**netPop3* Specifies the NETPOP3 structure.
msgNo Specifies the mail number.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be NET_POP3_STACK_ERROR, NET_POP3_COMMAND_ERROR, NET_POP3_NO_LIST, or NET_POP3_INVALID_STATE.

NetPop3Logoff

Synopsis

```
int NetPop3Logoff (NETPOP3 *netPop3);
```

Description

Logs off and closes a session with a POP3 server.

**netPop3* Specifies the NETPOP3 structure.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be NET_POP3_STACK_ERROR, or NET_POP3_COMMAND_ERROR

NetPop3Logon

Synopsis

```
int NetPop3Logon (NETPOP3 *netPop3);
```

Description

Logs onto a POP3 server.

**netPop3* Specifies the NETPOP3 structure.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be NET_POP3_STACK_ERROR, or NET_POP3_INVALID_USER.

NetPop3Noop

Synopsis

```
int NetPop3Noop (NETPOP3 *netPop3);
```

Description

Sends out a noop command.

**netPop3* Specifies the NETPOP3 structure.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be NET_POP3_STACK_ERROR, NET_POP3_COMMAND_ERROR, or NET_POP3_INVALID_STATE.

NetPop3Reset

Synopsis

```
int NetPop3Reset (NETPOP3 *netPop3);
```

Description

Resets the POP3 server.

**netPop3* Specifies the NETPOP3 structure.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be NET_POP3_STACK_ERROR, NET_POP3_COMMAND_ERROR or NET_POP3_INVALID_STATE.

NetPop3RetrieveCallback

Synopsis

```
void NetPop3RetrieveCallback(int *timeOut(void));
```

Description

Registers a callback function of POP3 receiver.

**timeOut* Specifies the callback function for the POP3 receiver. The following are possible returned values of this callback function:

1	Keep waiting
0	Abort

Returned Value

None.

NetPop3RetrieveMsg

Synopsis

```
int NetPop3RetrieveMsg (NETPOP3 *netPop3, long msgNo);
```

Description

Obtains the message from the POP3 server.

**netPop3* Specifies the structure NETPOP3.
msgNo Specifies the number of the message. If it equals 0, then all of the messages will be got from the server.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be NET_POP3_STACK_ERROR, NET_POP3_INVALID_MSG, NET_POP3_COMMAND_ERROR, NET_POP3_NO_WRITE, or NET_POP3_INVALID_STATE

NetPop3State

Synopsis

```
NETPOP3STATE *NetPop3State (NETPOP3 *netPop3);
```

Description

Obtains the status of the POP3 server.

**netPop3* Specifies the NETPOP3 structure.

Returned Value

The returned value is pointer of NETPOP3STATE if this function succeeds. Otherwise, the returned value is NULL pointer

NetPop3Top

Synopsis

```
int NetPop3Top (NETPOP3 *netPop3, long msgNo, long lines);
```

Description

Tests if the line of the designated mail exists.

**netPop3* Specifies the structure NETPOP3.
msgNo Specifies the number of the mail.
lines Specifies the line number to test.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be NET_POP3_STACK_ERROR, NET_POP3_INVALID_MSG, NET_POP3_COMMAND_ERROR, NET_POP3_NO_WRITE or NET_POP3_INVALID_STATE

NetPop3Uidl

Synopsis

```
int NetPop3Uidl (NETPOP3 *netPop3, long msgNo);
```

Description

Obtains the unique identifier of the designated mail from the POP3 server. If msg = 0, then it will list all of the unique identifiers of the mails in the server.

**netPop3* Specifies the NETPOP3 structure.
msgNo Specifies the mail number.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be NET_POP3_STACK_ERROR, NET_POP3_INVALID_MSG, NET_POP3_COMMAND_ERROR, NET_POP3_NO_LIST, or NET_POP3_INVALID_STATE.

NetReceive

Synopsis

```
int NetReceive (WORD socket, char *buffer, WORD bufferLen);
```

Description

Receives data.

socket Specifies the ID of the Socket.
**buffer* Specifies data buffer.
bufferLen Specifies the length of the data buffer.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be NET_INVALID_SOCKET, NET_NO_PORT_NUMBER, NET_NOT_CONNECTED, or NET_NOT_A_TASK.

NetResetFD

Synopsis

```
void NetResetFD (long socket, NETFD *bitMap);
```

Description

Resets Bitmap and set the designated bit of Bitmap as 0.

socket Specifies the ID of Socket.
**bitMap* Specifies the NETFD structure.

Returned Value

None.

NetReceiveFrom

Synopsis

```
int NetReceiveFrom (WORD socket, char *buffer, WORD bufferLen, NETADDR *netAddr);
```


Description

Receives data and obtains the data from the other side.

<i>socket</i>	Specifies the ID of Socket.
<i>*buffer</i>	Specifies data buffer to which the data is copied.
<i>bufferLen</i>	Specifies the length of the data buffer.
<i>*netAddr</i>	Specifies the structure NETADDR.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be NET_INVALID_SOCKET , NET_NO_PORT_NUMBER, or NET_NO_DATA_TRANSFER.

NetSelect

Synopsis

```
int NetSelect(long maxSockets, NETFD *bitMap, DWORD timeout);
```

Description

Waits for the online data. There are two situations in which the function will stop waiting-one is that there is data on line. The other is timeout.

<i>maxSockets</i>	Specifies the maximum number of the connected socket.
<i>*bitMap</i>	Specifies the bitmap socket to check.
<i>timeout</i>	Specifies to limit time.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be NET_NO_SOCKETS or NET_NO_DATA.

NetSend

Synopsis

```
int NetSend(WORD socket, char *buffer, WORD bufferLen);
```

Description

Transfers data.

<i>socket</i>	Specifies the ID of Socket.
<i>*buffer</i>	Specifies data buffer.
<i>bufferLen</i>	Specifies the length of the data buffer.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be NET_INVALID_SOCKET, NET_NO_PORT_NUMBER, NET_NOT_CONNECTED, or NET_NO_BUFFERS.

NetSendTo

Synopsis

```
int NetSendTo(WORD socket, char *buffer, WORD bufferLen, NETADDR *netAddr);
```

Description

Sends data and get the data from the sender.

<i>socket</i>	Specifies the ID of Socket.
* <i>buffer</i>	Specifies data buffer.
<i>bufferLen</i>	Specifies the length of the data buffer
* <i>netAddr</i>	Specifies the structure NETADDR.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be NET_INVALID_SOCKET, NET_NO_PORT_NUMBER, NET_NO_DATA_TRANSFER, or NET_INVALID_ADDRESS.

NetSetBlock

Synopsis

```
int NetSetBlock(WORD socket, BOOL flag);
```

Description

Sets the socket related block. When the client and server are waiting for the data of NetReceive, it would call this function.

<i>socket</i>	Specifies the ID of socket.
<i>flag</i>	Specifies the setting for the block flag.
TRUE	Enables blocking.
FALSE	Disables blocking.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be NET_INVALID_SOCKET or NET_NO_ACTION.

NetSetDialInfo

Synopsis

```
BOOL NetSetDialInfo(NETDIALINFO *dailInfo);
```

Description

Sets the data of the dial setting into system.

**dailInfo* Specifies the structure NETDIALINFO storing the dial setting

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value.

NetSetFD

Synopsis

```
void NetSetFD(WORD socket, NETFD *bitMap);
```

Description

Sets the bitmap. It is usually used in NetSelect to check the returned value of the bitmap.

socket Specifies the ID of Socket.
**bitMap* Specifies the NETFD structure.

Returned Value

None

NetSetMailInfo

Synopsis

```
BOOL NetSetMailInfo(NETMAILINFO *netMailInfo);
```

Description

Sets the data of the mailbox setting into system.

netMailInfo Specifies the structure NETDIALINFO storing the mailbox setting.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

NetSetSocketOption

Synopsis

```
int NetSetSocketOption(WORD socket, long level, long optName, void *optVal,  
                        long optValLen);
```

Description

Sets the options of Socket.

socket Specifies the ID of Socket.
level Specifies protocol level. The following are possible values:
SOL_SOCKET Socket option
IPPROTO_IP IPv4 option
optName Specifies the designated option.
**optVal* Specifies the new value of the option.
**optValLen* Specifies the size of the area pointed by the pointer pOptVal.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be NET_INVALID_LEVEL, NET_INVALID_OPTION, NET_INVAL, NET_ACCESS, NET_ADDRINUSE, NET_HOST_UNREACHABLE, NET_MSGSIZE, NET_NOBUFS, NET_UNRESOLVED_ADDR, NET_CLOSING, NET_MEM_ALLOC, or NET_RESET.

NetSocket

Synopsis

```
int NetSocket(WORD type) ;
```

Description

Establishes a new Socket and a protocol.

<i>type</i>	Specifies the type of communication protocol. The following are possible values:
SK_TYPE_STREAM	Stream Socket
SK_TYPE_DGRAM	Datagram Socket
SK_TYPE_RAW	Raw Socket
SK_TYPE_SEQPACKET	Sequenced packet Socket.
SK_TYPE_RDM	Reliably delivered msg Socket

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is -1.

NetSmtClose

Synopsis

```
short NetSmtClose(NETSMTP *netSmt) ;
```

Description

Closes the SMTP server connected to the socket.

**netSmt* Specifies the NETSMTP structure.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value.

NetSmtData

Synopsis

```
short NetSmtData(NETSMTP *netSmt) ;
```

Description

Tells SMTP server send the data of the client which sends email.

** netSmt* Specifies the structure NETSMTP.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value.

NetSmtExpn

Synopsis

```
short NetSmtExpn(NETSMTP *netSmt, BYTE *name) ;
```

Description

Closes the SMTP server connected to the socket.

**netSntp* Specifies the structure NETSMTP.
**name* Specifies the name of the mailing list to expand.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value.

NetSntpHello

Synopsis

```
short NetSntpHello (NETSMTP *netSntp) ;
```

Description

Sends the hello command to server. When the transmission has not been started in the client, this is the first command it will send out. This will happen once when the user logs in.

**netSntp* Specifies the NETSMTP structure.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value.

NetSntpHelp

Synopsis

```
short NetSntpHelp (NETSMTP *netSntp, short command) ;
```

Description

Sends the help request to the SMTP Server.

**netSntp* Specifies the structure NETSMTP.
command Specifies get the type of the help command from the smtp server.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value.

NetSntpMail

Synopsis

```
short NetSntpMail (NETSMTP *netSntp, BYTE *from) ;
```

Description

Starts to execute mail transfer.

**netSntp* Specifies the structure NETSMTP.
**from* Specifies the source of the e-mail.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value.

NetSntpMessage

Synopsis

```
short NetSntpMessage (NETSMTP *netSntp, BYTE *message);
```

Description

Sends message to the SMTP server.

**netSntp* Specifies the structure NETSMTP.
**message* Specifies the message data.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value.

NetSntpNoop

Synopsis

```
short NetSntpNoop (NETSMTP *netSntp);
```

Description

Sends a noop command to SMTP server.

**netSntp* Specifies the NETSMTP structure.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value.

NetSntpQuit

Synopsis

```
short NetSntpQuit (NETSMTP *netSntp);
```

Description

Quits SMTP server.

**netSntp* Specifies the NETSMTP structure.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value.

NetSntpRcpt

Synopsis

```
short NetSntpRcpt (NETSMTP *netSntp, BYTE *to);
```

Description

Sends the recipient's email address to the SMTP server.

**netSmtp* Specifies the structure NETSMTP.
**to* Specifies the target pointer.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value.

NetSmtpReset

Synopsis

```
short NetSmtpReset (NETSMTP *netSmtp) ;
```

Description

Resets SMTP client, and in the meantime clear buffer and state.

**netSmtp* Specifies the NETSMTP structure.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value.

NetSmtpStart

Synopsis

```
short NetSmtpStart (NETSMTP *netSmtp) ;
```

Description

Activates SMTP server.

**netSmtp* Specifies the NETSMTP structure

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a -1.

NetSmtpVerify

Synopsis

```
short NetSmtpVerify (NETSMTP *netSmtp, BYTE *userName) ;
```

Description

Makes SMTP server confirm that the sent user name exists on the server.

**netSmtp* Specifies the structure NETSMTP.
**userName* Specifies user e-mail address to confirm.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value.

Net_Abort

Synopsis

```
BOOL Net_Abort (WORD socket) ;
```

Description

Closes socket(stop TCP or UDP connection).

socket Specifies the ID of the socket.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Net_Accept

Synopsis

```
BOOL Net_Accept (WORD socket, NETADDR *netAddr, WORD *netAddrLen) ;
```

Description

Establishes a new socket, which contains information of server and client.

socket Specifies the ID of the socket.

**netAddr* Specifies the structure NETADDR.

**netAddrLen* Specifies the length of the structure NETADDR.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Net_CloseSocket

Synopsis

```
BOOL Net_CloseSocket (WORD socket) ;
```

Description

Closes socket connection. There are two things to remind when closing Socket connections:

1. If TCP connection is used, the service of the function call will be paused until the activity of closing connection finishes.
2. If UDP is used, then the socket will be closed directly.

This function works in background and will send MT_NET_SOCKET_CLOSED message to container.

socket Specifies the ID of Socket.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Net_Connect

Synopsis

```
BOOL Net_Connect (WORD socket, NETADDR *netAddr);
```

Description

Establishes the connection. Only clients need to call this function. When in connection-oriented transfer mode, client and server have to exchange messages and data; therefore, you have to establish the connection. But, because a connection-oriented transfer needs to be run in TCP, only Net_Send and Net_Receive can be used. This function works in background and will send MT_NET_CONNECTED message to the container.

socket Specifies the ID of Socket.
**netAddr* Specifies the structure NETADDR.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Net_Dial

Synopsis

```
BOOL Net_Dial (NETDIALINFO *dailInfo);
```

Description

Prompts a dialing message. This function works in background and will send MT_NET_DIAL_DONE message to the container.

**dailInfo* Specifies the structure NETDIALINFO storing the dial setting.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Net_GetHostByAddr

Synopsis

```
BOOL Net_GetHostByAddr (char *ipAddr, NETHOST *host);
```

Description

Obtains the structure Host by using IP address. This function works in background and will send MT_NET_GOT_HOST_NAME message to container.

**ipAddr* Specifies host's IP address to search for.
**host* Specifies the NETHOST structure to which the data of host name is copied.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Net_GetHostByName

Synopsis

```
BOOL Net_GetHostByName(char *ipAddr, NETHOST *host);
```

Description

Obtains the structure Host by using Host's name. This function works in background and will send MT_NET_GOT_HOST_NAME message to container.

**ipAddr* Specifies host's IP address to which the data of IP address is copied.
**host* Specifies the NETHOST structure storing the host data to search for.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Net_GetPop3Info

Synopsis

```
NETPOP3 *Net_GetPop3Info(void);
```

Description

Obtains POP3 information.

Returned Value

The returned value is a pointer of NETPOP3.

Net_GetSmtpInfo

Synopsis

```
NETSMTP *Net_GetSmtpInfo(void);
```

Description

Obtains SMTP information.

Returned Value

The returned value is a pointer of SMTP.

Net_Listen

Synopsis

```
BOOL Net_Listen(WORD netSmtp, WORD backLog);
```

Description

Waits for connection request. Called by server when connection-oriented transfer is being established. This function works in background and will send MT_NET_LISTEN message to container.

netSmtp Specifies the ID of the socket.
backLog Specifies the number of requests, which can be queued.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Net_Pop3Close

Synopsis

```
BOOL Net_Pop3Close(void);
```

Description

Logs off POP3 server without deleting marked messages. This function works in background and will send MT_NET_POP3_CLOSE message to container.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Net_Pop3Commit

Synopsis

```
BOOL Net_Pop3Commit(void);
```

Description

Execute the deletion according to the flag of Net_Pop3Delete. This function works in background and will send MT_NET_POP3_COMMIT message to container.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Net_Pop3Delete

Synopsis

```
BOOL Net_Pop3Delete(long msgNum);
```

Description

Sets the delete flag of a record in POP3 server. This function works in background and will send MT_NET_POP3_DELET message to container.

msgNum Specifies the data number to delete.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Net_Pop3List

Synopsis

```
BOOL Net_Pop3List(long msgNo);
```

Description

Obtains the length of the mail from the server and list it. The mail to get is according to the msg parameter. If msg equals 0, then it will list the length of all the mails. This function works in background and will send MT_NET_POP3_LIST message to container.

msgNo Specifies the mail number.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Net_Pop3Logoff

Synopsis

```
BOOL Net_Pop3Logoff(void);
```

Description

Logs off and closing a session with a POP3 server. This function works in background and will send MT_NET_POP3_LOGGED_OFF message to container.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Net_Pop3Logon

Synopsis

```
BOOL Net_Pop3Logon(void);
```

Description

Logs onto a POP3 server. This function works in background and will send MT_NET_POP3_LOGGED_ON message to container.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Net_Pop3Noop

Synopsis

```
BOOL Net_Pop3Noop(void);
```

Description

Sends out a Noop Command. This function works in background and will send MT_NET_POP3_NOOP message to container.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Net_Pop3Reset

Synopsis

```
BOOL Net_Pop3Reset(void);
```

Description

Resets POP3 server. This function works in background and will send MT_NET_POP3_RESET message to container.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Net_Pop3RetrieveMsg

Synopsis

```
BOOL Net_Pop3RetrieveMsg(long msgNo);
```

Description

Obtains the message from the POP3 server. This function works in background and will send MT_NET_POP3_RETRIVE_MSG message to container.

msgNo Specifies the number of the message. If it equals 0, then all of the messages will be got from the server.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Net_Pop3Top

Synopsis

```
BOOL Net_Pop3Top(long msgNo, long lines);
```

Description

Tests if the line of the designated mail exists. This function works in background and will send MT_NET_POP3_TOP message to container.

msgNo Specifies the number of the mail.
lines Specifies the line number to test.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Net_Pop3State

Synopsis

```
BOOL Net_Pop3State(void);
```

Description

Obtains the status of the POP3 server. This function works in background and will send

MT_NET_POP3_STATE message to container.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Net_Pop3Uidl

Synopsis

```
BOOL Net_Pop3Uidl(long msgNo);
```

Description

Obtains the unique identifier of the designated mail from the POP3 server. If *msg* equals 0, then it will list all of the unique identifiers of the mails in the server. This function works in background and will send MT_NET_POP3_UIDL message to container.

msgNo Specifies the mail number.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Net_Receive

Synopsis

```
BOOL Net_Receive(WORD socket, char *buffer, WORD bufferLen);
```

Description

Receives data.

socket Specifies the ID of the Socket.
**buffer* Specifies data buffer.
bufferLen Specifies the length of the data buffer.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Net_ReceiveFrom

Synopsis

```
BOOL Net_Listen(WORD netSntp, WORD backLog);
```

Description

Receives data and obtains the data from the other side. This function works in background and will send MT_NET_RECV_FROM message to container.

netSntp Specifies the ID of the socket.
backLog Specifies the number of requests, which can be queued.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Net_Select

Synopsis

```
BOOL Net_Select(long maxSockets, NETFD *bitMap, DWORD timeout);
```

Description

Waits for the online data. There are two situations in which the function will stop waiting-one is that there is data on line. The other is timeout. This function works in background and will send MT_NET_SELECTED message to container.

maxSockets Specifies the maximum number of the connected socket.
**bitMap* Specifies the bitmap socket to check.
timeout Specifies to limit time.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Net_Send

Synopsis

```
BOOL Net_Send(WORD socket, char *buffer, WORD bufferLen);
```

Description

Transfers data. This function works in background and will send MT_NET_SENT message to container.

socket Specifies the ID of Socket.
**buffer* Specifies data buffer.
bufferLen Specifies the length of the data buffer.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Net_SendTo

Synopsis

```
BOOL Net_SendTo(WORD socket, char *buffer, WORD bufferLen, NETADDR *netAddr);
```

Description

Sends data and obtains the data from the sender. This function works in background and will send MT_NET_SENT_TO message to container.

socket Specifies the ID of Socket.
**buffer* Specifies data buffer.
bufferLen Specifies the length of the data buffer
**netAddr* Specifies the structure NETADDR.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Net_SmtpClose

Synopsis

```
BOOL Net_SmtpClose(void);
```

Description

Closes the SMTP server connected to the socket. This function works in background and will send MT_NET_SMTP_CLOSE message to container.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Net_SmtpData

Synopsis

```
BOOL Net_SmtpData(void);
```

Description

Tells SMTP server send the data of the client which sends email. This function works in background and will send MT_NET_SMTP_DATA message to container.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Net_SmtpExpn

Synopsis

```
BOOL Net_SmtpExpn(BYTE *name);
```

Description

Sends a Command to SMTP server, to make SMTP expand the name of the mailing list. This function works in background and will send MT_NET_SMTP_EXPAN message to container.

**name* Specifies the name of the mailing list to expand.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Net_SmtpHello

Synopsis

```
BOOL Net_SmtpHello(void);
```

Description

Sends hello command to server. When the transmission has not been started in the client, this is

the first command it will send out. This will happen once when the user logs in. This function works in background and will send MT_NET_SMTP_HELLO message to container.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Net_SmtpHelp

Synopsis

```
BOOL Net_SmtpHelp(short command);
```

Description

Sends the Help request to the SMTP Server. This function works in background and will send MT_NET_SMTP_HELP message to container.

command Specifies the type of the help command from the SMTP server.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Net_SmtpMail

Synopsis

```
BOOL Net_SmtpMail(BYTE *mailSrc);
```

Description

Starts to execute mail transfer. This function works in background and will send MT_NET_SMTP_MAIL message to container.

**mailSrc* Specifies the source of the e-mail.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Net_SmtpMessage

Synopsis

```
BOOL Net_SmtpMessage(BYTE *message);
```

Description

Sends message to the SMTP server. This function works in background and will send MT_NET_SMTP_MESSAGE message to container.

**message* Specifies the message data.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Net_SmtpNoop

Synopsis

```
BOOL Net_SmtpNoop(void);
```

Description

Sends a Noop Command to SMTP server. This function works in background and will send MT_NET_SMTP_NOOP message to container.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Net_SmtpRcpt

Synopsis

```
BOOL Net_SmtpRcpt(BYTE *to);
```

Description

Resets SMTP Client, and in the meantime clear buffer and state. This function works in background and will send MT_NET_SMTP_RESET message to container.

**to* Specifies the target pointer.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Net_SmtpStart

Synopsis

```
BOOL Net_SmtpStart(void);
```

Description

Activates SMTP server. This function works in background and will send MT_NET_SMTP_START message to container.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Net_SmtpQuit

Synopsis

```
BOOL Net_SmtpQuit(void);
```

Description

Quits from SMTP server. This function works in background and will send MT_NET_SMTP_QUIT message to container.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Net_SmtpReset

Synopsis

```
BOOL Net_SmtpReset(void);
```

Description

Resets SMTP Client, and in the meantime clear buffer and state. This function works in background and will send MT_NET_SMTP_RESET message to container.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Net_SmtpVerify

Synopsis

```
BOOL Net_SmtpVerify(BYTE *userName);
```

Description

Makes SMTP server confirm that the sent user name exists on the server. This function works in background and will send MT_NET_SMTP_VERIFY message to container.

**userName* Specifies user e-mail address to confirm.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

System Resource

In this chapter, some functions are introduced to access system resource.

Container

General Description

Retrieves system containers.

Related Data Structure

None.

System Message

MT_SYS_CONT_BEGIN	Container begins.
MT_SYS_CONT_END	Container ends.
MT_SYS_POP_CONT_BEGIN	Popup container begins.
MT_SYS_CONT_BEHIND	Before popup container begins, the container running at the background gets this message.
MT_SYS_CONT_UPDATE	After popup container ends, the container running at the background gets this message.

ChangeActiveContainer

Synopsis

```
BOOL ChangeActiveContainer(WORD id);
```

Description

Replaces the active container with another one.

id Specifies the unique ID number of container.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

ContAddObj

Synopsis

```
int ContAddObj(void *object);
```

Description

Adds objects to the container.

**object* Specifies an object, such as CButton or CTable.

Returned Value

The returned value is the index number of the object in the container if this function succeeds. Otherwise, the returned value is -1.

ContGetObj

Synopsis

```
void *ContGetObj(WORD id);
```

Description

Obtains an object from the container.

id Specifies the unique ID number of the object, such as CButton or CTable.

Returned Value

The returned value is a pointer of an object if this function succeeds. Otherwise, the returned value is NULL.

ContRedraw

Synopsis

```
void ContRedraw(void);
```

Description

Repaints the container.

Returned Value

None.

ContRemoveObj

Synopsis

```
BOOL ContRemoveObj(WORD id);
```

Description

Removes an object from the container.

id Specifies the unique ID number of the object, such as CButton or CTable.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

ContSetTitle

Synopsis

```
void ContSetTitle(char *title);
```

Description

Sets the title of the container.

**title* Specifies the container title.

Returned Value

None.

EnterCriticalSection

Synopsis

```
void EnterCriticalSection(void);
```

Description

Disables the ALARM and MT_BATTERY_LOW message and the conflict of different messages on screen.

Returned Value

None.

HookContMsg

Synopsis

```
SysCallback HookContMsg(SysCallback hookProc);
```

Description

Allows an application to add a callback function to container. This function is invoked before system messages being sent to container. Then user can use *hookProc* to intercept system messages and have some handle. When an application starts running, *hookProc* doesn't exist. *hookProc* will be removed from OS no matter if there is a *hookProc* when this application ends. If the parameter *hookProc* equals 0, the callback function will be removed.

hookProc Specifies the name of the callback function of container.

Returned Value

The returned value is previous callback function have been set.

HookPopContMsg

Synopsis

```
SysCallback HookPopContMsg(SysCallback hookProc);
```

Description

Uses this function to allow AP to add a Callback Function to pop container. Before the system sends a message to the pop container, it will call this Callback Function. Then HookProc has a chance to intercept message, and handle it.

hookProc Specifies the function which can handle the message.

Returned Value

The returned value is previous callback function have been set.

IsPopContActive

Synopsis

```
BOOL IsPopContActive(void);
```

Description

Obtains the active status of a pop up container.

Returned Value

The returned value is TRUE if the pop up container is active. Otherwise, the returned value is FALSE.

LeaveCriticalSection

Synopsis

```
void LeaveCriticalSection(void);
```

Description

Enables the ALARM and MT_BATTERY_LOW message which are disabled by EnterCriticalSection ().

Returned Value

None.

PopContainer

Synopsis

```
int PopContainer(SysCallback entry, const char *title, void *data);
```

Description

Pops out a new container.

<i>entry</i>	Specifies the name of the registered (callback) function of container.
* <i>title</i>	Specifies the title to be displayed.
* <i>data</i>	Specifies the data in the container to be displayed.

Returned Value

The returned value is a non-zero value if this function succeeds. Otherwise, the returned value is zero.

PopContainerEnd

Synopsis

```
void PopContainerEnd(int retValue);
```

Description

Terminates a pop up container.

<i>retValue</i>	Specifies the returned value of PopContainer().
-----------------	---

Returned Value

None.

SetActiveContainer

Synopsis

```
BOOL SetActiveContainer(WORD id);
```

Description

Sets the container to be executed. This function switches the active container from the current to a new one.

<i>id</i>	Specifies the unique ID number of a container.
-----------	--

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

SetContainerEntry

Synopsis

```
BOOL SetContainerEntry(WORD id, SysCallback entry);
```

Description

Sets the entry point of the container; that is, to register a container.

<i>id</i>	Specifies the unique ID number of a container.
<i>entry</i>	Specifies the name of the registered (callback) function of container.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

SetTabContEntry

Synopsis

```
BOOL SetTabContEntry(WORD id, SysCallback entry, const char *tag);
```

Description

Sets the entry point of the tab container. This function sets the active container for each tab menu. After the user click on the specific tab menu, the associated container is active.

<i>id</i>	Specifies the unique ID number of a container.
<i>entry</i>	Specifies the name of the registered (callback) function of container.
<i>*tag</i>	Specifies the name string, which will be shown in the tag.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

SetTitleTabContEntry

Synopsis

```
BOOL SetTitleTabContEntry(WORD id, SysCallback entry, const char *tag);
```

Description

Sets the entry point of the container of the form TitleTable; that is, to register a container of TitleTable.

<i>id</i>	Specifies the unique ID number of a container.
<i>entry</i>	Specifies the name of the registered (callback) function of container.
<i>*tag</i>	Specifies the name string, which will be shown in the tag.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Timer

General Description

Retrieves system time.

Related Data Structure

None.

System Message

MT_TIMER	After the number of real-time specified in the Time has elapsed.
MT_SLEEP	When system is going to the sleep mode.
MT_WAKEUP	When system is going to wake up.

DisableTimerMsg

Synopsis

```
void DisableTimerMsg(void);
```

Description

Disables the Timer.

Returned Value

None.

EnableAutoSleep

Synopsis

```
int EnableAutoSleep(BOOL isEnabled);
```

Description

Enables or disables auto-sleep.

<i>isEnabled</i>	Specifies the enabling flag.
TRUE	Enable
FALSE	Disable

Returned Value

The returned value is the previous enabling status.

- 1 The previous status is enabled.
- 0 The previous status is disabled.

EnableTimerMsg

Synopsis

```
void EnableTimerMsg(void);
```

Description

Enables the Timer.

Returned Value

None.

GetAutoSleepTime

Synopsis

```
DWORD GetAutoSleepTime(void);
```

Description

Obtains the sleeping time interval of the device. When the device is in idle mode for a specific sleep interval, it will be going to sleep mode.

Returned Value

The returned value is sleeping time interval in seconds.

IsTimerMsgOn

Synopsis

```
BOOL IsTimerMsgOn(void);
```

Description

Obtains the status of the Timer.

Returned Value

The returned value is the status of the Timer.

- 0 The Timer is enabled.
- 1 The Timer is disabled.

SetAutoSleepTime

Synopsis

```
DWORD SetAutoSleepTime(DWORD interval);
```

Description

Obtains the sleeping time interval of the device. When the device was in idle mode for a specific sleep interval, it will be going to sleep mode.

interval Specifies the auto-sleep interval in second.

Returned Value

The returned value is the previous time interval which has been set.

SetTimerMsg

Synopsis

```
DWORD SetTimerMsg(DWORD interval);
```

Description

Sets the time interval of timer.

interval Specifies the interval between each MT_TIMER message is send out, in milliseconds.

Returned Value

The returned value is the time interval which has been set.

Sleeping

Synopsis

```
void Sleeping(DWORD interval);
```

Description

Sets the sleeping time interval of the system. After executing this function, system will not be able to receive any message for a certain period of time.

interval Specifies the sleeping time interval in milliseconds.

Returned Value

None.

Message

General Description

Retrieves system message.

Related Data Structure

SysMsg Structure

System Message

MT_HOME	Presses the HOME soft key on the pad.
MT_MENU	Presses the MENU soft key on the pad.
MT_KEYBOARD	Presses the KEYBOARD soft key on the pad.
MT_BACK	Presses the BACK soft key on the pad.
MT_HELP	Presses the HELP soft key on the pad.
MT_SEARCH	Presses the SEARCH soft key on the pad.

AppProcess

Synopsis

```
void AppProcess(SysMsg *msg);
```

Description

Processes the message of application program. The sequence of process steps is as below:

1. Process the message PRINTPANEL.
2. Process the message of containerEntry.
3. Process the message of external key (left, right, up or down key).
4. Process the message of Tab Container.
5. Process the message HOME.

**msg* Specifies the structure SysMsg.

Returned Value

None.

GetMsg

Synopsis

```
BOOL GetMsg(SysMsg *msg);
```

Description

Obtains the system message event.

**msg* Specifies the structure SysMsg.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

GetMsgNoWait

Synopsis

```
BOOL GetMsgNoWait(SysMsg *msg);
```

Description

Obtains the message from the message queue. The function returns the system message immediately without waiting for finishing the current process.

**msg* Specifies the structure SysMsg.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

SendMsg

Synopsis

```
BOOL SendMsg (SysMsg *msg) ;
```

Description

Sends a message to system.

**msg* Specifies the structure SysMsg.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

SendMsgTop

Synopsis

```
BOOL SendMsgTop (SysMsg *msg) ;
```

Description

Sends the message into the mssege queue with top priority.

**msg* Specifies the structure SysMsg.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

SysProcess

Synopsis

```
BOOL SysProcess (SysMsg *msg) ;
```

Description

Obtains important system messages processed by system. The possible values are in the following:

```
MT_KERNEL_CHANGE_CONT  
MT_KERNEL_SET_CONT  
MT_KERNEL_CHANGE_AP  
MT_KERNEL_SLEEP  
MT_KERNEL_POWER_OFF  
MT_KERNEL_BATTERY_LOW  
MT_KERNEL_NO_ACT_CONT  
MT_KERNEL_ALARM
```

**msg* Specifies the structure SysMsg.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Key

General Description

Obtains the action of hardware keys.

Related Data Structure

None.

System Message

MT_KEY_UP The external key is pressed
 pMsg->id Current combination of pressing keys.

MT_KEY_DOWN The external key is released.
 pMsg->id Current combination of pressing keys.

MT_KEY_REPEAT The external key is holded.
 pMsg->id Current combination of pressing keys.

EnableKeyRepeatMsg

Synopsis

```
void EnableKeyRepeatMsg(int isEnabled);
```

Description

Enables or disables the message of key repeat. The default value is enabled.

<i>isEnabled</i>	Specifies the enable or disable option
0	Disable the message.
1	Enable the message.

Returned Value

None.

GetCurrentKeyState

Synopsis

```
WORD GetCurrentKeyState(void);
```

Description

Obtains the current pressed key.

Returned Value

The returned value is the value of the previously pressed key.

SetKeyRepeatTick

Synopsis

```
DWORD SetKeyRepeatTick(DWORD ticks);
```

Description

Sets the key-repeat interval in ticks (1 tick = 10 ms). If a key is pressed and hold over this interval, it will be taken as pressed two or more times. Setting this interval to 9 is to set the default value.

<i>ticks</i>	Specifies the key-repeat interval in ticks.
--------------	---

Returned Value

The returned value is the previous key-repeat interval.

Pen

General Description

Obtains the action of touch pen.

Related Data Structure

None.

System Message

MT_PEN_UP	The pen leaves the touch panel.
pMsg->data.pos.x	The x coordinates of the pen position.
pMsg->data.pos.y	The y coordinates of the pen position.
MT_PEN_DOWN	The pen touches the touch panel.
pMsg->data.pos.x	The x coordinates of the pen position.
pMsg->data.pos.y	The y coordinates of the pen position.
MT_PEN_MOVE	The pen moves on the touch panel.
pMsg->data.pos.x	The x coordinates of the pen position.
pMsg->data.pos.y	The y coordinates of the pen position.
MT_PEN_REPEAT	The pen holds on the touch panel, and don't move.
pMsg->data.pos.x	The x coordinates of the pen position.
pMsg->data.pos.y	The y coordinates of the pen position.
MT_PEN_DBCLICK	The pen double clicks on the touch panel.
pMsg->data.pos.x	The x coordinates of the pen position.
pMsg->data.pos.y	The y coordinates of the pen position.
MT_PEN_CALIBRATE_OK	The pen calibration process has been finished.
pMsg->data.pos.x	The x coordinates of the pen position.
pMsg->data.pos.y	The y coordinates of the pen position.

Calibration

Synopsis

```
void Calibration(BOOL isShow, void (*fnCaliStr)(int spot));
```

Description

Calibrates the pen position.

isShow Specifies the flag to display the coordinates while calibrating.
 TRUE Display the coordinates of the pen
 FALSE Does not display the coordinates of the pen

**fnCaliStr*(int *spot*) Specifies the callback function. This callback function is to display the prompt string at the specified position when calibrating the pen.

spot Specifies a number (range from 1 to 3).

Returned Value

None.

EnablePenDoubleClick

Synopsis

```
void EnablePenDoubleClick(int isEnabled);
```

Description

Sets the setting of the double-click message function

isEnabled Specifies the setting of PEN_DOUBLE_CLICK message function.
 1 Enable the function to send the PEN_DOUBLE_CLICK message
 0 Disable the function to send the PEN_DOUBLE_CLICK message

Returned Value

None.

EnablePenMoveMsg

Synopsis

```
void EnablePenMoveMsg(int isEnabled);
```

Description

Sets the setting of the PEN_MOVE message function.

isEnabled Specifies the setting of MT_PEN_MOVE Message function.
 1 Enable the function to send the MT_PEN_MOVE message.
 0 Disable the function to send the MT_PEN_MOVE message.

Returned Value

None.

EnablePenRepeatMsg

Synopsis

```
void EnablePenRepeatMsg(int isEnabled);
```

Description

Sets the setting of the PEN_REPEAT message function

<i>isEnabled</i>	Specifies the setting of MT_PEN_REPEAT Message function.
1	Enable the function to send the MT_PEN_REPEAT message.
0	Disable the function to send the MT_PEN_REPEAT message.

Returned Value

None.

SetDoubleClickTick

Synopsis

```
WORD SetDoubleClickTick(WORD ticks);
```

Description

Sets the double-click interval in ticks (1 tick = 10 ms). If a pen is pressed and hold over this interval, it will be taken as double clicked. Setting this interval to 0 is to set the default value.

<i>ticks</i>	Specifies the double-click interval in ticks.
--------------	---

Returned Value

The returned value is the previous pen-repeat interval.

SetPenMoveTick

Synopsis

```
DWORD SetPenMoveTick(DWORD ticks);
```

Description

Sets the key-move interval in ticks (1 tick = 10 ms). If a key is pressed and hold over this interval, it will be taken as pressed two or more times. Setting this interval to 0 is to set the default value.

<i>ticks</i>	Specifies the key-move interval in ticks.
--------------	---

Returned Value

The returned value is the previous key-move interval.

SetPenRepeatTick

Synopsis

```
DWORD SetPenRepeatTick(DWORD ticks);
```

Description

Sets the pen-repeat interval in ticks (1 tick = 10 ms). If a pen is pressed and hold over this interval, it will be taken as pressed two or more times. Setting this interval to 0 is to set the default value.

ticks Specifies the pen-repeat interval in ticks.

Returned Value

The returned value is the previous pen-repeat interval.

Application

General Description

Retrieves the applications on Penbex OS.

Related Data Structure

ApCell Structure

This structure stores all information for a Penbex Application.

```
typedef struct tagApCell {
    char szFilename[FILENAME_LEN+1];           // Filename. It is in "8.3" format
    char szAppName[APPNAME_LEN+1];            // Application Name
    const void *ApSmallIcon;                   // Small icon. The size is 16*16, in pixel
    const void *ApLargeIcon;                   // Large icon. The size is 16*16, in pixel
    StdMain entry;                             // AP entry. It is PenbexMain()
    DWORD dwCodeSize;                          // 0: means PrePBX
    const void *pBssData;                      // Initial value the global variable structure
    DWORD dwBssDataLen;                        // the length of global variable structure
    void **ppGlobalVar;                        // The location of global data of preload
                                           // AP, It is 0 for download AP
    VHANDLE hRes;                              // Resource data
    DWORD *pdwRelocText;
    WORD wRelocTextCnt;
    DWORD *pdwRelocData;
    WORD wRelocDataCnt;
    void *PrePBXEntry;
    WORD wAppType;                             // The type of the Application.The
                                           // following are possible values:
                                           // APPTYPE_PREPBX          Preload AP
                                           // APPTYPE_DOWNLOAD       Download AP
    char szAppVersion[APPVEERSION_LEN+1];     // Application's version information
    char szAppGroup[APPGROUP_LEN+1];          // Group Information for AP
    BYTE bReserved[20];
} ApCell;
```

System Message

APP_RUN_NORMAL argv	Runs the application in normal state. The Container ID
APP_RUN_COLD_RESET	System cold reset.
APP_RUN_WARM_RESET	System warm reset.
APP_RUN_SEARCH	System global search.
APP_RUN_ALARM	argv is user defined data.
APP_RUN_SYNC_NOTIFY	System synchronism notify.
APP_RUN_IRDA_NOTIFY	System Irda notify.
APP_RUN_KILLED	System kill the application.
APP_RUN_SYSTIME_CHANGE	System time sah been changed.

APP_RUN_GOTO

System Goto command.

AppAdd

Synopsis

```
int AppAdd(const void *sIcon, const void *bIcon, const char *filename,  
           const char *apName, StdMain entry, DWORD codeSize, const void *bssData,  
           DWORD len, void **ppVar);
```

Description

Adds a new application program into emulator. This function is called in “exprep.c” and used only for emulator to add an application to the desktop. Application program with extension name “PBX” is always added to desktop automatically after being downloaded.

<i>sIcon</i>	Specifies the file name of the small icon.
<i>bIcon</i>	Specifies the file name of the large icon.
<i>filename</i>	Specifies the file name of the executable.
<i>apName</i>	Specifies the application name appeared in PDA desktop.
<i>entry</i>	Specifies the name of application program entry.
<i>codeSize</i>	Specifies the size of the application structure.
<i>bssData</i>	
<i>len</i>	Specifies the length of structure application program.
<i>ppVar</i>	Specifies the application structure.

Returned Value

The returned value is a unique number assigned to application program.

AppCallAp

Synopsis

```
void AppCallAp(int index, DWORD argc, void *argv);
```

Description

Calls another application program from current one.

<i>index</i>	Specifies the unique index number of the application program.
<i>argc</i>	Specifies the active command.
<i>argv</i>	Specifies the active command parameters.

Returned Value

None.

AppGetCell

Synopsis

```
ApCell *AppGetCell(int index);
```

Description

Obtains the information of an application program.

<i>index</i>	Specifies the unique index number of the application program.
--------------	---

Returned Value

The returned value is the ApCell structure which points to an application program.

AppGetID

Synopsis

```
DWORD AppGetID(int index);
```

Description

Obtains the resource ID of an application program. This resource ID is used in AlarmInfo structure.

index Specifies the unique index number of the application program.

Returned Value

The returned value is a unique resource ID of application program if this function succeeds. Otherwise, the returned value is 0xFFFF.

AppGetIndex

Synopsis

```
int AppGetIndex(const char *filename);
```

Description

Obtains the index of application program.

**filename* Specifies the filename of application program with complete path.

Returned Value

The returned value is the unique index number of the application program if this function succeeds. Otherwise, the returned value is -1.

AppIsReEntrant

Synopsis

```
BOOL AppIsReEntrant(void);
```

Description

Checks the status of ReEntrant situation.

Returned Value

The returned value is TRUE if application on reentrant situation. Otherwise, the returned value is FALSE.

AppInsert

Synopsis

```
int AppInsert(void *downData);
```

Description

Adds a downloadable application program to system.

**downData* Specifies the downloadable application program.

Returned Value

The returned value is 1 if this function succeeds. Otherwise, the returned value is 0.

AppInsertByDR

Synopsis

```
int AppInsertByDR(void *pDRBlock);
```

Description

Adds a downloadable application program to system.

**pDRBlock* Specifies the DR pointer of downloadable application program.

Returned Value

The returned value is greater than zero if this function succeeds. Otherwise, the returned value is equal to or smaller than zero.

AppRemove

Synopsis

```
BOOL AppRemove(int index);
```

Description

Removes an application program.

index Specifies the unique index number of the application program.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

AppReturnHome

Synopsis

```
void AppReturnHome(void);
```

Description

Returned Value to the upper level of this application program.

Returned Value

None.

AppRunAp

Synopsis

```
void AppRunAp(int index);
```

Description

Sets next application program to run.

index Specifies the unique index number of the application program.

Returned Value

None.

AppRunApCont

Synopsis

```
void AppRunApCont(int index, WORD contID);
```

Description

Forces to run the application program of specific container.

index Specifies the unique index number of the application program.
contID Specifies the unique number of the specific container.

Returned Value

None.

AppRunApEx

Synopsis

```
void AppRunApEx(int index, DWORD argc, void *argv);
```

Description

Sets next application program to run.

index Specifies the unique index number of the application program.
argc Specifies the active command.
**argv* Specifies the active command parameters.

Returned Value

None.

AppRunPbxFile

Synopsis

```
BOOL AppRunPbxFile(const char *filename, DWORD argc, void *argv);
```

Description

Starts running an application which is already in file system.

<i>filename</i>	Specifies the file name of application program with complete path.
<i>argc</i>	Specifies the active command.
<i>*argv</i>	Specifies the active command parameters.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

AppRunPreAp

Synopsis

```
void AppRunPreAp(void);
```

Description

Terminates the current application and returns to the previous application program.

Returned Value

None

AppTotal

Synopsis

```
int AppTotal(void);
```

Description

Obtains the total of application program on the desktop.

Returned Value

The returned value is the total of application program.

Memory

General Description

The functions of memory retrieve the consecutive memory buffer. The memory handle can be locked/unlocked via MmLockH()/MmUnlockH().

Related Data Structure

None.

System Message

No system message for memory retrieving.

MmDelH

Synopsis

```
BOOL MmDelH(MmHandle *handle);
```

Description

Releases the memory handle which is allocated via MmNewH().

**handle* Specifies the memory handle to be deleted.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

MmDelP

Synopsis

```
BOOL MmDelP(void *memory);
```

Description

Releases the memory buffer which is allocated via MmNewP().

**memory* Specifies the memory buffer to be deleted.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

MmLockH

Synopsis

```
void *MmLockH(MmHandle *handle);
```

Description

Locks the memory handle for writing.

**handle* Specifies the memory handle to be locked.

Returned Value

The returned value is the locked memory handle if this function succeeds. Otherwise, the returned value is 0.

MmNewH

Synopsis

```
MmHandle MmNewH(DWORD size);
```

Description

Allocates a consecutive memory handle and obtains the pointer of this memory.

size Specifies the size of the memory buffer to be allocated.

Returned Value

The returned value is the handle of the memory buffer if this function succeeds. Otherwise, the returned value is NULL.

MmNewP

Synopsis

```
void *MmNewP(DWORD size);
```

Description

Allocates a consecutive memory buffer and obtains the pointer of this memory.

size Specifies the size of the memory buffer to be allocated.

Returned Value

The returned value is the pointer of the memory buffer if the function succeeds. Otherwise, the returned value is NULL.

MmResizeH

Synopsis

```
BOOL MmResizeH(MmHandle handle, DWORD newSize);
```

Description

Changes the size of the memory handle which is allocated via MmNewH().

handle Specifies the memory handle to be reallocated.
newSize Specifies the new size of the memory handle.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

MmResizeP

Synopsis

```
BOOL MmResizeP(void *memory, DWORD newSize);
```

Description

Changes the size of the memory buffer which is allocated via MmNewP().

**memory* Specifies the memory buffer to be reallocated.
newSize Specifies the new size of the memory buffer.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

MmSizeofH

Synopsis

```
DWORD MmSizeofH(MmHandle handle);
```

Description

Obtains the size of the memory handle which is allocated via MmNewH().

handle Specifies the memory handle to be retrieved.

Returned Value

The returned value is the size of the memory buffer if this function succeeds. Otherwise, the returned value is 0.

MmSizeofP

Synopsis

```
DWORD MmSizeofP(void *memory);
```

Description

Obtains the size of the memory buffer which is allocated via MmNewP().

**memory* Specifies the memory buffer to be retrieved.

Returned Value

The returned value is the size of the memory buffer if this function succeeds. Otherwise, the returned value is 0.

MmMaxFreeBlock

Synopsis

```
DWORD MmMaxFreeBlock(void);
```

Description

Obtains the maximum consecutive free memory space of the system.

Returned Value

The returned value is the free memory space of the system in blocks if this function succeeds. Otherwise, the returned value is 0.

MmTotalFreeMem

Synopsis

```
DWORD MmTotalFreeMem(void);
```

Description

Obtains the free memory space of the system in bytes.

Returned Value

The returned value is the free memory space of the system in bytes if this function succeeds. Otherwise, the returned value is 0.

MmUnlockH

Synopsis

```
BOOL MmUnlockH(MmHandle handle);
```

Description

Unlocks the memory handle.

**handle* Specifies the memory handle to be unlocked.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Battery

General Description

Obtains the information of battery.

Related Data Structure

None.

System Message

MT_BATTERY_LOW	When the system' s battery reach the low condition.
pMsg->id	The battery' s voltage state. The following are possible value:
	1 The battery' s voltage is very low.
	0 The battery' s voltage is low.

BattCurrentValue

Synopsis

```
DWORD BattCurrentValue(void);
```

Description

Obtains the current power of the battery in mV. The largest value is 3000 mV.

Returned Value

The returned value is the current power of the battery in mV.

BattReadChargingStatus

Synopsis

```
int BattReadChargingStatus(void);
```

Description

Obtains the charging status of the battery.

Returned Value

The returned value is status of charging. There are three possible values:

BATT_NO_CHARGING Battery is not charging.

BATT_IS_CHARGING Battery is charging and the power is not full.

BATT_FINISH_CHARGING Battery is charging and the power is full.

Alarm

General Description

Creates alarm object.

Related Data Structure

AlarmInfo Structure

```
typedef struct tagAlarmInfo {  
    WORD Flag;  
  
    WORD SongIndex;  
    WORD ReminderTimes;  
    WORD RepeatValue;  
    WORD RepeatTimes;  
  
    char *szMsg;  
    DWORD PreSec;  
    DWORD dwApID;  
    DWORD RemindPeriod;  
    WORD ExFlag;  
  
    BYTE MonthNumWeek;  
    BYTE WeekOnDay;  
  
    RtcDateTime BeginDate;  
    RtcDate EndDate;  
  
    // The flag for alarm type. The following are possible value:  
    // ALM_FLAG_SHOW_MSG          Pop up messages while  
                                alarm occur.  
    // ALM_FLAG_PLAY_SOUND        Play sounds.  
    // ALM_FLAG_REMINDER          Contine reminder (every  
                                3 mins) the user.  
    // ALM_FLAG_REMOVE_AUTO       Unreachable item will be  
                                removed.  
    // ALM_FLAG_REPEAT_DAY        Repeat every day.  
    // ALM_FLAG_REPEAT_WEEK       Repeat every week.  
    // ALM_FLAG_REPEAT_MONTH      Repeat every month.  
    // ALM_FLAG_REPEAT_YEAR       Repeat every year.  
    // ALM_FLAG_REPEAT_NUMDAY     Repeat every X days.  
    // ALM_FLAG_REPEAT_NUMWEEK    Repeat every X weeks.  
    // ALM_FLAG_REPEAT_LUNAR      Repeat Lunar day.  
    // ALM_FLAG_REPEAT_ENABLE     Repeat function will  
                                effect.  
    // ALM_FLAG_REPEAT_ALWAYS     Repeat always,  
    // 0  
    // 1-7.  
    // For NUMDAY:every x days, NUMWEEK:every x weeks.  
    // Repeat how many times. It will ignore if  
    // ALM_FLAG_REPEAT_ALWAYS set.  
    // Alarm messages to be shown.  
    // Trigger the alarm before the setting seconds.  
    // Get by AppGetID (index)  
    // 1-63 (min.)  
    // The extra flag for alarm type. The following are possible value:  
    // ALM_EXFLAG_MONTH_BY_DAY  
    // ALM_EXFLAG_MONTH_LAST_DAY  
    // ALM_EXFLAG_MONTH_LAST_WEEK  
    // ALM_EXFLAG_PRE_UNIT_MIN  
    // ALM_EXFLAG_PRE_UNIT_HOUR  
    // ALM_EXFLAG_PRE_UNIT_DAY  
    // The week number in a month to be set to alarm.  
    // The day number in a week to be set to alarm. The following  
    // are possible value:  
    // ALM_WEEK_SUN  
    // ALM_WEEK_MON  
    // ALM_WEEK_TUE  
    // ALM_WEEK_WED  
    // ALM_WEEK_THU  
    // ALM_WEEK_FRI  
    // ALM_WEEK_SAT  
    // The beginning date for alarm  
    // The end date for alarm
```

```
} AlarmInfo;
```

System Message

No system message for alarm retrieving.

RtcAddAlarm

Synopsis

```
int RtcAddAlarm(AlarmInfo *info, RtcDateTime *date);
```

Description

Adds an alarm object.

<i>*info</i>	Specifies the alarm data structure.
<i>*date</i>	Specifies the date-and-time structure.

Returned Value

The returned value is a unique handle if the function succeeds. Otherwise it returns 0.

RtcDelAlarm

Synopsis

```
void RtcDelAlarm(int handle);
```

Description

Deletes an alarm object.

<i>handle</i>	Specifies the handle of the alarm.
---------------	------------------------------------

Returned Value

None.

RtcDelAllAlarm

Synopsis

```
void RtcDelAllAlarm(DWORD apID);
```

Description

Deletes all alarm objects of an application.

<i>apID</i>	Specifies the unique ID of the application. This can be obtained by AppGetID().
-------------	---

Returned Value

None.

Real Time Clock

General Description

Retrieves real time clock.

Related Data Structure

RtcTime Structure

This structure stores all information of RTC Time.

```
typedef struct tagRtcTime {  
    WORD sec;           // Second. The range is 0-59.  
    WORD min;           // Minute. The range is 0-59.  
    WORD hour;          // Hour. The range is 0-23.  
} RtcTime;
```

RtcTime Structure

This structure stores all information of RTC Date.

```
typedef struct tagRtcDate {  
    WORD day;           // Day. The range is 1-31.  
    WORD month;         // Month. The range is 0-11.  
    WORD year;          // Year. The range is 1970-2038.  
} RtcDate;
```

RtcDateTime Structure

This structure stores all information of of RTC Date and Time.

```
typedef struct tagRtcDateTime {  
    WORD sec;           // Second. The range is 0-59.  
    WORD min;           // Minute. The range is 0-59.  
    WORD hour;          // Hour. The range is 0-23.  
    WORD day;           // Day. The range is 1-31.  
    WORD month;         // Month. The range is 0-11.  
    WORD year;          // Year. The range is 1970-2038.  
} RtcDateTime;
```

System Message

MT_RTC_STPWCH When the time setting by the RtcEnableStopWatch() is reached.

RtcDisableStopWatch

Synopsis

```
void RtcDisableStopWatch(void);
```

Description

Enables the count-down function of the Dragon CPU.

Returned Value

None.

RtcEnableStopWatch

Synopsis

```
int RtcEnableStopWatch(int min);
```

Description

Enables the count-down function of the Dragon CPU.

min Specifies the minute to countdown.

Returned Value

The returned value is old value.

RtcGetDate

Synopsis

```
void RtcGetDate(WORD *year, WORD *month, WORD *day);
```

Description

Obtains the values of the year, month, and day from system.

**year* Specifies the memory buffer to which the value of the year is copied.
**month* Specifies the memory buffer to which the value of the month is copied.
**day* Specifies the memory buffer to which the value of the day is copied.

Returned Value

None.

RtcGetDateTime

Synopsis

```
void RtcGetDateTime(RtcDateTime *rtcdate);
```

Description

Obtains the system date and time in RtcDateTime structure.

**rtcdate* Specifies the memory buffer of RtcDateTime structure to which the time data is

copied.

Returned Value

None.

RtcGetTime

Synopsis

```
void RtcGetTime(WORD *hour, WORD *min, WORD *sec);
```

Description

Obtains the system time in values of the hour, minute and second.

<i>*hour</i>	Specifies the memory buffer to which the value of the hour is copied.
<i>*min</i>	Specifies the memory buffer to which the value of the minute is copied.
<i>*sec</i>	Specifies the memory buffer to which the value of the second is copied.

Returned Value

None.

RtcGetTime2

Synopsis

```
void RtcGetTime2(RtcTime *rtc);
```

Description

Obtains the system time in RtcTime structure.

<i>*rtc</i>	Specifies the memory buffer of RtcTime structure to which the time data is copied.
-------------	--

Returned Value

None.

RtcSetDate

Synopsis

```
void RtcSetDate(WORD year, WORD month, WORD day);
```

Description

Sets the value of system year, month and day.

<i>*year</i>	Specifies the value of the year. The range is from 1970 to 2038.
<i>*month</i>	Specifies the value of the month. The range is from 0 to 11.
<i>*day</i>	Specifies the value of the day. The range is from 1 to 31.

Returned Value

None.

RtcSetTime

Synopsis

```
void RtcSetTime(WORD hour, WORD min, WORD sec);
```

Description

Sets the value of system hour, minute and second.

<i>*hour</i>	Specifies the value of the hour. The range is from 0 to 23.
<i>*min</i>	Specifies the value of the minute. The range is from 0 to 59.
<i>*sec</i>	Specifies the value of the second. The range is from 0 to 59.

Returned Value

None.

Copy Pool

General Description

Retrieves the global copy pool.

Related Data Structure

None.

System Message

No system message for copy pool retrieving.

CPClean

Synopsis

```
void CPClean(void);
```

Description

Clears the data in the pool which is a memory buffer provided by system.

Returned Value

None.

CPGetData

Synopsis

```
BOOL CPGetData(void *buffer);
```

Description

Obtains data from pool which is a memory buffer provided by system.

**buffer* Specifies the memory buffer to which the data of the pool is copied.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

CPGetDataInfo

Synopsis

```
void CPGetDataInfo(WORD *type, DWORD *size);
```

Description

Obtains data type and size of the copied data from copy pool.

**type* Specifies the returned data type. The data type could be the followings:

CP_EMPTY	Empty
CP_BMP	BMP graphics
CP_STRING	Character string
CP_BIN	Binary

**size* Specifies the returned data size.

Returned Value

None.

CPPutData

Synopsis

```
BOOL CPPutData(WORD type, void *data, DWORD size);
```

Description

Puts the data into the pool which is a memory buffer provided by system.

<i>type</i>	Specifies the data type of copied data. The data type could be the followings: CP_EMPTY Empty CP_BMP BMP graphics CP_STRING Character string CP_BIN Binary
<i>*data</i>	Specifies the data to be set to the pool.
<i>size</i>	Specifies the data size.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Shared Library

General Description

The following are possible returned values for some of the shared library API's:

SLERR_INVALID_PARAMETER
SLERR_NO_MEMORY
SLERR_PBL_CANNOT_OPEN
SLERR_PBL_CANNOT_READ

Related Data Structure

PBL_HEADER Structure

This structure stores all information of a field of database.

```
typedef struct tagPBL_HEADER {  
    char    szSignature[4];                // "PBL\0"  
    BYTE    bMajorVersion;                //  
    BYTE    bMinorVersion;                //  
    BYTE    bMicroVersion;                //  
    BYTE    bLibLanguage;                //  
    unsigned long ulFileSize;              // the total size of file  
    unsigned long ulTextSize;              // including offset table & code  
    unsigned long ulTextStart;             // the start position of .text section  
    unsigned long ulDataSize;              // the size of data section  
    unsigned long ulDataStart;             // the start position of .data section  
    unsigned long ulBssSize;               // the size of uninitial data  
    char    szFilename[FILENAME_LEN + 1]; //  
    char    szLibName[LIBNAME_LEN + 1];   //  
    char    szLibDate[LIBDATE_LEN + 1];   //e.g. "2000/11/02\0"  
    char    szLibComment[LIBCOMMENT_LEN + 1]; //  
} PBL_HEADER;
```

System Message

No system message for shared library retrieving.

SlClose

Synopsis

```
int SlClose(BYTE id);
```

Description

Closes a shared library.

id Specifies the ID for the shared library used in the program.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be SLERR_INVALID_PARAMETER, SLERR_NO_MEMORY, SLERR_PBL_CANNOT_OPEN or SLERR_PBL_CANNOT_READ.

SlCloseAll

Synopsis

```
void SlCloseAll(void);
```

Description

Closes all shared libraries.

Returned Value

None.

SlFunc

Synopsis

```
void SlFunc(BYTE id, BYTE funcID);
```

Description

Calls the function in the shared library.

id Specifies the ID for the shared library used in the program.

funcID Specifies the function ID in the share library.

Returned Value

None.

SlInit

Synopsis

```
void SlInit(void);
```

Description

Allocates memory for shared library.

Returned Value

None.

S1Open

Synopsis

```
int S1Open(const char *libName, BYTE id, PBL_HEADER pblHeader);
```

Description

Opens a shared library.

<i>*libName</i>	Specifies the name for the shared library used in the program.
<i>id</i>	Specifies the ID for the shared library used in the program.
<i>pblHeader</i>	Specifies the PBL_HEADER structure. All needed information must be stored in this structure.

Returned Value

The returned value is 0 if this function succeeds. Otherwise, the returned value is a negative value, which can be SLERR_INVALID_PARAMETER, SLERR_NO_MEMORY, SLERR_PBL_CANNOT_OPEN or SLERR_PBL_CANNOT_READ.

System Settings

General Description

Obtains the information of system settings.

Related Data Structure

None.

System Message

No system message for system information retrieving.

SysGetHwInfo

Synopsis

```
long SysGetHwInfo(long hwType, long option);
```

Description

Obtains the hardware information of the system.

hwType Specifies the type of the hardware.
option This is reserved for future use.

Returned Value

The returned value depends on the type of hardware if this function succeeds. The following are the hardware type and the associated returned value(s).

Type	Returned Value
SYS_HW_CPU	SYS_HW_CPU_M68EZ328/ SYS_HW_CPU_M68VZ328/ SYS_HW_CPU_ARM
SYS_HW_ROM_COUNT	Actual count.
SYS_HW_ROM_TYPE	SYS_HW_ERROR/ SYS_HW_ROM_NOR/ SYS_HW_ROM_MASK
SYS_HW_ROM_SIZE	Size in Kbyte
SYS_HW_RAM_SIZE	Size in Kbyte
SYS_HW_NAND_SIZE	Size in Kbyte
SYS_HW_LCD_WIDTH	Number in pixel
SYS_HW_LCD_HEIGHT	Number in pixel
SYS_HW_LCD_TYPE	SYS_HW_LCD_GRAY/SYS_HW_LCD_COLOR
SYS_HW_KEY_COUNT	Actual count
SYS_HW_TP_EXIST	SYS_HW_YES/SYS_HW_NO
SYS_HW_UART_EXIST	SYS_HW_YES/SYS_HW_NO
SYS_HW_USB_EXIST	SYS_HW_YES/SYS_HW_NO
SYS_HW_MIC_EXIST	SYS_HW_YES/SYS_HW_NO
SYS_HW_SPK_EXIST	SYS_HW_YES/SYS_HW_NO
SYS_HW_BUZZER_EXIST	SYS_HW_YES/SYS_HW_NO
SYS_HW_IR_EXIST	SYS_HW_YES/SYS_HW_NO
SYS_HW_CF_SOCKET_EXIST	SYS_HW_YES/SYS_HW_NO
SYS_HW_SMC_SOCKET_EXIST	SYS_HW_YES/SYS_HW_NO
SYS_HW_SD_SOCKET_EXIST	SYS_HW_YES/SYS_HW_NO
SYS_HW_PCMCIA_SOCKET_EXIST	SYS_HW_YES/SYS_HW_NO
SYS_HW_SERIAL_NUMBER	SYS_HW_YES/SYS_HW_NO
SYS_HW_MP3_EXIST	SYS_HW_YES/SYS_HW_NO

Otherwise, the returned value is SYS_INFO_ERROR.

SysGetIOInfo

Synopsis

```
long SysGetIOInfo(long ioType, long option);
```

Description

Obtains the I/O information of the system.

ioType Specifies the type of the I/O.

option This is reserved for future use.

Returned Value

The returned value depends on the type of hardware if this function succeeds. The following are the hardware type and the associated returned value(s).

Type	Returned Value
SYS_IO_COLOR	Actual number of color.
SYS_IO_LANGUAGE_COUNT	Internal use.
SYS_IO_LANGUAGE_TYPE	SYS_OS_LANGUAGE_ENG/SYS_OS_LANGUAGE_GB/ SYS_OS_LANGUAGE_BIG5
SYS_IO_IRDA11_EXIST	SYS_OS_YES/SYS_OS_NO

Otherwise, the returned value is SYS_INFO_ERROR.

SysGetOsInfo

Synopsis

```
long SysGetOsInfo(long osType, long option);
```

Description

Obtains the OS information of the system.

osType Specifies the type of the OS.
option This is reserved for future use.

Returned Value

The returned value depends on the type of operating system if this function succeeds. The following are the hardware type and the associated returned value(s).

Type	Returned Value
SYS_OS_COLOR	Actual number of color.
SYS_OS_LANGUAGE_COUNT	Internal use.
SYS_OS_LANGUAGE_TYPE	SYS_OS_LANGUAGE_ENG/SYS_OS_LANGUAGE_GB/ SYS_OS_LANGUAGE_BIG5
SYS_OS_IRDA11_EXIST	SYS_OS_YES/SYS_OS_NO

Otherwise, the returned value is SYS_INFO_ERROR.

Handwriting Recognition

General Description

Provides some extensible API's for handwriting recognition application.

Related Data Structure

HREGPARA Structure

This structure stores all information of the Hreg parameter.

```
typedef struct tagHREGPARA{
    char IsChnPlugIn;           // Whether there exists input method that has been plugged in.
    char IsInkingEnable;        // The milliseconds of repetition.
    WORD RepeatMS;              // Milliseconds of repetition.
    WORD MoveMS;                // Sampling intervals of strokes.
    WORD VibrateRange;          // Vibration range of first stroke.
    WORD wReserved[4];          // Reserved.
} HREGPARA;
```

PLUGINPROC Structure

This structure stores all information for the plug-in process.

```
typedef struct tagPLUGINPROC{
    CHI_HREG_INIT      HregInit;    // Initialization routine.
    CHI_HREG_BEGIN     HregBegin;    // Start routine.
    CHI_HREG_PROC       HregProc;    // Stroke handling routine.
    CHI_HREG_REG        HregReg;     // Recogniziton routine.
} PLUGINPROC;
```

PbxHregInfo Structure

This structure stores all information for the plug-in process.

```
typedef struct tagPbxHregInfo {
    BYTE      PenWidth, DefPenWdith, IsFirstStroke, bWaitStateCnt;
    WORD      wRegCnt, wChiRegCnt, DefTimeOut;
    DWORD     dwLastMoveTick, dwKept;
    UITMR_HANDLE pTimer;
    WORD      wHregBuf[MAX_COLLECT_POINTS*3+8];
    BYTE      bScrnBuf[160*160*2];
} PbxHregInfo;
```

System Message

MT_HANDREG_COMMAND	Presses on the Handreg Object.
pMsg->id	The right side or left side of Handreg. The following are possible value:
1	Left side.
0	Right side.

EnhGetState

Synopsis

```
BYTE EnhGetState(void);
```

Description

(This function is reserved for future implementation.)

EnhRecognize

Synopsis

```
WORD EnhRecognize(unsigned short *handwrite);
```

Description

Obtains the recognition of handwriting. The handwriting is a serial bit-value with NULL terminated (' \0'). For handwriting with multiple strokes, 0xFFFF is used as a separator for every stroke.

**handwrite* Specifies the bit values of handwriting.

Returned Value

The returned value is the recognized character if this recognition succeeds. Otherwise, the returned value is 0.

EnhSetState

Synopsis

```
int EnhSetState(int status);
```

Description

(This function is reserved for future implementation.)

hregGetDefPara

Synopsis

```
BOOL hregGetDefPara(BYTE *penWidth, WORD *timeout, void *config);
```

Description

Obtains default configurations of handwriting recognition system.

<i>*penWidth</i>	Specifies the memory buffer to which the pen width of strokes is copied.
<i>*timeout</i>	Specifies the memory buffer to which the timeout value in millisecond is copied. This timeout value is used to recognize a word is finished.
<i>*config</i>	Specifies the memory buffer to which the configuration parameters of recognition engine.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

hregNewEx

Synopsis

UIOBJ hregNewEx(WORD *id*, WORD *x*, WORD *y*, WORD *w*, WORD *h*, REGINIT *reginit*);

Description

Creates an extended handwriting recognition object.

<i>id</i>	Specifies the object ID of this handwriting recognition object.
<i>x</i>	Specifies x-coordinate of left side of this object, in pixels.
<i>y</i>	Specifies y-coordinate of top side of this object, in pixels.
<i>w</i>	Specifies the width of this object in pixels.
<i>h</i>	Specifies the height of this object in pixels.
<i>reginit</i>	Specifies the initialization routine of engine.

Returned Value

The returned value is the handwriting recognition object if this function succeeds. Otherwise, the returned value is NULL.

hregSetDefPenWidth

Synopsis

BYTE hregSetDefPenWidth(BYTE *penWidth*);

Description

Set the default width of stroke.

penWidth Specifies the pen width of strokes. The range is from 1 to 7. The default value is 2.

Returned Value

The returned value is the previous pen width.

hregSetDefTimeOut

Synopsis

WORD hregSetDefTimeOut(WORD *timeout*);

Description

Sets the timeout value for handwriting recognition.

timeout Specifies the timeout value in milliseconds. This timeout value is used to recognize a word is finished. The range is from 200 to 3000. The default value is 600.

Returned Value

The returned value is the previous pen width.

PbxHregEnableInk

Synopsis

```
int PbxHregEnableInk(BOOL isEnabled) ;
```

Description

Enables or disables full-screened handwriting recognition.

<i>isEnabled</i>	Specifies the enabling flag.
TRUE	Enabling
FALSE	Disabling

Returned Value

The returned value is the previous enabling status.

1	The previous status is enabled.
0	The previous status is disabled.

PbxHregGetParam

Synopsis

```
BOOL PbxHregGetParam(HREGPARAM *hregPara) ;
```

Description

Obtains configuration parameters of handwriting recognition system.

**hregPara* Specifies the HREGPARAM buffer to which the configuration parameters are copied.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

PbxHregInit

Synopsis

```
BOOL PbxHregInit(int x, int y, int w, int h) ;
```

Description

Initializes and sets the handwriting area.

<i>x</i>	Specifies x-coordinate of left side of this area, in pixels.
<i>y</i>	Specifies y-coordinate of top side of this area, in pixels.
<i>w</i>	Specifies the width of this area in pixels.
<i>h</i>	Specifies the height of this area in pixels.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

PbxHregPlugIn

Synopsis

```
BOOL PbxHregPlugIn(PLUGINPROC *proc, WORD cmd) ;
```


Description

Enables full screen of handwriting recognition system and sets the control character for turning on the recognition engine.

**proc* Specifies the PLUGINPROC structure of the engine.
cmd Specifies the control character.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

PbxHregPlugOut

Synopsis

```
BOOL PbxHregPlugOut(void);
```

Description

Releases the full screen of handwriting recognition system.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

PbxHregSetParam

Synopsis

```
BOOL PbxHregSetParam(HREGPARAM *hregPara);
```

Description

Sets the parameters of handwriting recognition.

**hregPara* Specifies the HREGPARAM structure to be set to system.

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE.

Trap

General Description

Provides functions of system interrupt.

Related Data Structure

None.

System Message

No system message for trap retrieving.

TrapGetEntry

Synopsis

```
FuncPtr TrapGetEntry(WORD trapNo);
```

Description

Obtains function pointer according to the trap number
trapNo Specifies the trap number.

Returned Value

The returned value is function pointer if this function succeeds. Otherwise, the returned value is NULL.

TrapGetLastApiNum

Synopsis

```
WORD TrapGetLastApiNum(void);
```

Description

Obtains the maximal trap number of the system API.

Returned Value

The returned value is the maximal trap number of the system API.

TrapHookEntry

Synopsis

```
FuncPtr TrapHookEntry(WORD trapNo, FuncPtr newEntry);
```

Description

Registers a new function into the trap table.

trapNo Specifies the trap number.
newEntry Specifies the new function to be registered.

Returned Value

The returned value is old function pointer if this function succeeds. Otherwise, the returned value is 0.

Miscellaneous

OSVersionStr

Synopsis

```
char *OSVersionStr(void);
```

Description

Obtains the version information of Penbex OS.

Returned Value

The returned value is the OS version string.

Appendix A

Supported ANSI C Library

Abs

Synopsis

```
int Abs(int j);
```

ACos

Synopsis

```
double ACos(double x);
```

Asctime

Synopsis

```
char *AscTime(const struct tm *timeptr);
```

ASin

Synopsis

```
double ASin(double x);
```

Assertion

Synopsis

```
int Assertion(char *express, char *filename, int lineNo);
```

Description

Displays a maintenance description.

<i>*express</i>	Specifies the expression.
<i>*filename</i>	Specifies the file name.
<i>lineNo</i>	Specifies the line number.

Returned Value

The returned value is 1 if this function succeeds. Otherwise, the program terminates.

ATan

Synopsis

```
double ATan(doulbe x);
```

ATan2

Synopsis

```
double ATan2(double y, double x);
```

Atof

Synopsis

```
double Atof(const char *nptr);
```

Atoi

Synopsis

```
int Atoi(const char *nptr);
```

Atol

Synopsis

```
long int Atol(const char *nptr);
```

Bsearch

Synopsis

```
void *Bsearch(const void *key, const void *base, size_t nmemb, size_t size,  
              int(*compar)(const void *, const void *));
```

CalcWeekDay

Synopsis

```
int CalcWeekDay(int year, int month, int day);
```

Description

Obtains what day is the date which is input in the function.

<i>year</i>	Specifies the year to calculate (1904-2099).
<i>month</i>	Specifies the month to calculate [0 (January) - 11(December)].
<i>day</i>	Specifies the day to calculate (1 - 31).

Returned Value

The returned value is 0 (Sunday) - 6 (Saturday) if the function succeeds. Otherwise, the returned value is -1.

CalcWeekDay2

Synopsis

```
int CalcWeekDay2(int year, int day);
```

Description

Obtains what day is the input day of the year.

<i>year</i>	Specifies the year to calculate (1904-2099).
<i>day</i>	Specifies the day in a year to calculate (0-365).

Returned Value

The returned value is 0 (Sunday) - 6 (Saturday) if the function succeeds. Otherwise, the returned value is -1.

CalcYearDay

Synopsis

```
int CalcYearDay(int year, int month, int day);
```

Description

Obtains which day is the input date of a year.

<i>year</i>	Specifies the year to calculate (1904-2099).
<i>month</i>	Specifies the month to calculate [0 (January) - 11(December)].
<i>day</i>	Specifies the day to calculate (1 - 31).

Returned Value

The returned value is 0 - 365 if the function succeeds. Otherwise, the returned value is -1.

Calloc

Synopsis

```
void *Calloc(size_t nmemb, size_t size);
```

Ceil

Synopsis

```
double Ceil(double x);
```

Clock

Synopsis

```
Clock_t Clock(void);
```

Cos

Synopsis

```
double Cos(double x);
```

Cosh

Synopsis

```
double Cosh(double x);
```

CoTan

Synopsis

```
double CoTan(double x);
```

CTime

Synopsis

```
char *CTime(const time_t *timer);
```

DiffTime

Synopsis

```
double DiffTime(time_t time1, time_t time0);
```

Div

Synopsis

```
Div_t Div(int numer, int denom);
```

dw2Str

Synopsis

```
char *dw2Str(unsigned long number);
```

Description

Transforms a double-word integer to a string.

number Specifies the double-word integer to be transformed.

Returned Value

The returned value is the pointer of the transformed string.

Exp

Synopsis

```
double Exp(double x);
```

Fabs

Synopsis

```
double Fabs(double x);
```

Floor

Synopsis

```
double Floor(double x);
```

Fmod

Synopsis

```
double Fmod(double x, double y);
```

Free

Synopsis

```
void Free(void *ptr);
```

Frexp

Synopsis

```
double Frexp(double value, int *Exp);
```

Gmtime

Synopsis

```
struct tm *Gmtime(const time_t *timer);
```

Hypot

Synopsis

```
double Hypot(double x, double y);
```

Isalnum

Synopsis

```
int Isalnum(int c);
```

Isalpha

Synopsis

```
int Isalpha(int c);
```

Isascii

Synopsis

```
int Isascii(int c);
```

Iscntrl

Synopsis

```
int Iscntrl(int c);
```

Isdigit

Synopsis

```
int Isdigit(int c);
```

Isgraph

Synopsis

```
int Isgraph(int c);
```

Isleadbyte

Synopsis

```
int Isleadbyte(int c);
```

Islower

Synopsis

```
int Islower(int c);
```

Isprint

Synopsis

```
int Isprint(int c);
```

Ispunct

Synopsis

```
int Ispunct(int c);
```

Isspace

Synopsis

```
int Isspace(int c);
```

Isupper

Synopsis

```
int Isupper(int c);
```

Isxdigit

Synopsis

```
int Isxdigit(int c);
```

Labs

Synopsis

```
long int Labs(long int j);
```

Ldexp

Synopsis

```
double Ldexp(double x, int Exp);
```

Ldiv

Synopsis

```
Ldiv_t Ldiv(long int numer, long int denom);
```

Localtime

Synopsis

```
struct tm *Localtime(const time_t *timer);
```

Log

Synopsis

```
double Log(double x);
```

Log10

Synopsis

```
double Log10(double x);
```

Longjmp

Synopsis

```
void Longjmp(jmp_buf env, int val);
```

Lunar2Solar

Synopsis

```
BOOL Lunar2Solar(WORD year, WORD month, WORD day, BOOL isLeap);
```

Description

Translates Lunar calendar to Gregorian calendar. The parameters of the function are lunar year, month, day, and if the flag of leap month. The function returns a corresponding Gregorian date.

<i>*year</i>	Specifies the year (1902-2071).
<i>*month</i>	Specifies the month to calculate [0 (January) - 11(December)].
<i>*day</i>	Specifies the day to calculate (1 - 30).
<i>isLeap</i>	Specifies the flag of leap month.
	TRUE Leap month
	FALSE Not leap month

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE. If the function returns FALSE, it means the failure of translation. It is possible that the input date is incorrect, or there is no leap month in the input year.

This function does not check the number of days in a month. Therefore, if you input the 30th day of a lunar month, it will be shifted to the next day of the Gregorian calendar.

Malloc

Synopsis

```
void *Malloc(size_t size);
```

Memchr

Synopsis

```
void *Memchr(const void *s, int c, size_t n);
```

Memcmp

Synopsis

```
int Memcmp(const void *s1, const void *s2, size_t n);
```

Memcpy

Synopsis

```
void *Memcpy(void *s1, const void *s2, size_t n);
```

Memmove

Synopsis

```
void *Memmove(void *s1, const void *s2, size_t n);
```

Memset

Synopsis

```
void *Memset(void *s, int c, size_t n);
```

Mktime

Synopsis

```
time_t Mktime(struct tm *timeptr);
```

Example

What day of the week is July 4,2001?

```
static const char *const wday [] = {
    "Sunday", "Monday", "Tuesday", "Wednesday",
    "Thursday", "Friday", "Saturday", "-unknown-"
};
struct tm time_str;
/*...*/
time_str.tm_year = 2001 - 1900;
time_str.tm_mon = 7 - 1;
time_str.tm_mday = 4;
time_str.tm_hour = 0;
time_str.tm_min = 0;
time_str.tm_sec = 1;
time_str.tm_isdst = -1;
if(Mktime(&time_str) == -1)
time_str.tm_wday =7;
printf("%s\n", wday [time_str.tm_wday]);
```

Modf

Synopsis

```
double Modf(double value, double *iptr);
```

Pow

Synopsis

```
double Pow(double x, double y);
```

Qsort

Synopsis

```
void Qsort(void base, size_t nmemb, size_t size,  
            int(*compar)(const void *, const void *));
```

Rand

Synopsis

```
int Rand(void);
```

Realloc

Synopsis

```
void *Realloc(void ptr, size_t size);
```

SetGmRefTime

Synopsis

```
void SetGmRefTime(unsigned long seconds);
```

Description

Sets the offset seconds from GMT.

seconds Specifies the offset seconds.

Returned Value

None.

Setjmp

Synopsis

```
int Setjmp(jmp_buf env);
```

Sin

Synopsis

```
double Sin(double x);
```

Sinh

Synopsis

```
double Sinh(double x);
```

Solar2Lunar

Synopsis

```
BOOL Solar2Lunar(WORD *year, WORD *month, WORD *day, BOOL *isLeap);
```

Description

Translates Gregorian calendar to Lunar calendar. The parameters of the function are Gregorian year, month, day, and if the flag of leap month. The function returns a corresponding lunar date.

<i>*year</i>	Specifies the year (1903-2071).
<i>*month</i>	Specifies the month to calculate [0 (January) - 11(December)].
<i>*day</i>	Specifies the day to calculate (1 - 30).
<i>*isLeap</i>	Specifies the flag of leap month.
TRUE	Leap month
FALSE	Not leap month

Returned Value

The returned value is TRUE if this function succeeds. Otherwise, the returned value is FALSE. If the function returns 0, it means the failure of translation. It is possible that the input date is incorrect.

The acceptable year of the Gregorian calendar is 1903 while the acceptable year of the function Lunar2Solar() is 1904.

Sprintf

Synopsis

```
int Sprintf(char *s, const char *format, ...);
```

Sqrt

Synopsis

```
double Sqrt(double x);
```

Srand

Synopsis

```
void Srand(unsigned int seed);
```

Str2lower

Synopsis

```
char *Str2lower(char *string);
```

Description

Transforms a string to be all lowercase

**string* Specifies the string to be transformed.

Returned Value

The returned value is the pointer of the transformed string.

Str2upper

Synopsis

```
char *Str2upper(char *string);
```

Description

Transforms a string to be all uppercase

**string* Specifies the string to be transformed.

Returned Value

The returned value is the pointer of the transformed string.

Strcat

Synopsis

```
char *Strcat(char *s1, const char *s2);
```

Strchr

Synopsis

```
char *Strchr(const char *s, int c);
```

Strcmp

Synopsis

```
int Strcmp(const char *s1, const char *s2);
```

Strcpy

Synopsis

```
char *Strcpy(char *s1, const char *s2);
```


Strcspn

Synopsis

```
size_t Strcspn(const char *s1, const char *s2);
```

Strerror

Synopsis

```
char *Strerror(int errnum);
```

Stricmp

Synopsis

```
int Stricmp(const char *s1, const char *s2);
```

Strlen

Synopsis

```
size_t Strlen(const char *s);
```

Strncat

Synopsis

```
char *Strncat(char *s1, const char *s2, size_t n);
```

Strncmp

Synopsis

```
int Strncmp(const char *s1, const char *s2, size_t n);
```

Strncpy

Synopsis

```
char *Strncpy(char *s1, const char *s2, size_t n);
```

Strpbrk

Synopsis

```
char *Strpbrk(const char *s1, const char *s2);
```

Strrchr

Synopsis

```
char *Strrchr(const char *s, int c);
```

Strspn

Synopsis

```
size_t Strspn(const char *s1, const char *s2);
```

Strstr

Synopsis

```
char *Strstr(const char *s1, const char *s2);
```

Strtod

Synopsis

```
double Strtod(const char *nptr, char **endptr);
```

Strtok

Synopsis

```
char *Strtok(char *s1, const char *s2);
```

Example

```
static char str[] = "?a???b,,,#c";
char *t;

t = Strtok(str, "?"); /* t points to the token "a" */
t = Strtok(NULL, ","); /* t points to the token "??b" */
t = Strtok(NULL, "#,"); /* t points to the token "c" */
t = Strtok(NULL, "?"); /* t is a null pointer */
```

Strtol

Synopsis

```
long int Strtol(const char *nptr, char **endptr, int base);
```

Strtoul

Synopsis

```
unsigned long int Strtoul(const char *nptr, char **endptr, int base);
```

Tan

Synopsis

```
double Tan(double x);
```

Tanh

Synopsis

```
double Tanh(double x);
```

Time

Synopsis

```
time_t Time(time_t *timer);
```

Toascii

Synopsis

```
int Toascii(int c);
```

Tolower

Synopsis

```
int Tolower(int c);
```

Toupper

Synopsis

```
int Toupper(int c);
```

Index

A

Abs 373
 ACos 373
 AppAdd 337
 AppCallAp 337
 AppGetCell 337
 AppGetID 338
 AppGetIndex 338
 AppInsert 338
 AppInsertByDR 339
 AppIsReEntrant 338
 AppProcess 327
 AppRemove 339
 AppReturnHome 339
 AppRunAp 340
 AppRunApCont 340
 AppRunApEx 340
 AppRunPbxFile 340
 AppRunPreAp 341
 AppTotal 341
 Asctime 373
 ASin 373
 Assertion 373
 ATan 373
 ATan2 374
 Atof 374
 Atoi 374
 Atol 374

B

BattCurrentValue 348
 BattReadChargingStatus 348
 BroadcastMsg2Cont 34
 Bsearch 374
 btnClearCaption 48
 btnFrame 48
 btnGetCaption 48
 btnGetCheckBoxState 48
 btnNew 49
 btnNew3DEx 49
 btnNewEx 50
 btnSetBtnGroup 50
 btnSetCaption 51
 btnSetCheckBoxState 51
 btnSetRadioBtnGroup 51
 btnSetRepeat 52
 btnSetStrPosition 51
 btnSetThreeD 52

C

CalcWeekDay 374
 CalcWeekDay2 375
 CalcYearDay 375

Calibration 332
 Calloc 375
 Ceil 375
 ChangeActiveContainer 317
 ChangeActiveContainer 317
 ClearActiveParent 34
 ClearHookUIMsg 34
 Clock 375
 comboAddItem 120
 comboAddItemArray 120
 comboGetItemActive 120
 comboGetItemArray 121
 comboGetItemId 121
 comboGetItemText 121
 comboLastActive 122
 comboMoveLine 122
 comboMovePage 122
 comboNew 123
 comboNewEx 123
 comboRealHeight 124
 comboRemoveItem 124
 comboRemoveItemAll 124
 comboSetItemActive 125
 comboTotalItem 125
 ContAddObj 317
 ContGetObj 317
 ContRedraw 317
 ContRemoveObj 318
 ContSetTitle 318
 Cos 375
 Cosh 376
 CoTan 376
 CPClean 357
 CPGetData 357
 CPGetDataInfo 357
 CPPutData 357
 CTime 376

D

DbAppend 183
 DbCloseDb 183
 DbCreateDb 183
 DbCreateNdx 183
 DbDelete 184
 DbDeleteAndPack 184
 DbDeleteDb 184
 DbDeleteNdx 185
 DbGotoBottom 185
 DbGotoTop 185
 DbInitial 186
 DblsDeleted 186
 DbNameOfField 186
 DbNext 186
 DbNumberOfField 187

DbNumberOfRecord 187
 DbNumberOfRecordBykey 187
 DbOffsetBykey 188
 DbOpenDb 188
 DbOpenDbWithMulNdx 188
 DbOpenDbWithNdx 189
 DbPack 189
 DbPackEx 189
 DbPrev 189
 DbRead 190
 DbRelease 190
 DbSearch 190
 DbSearchFirst 191
 DbSearchLast 191
 DbSeek 192
 DbSeekBykey 192
 DbSetMasterNdx 192
 DbSizeOfField 193
 DbSizeOfRecord 193
 DbUnDelete 193
 DbWrite 194
 DiffTime 376
 DisableTimerMsg 323
 Div 376
 DMAppend 197
 DMCreateNDX 197
 DMCreateNDX2 197
 DMCreateNDX3 198
 DMDBCclose 198
 DMDBCcreate 198
 DMDBDelete 199
 DMDBOpen 199
 DMDelete 199
 DMDeleteNDX 200
 DMFlush 200
 DMGetFieldNoByName 200
 DMGetLastErrorCode 201
 DMGotoBottom 201
 DMGotoTop 201
 DMInitial 202
 DMNameOfField 202
 DMNextWithNDX 202
 DMNumberOfField 202
 DMNumberOfRecord 203
 DMNumberOfRecordBykey 203
 DMOffsetBykey 204
 DMOpenDBWithMulNDX 204
 DMOpenDBWithNDX 204
 DMPack 205
 DMPositionOfNDX 205
 DMPrevWithNDX 205
 DMRead 206
 DMRelease 206
 DMSearchFirstWithNDX 206
 DMSearchLastWithNDX 207
 DMSearchWithNDX 207
 DMSeek 208
 DMSeekWithNDX 208

DMSeekWithNDXBykey 208
 DMSetMasterNDX 209
 DMSizeOfField 209
 DMSizeOfLengthTable 210
 DMSizeOfRecord 210
 DMTypeOfField 210
 DMWrite 211
 DR_DeLP 178
 DR_GetP 179
 DR_LockP 179
 DR_MaxBlock 179
 DR_NewP 180
 DR_RenameP 180
 DR_ResizeP 180
 DR_SizeP 181
 dw2Str 376

E

EnableAutoSleep 323
 EnableDirectMsg 34
 EnableKeyRepeatMsg 330
 EnablePenDoubleClick 332
 EnablePenMoveMsg 332
 EnablePenRepeatMsg 333
 EnableTimerMsg 323
 EnhGetState 366
 EnhRecognize 366
 EnhSetState 366
 EnterCriticalSection 318
 Exp 376

F

Fabs 377
 FileAttrib 166
 FileClose 166
 FileDelete 166
 FileDeleteDir 167
 FileDiskAbort 167
 FileDiskClose 167
 FileDiskDefrag 168
 FileDiskFormat 168
 FileDiskOpen 168
 FileFlush 169
 FileGetCurrentDir 169
 FileGetDefaultDriver 169
 FileGetDone 170
 FileGetFirst 170
 FileGetFreeSpace 170
 FileGetFreeSpaceEx 171
 FileGetNext 171
 FileGetTime 171
 FileNewDir 172
 FileOpen 172
 FileRead 173
 FileReadEx 173
 FileRename 173
 FileRewind 174

FileSeek 174
 FileSetCurrentDir 174
 FileSetDefaultDriver 175
 FileSetTime 175
 FileTell 176
 FileTruncate 176
 FileWrite 176
 FileWriteEx 176
 fldAddRecord 152
 fldClearReserve 152
 fldDelAllRecord 152
 fldDelRecord 152
 fldEnableRepeat 153
 fldGetRecord 153
 fldGetRecordReserve 153
 fldGetRecordReserveNum 154
 fldGetTop 154
 fldGetWidth 154
 fldMove 155
 fldMoveLine 155
 fldMovePage 155
 fldNew 156
 fldNewEx 156
 fldSetRecordReserve 156
 fldSetReserve 157
 fldSetSort 157
 fldSetSpecialReserve 158
 fldSetTop 158
 fldSetWidth 157
 fldTotalRecord 158
 FloatBox 107
 Floor 377
 Fmod 377
 ForwardMsg2Cont 35
 Free 377
 Frexp 377

G

GetAutoSleepTime 323
 GetCoverRect 35
 GetCurrentKeyState 330
 GetDiskVolumeInfo 177
 GetFileLastError 177
 GetFilename 160
 GetFilenameEx 160
 GetItemString 55
 GetMsg 327
 GetMsgNoWait 327
 GetUILastError 35
 GmBmpCopy 214
 GmBmpCopyEx 214
 GmCalcStrWidth 214
 GmCaptureBmp 215
 GmCaptureGray4Bmp 215
 GmCheckBmpSize 216
 GmCheckGray4BmpSize 215
 GmContrastDecrease 216

GmContrastGetValue 216
 GmContrastIncrease 217
 GmCreatePenQuickVal 217
 GmDrawBmp 217
 GmDrawBmpEx 217
 GmDrawDot 218
 GmDrawEllip 218
 GmDrawLine 218
 GmDrawRect 219
 GmDrawSymbol 219
 GmDrawSymbolCnt 219
 GmFillArea 220
 GmFillEllip 220
 GmFillRect 220
 GmFillScreen 221
 GmGetCharWidth 221
 GmGetCurBlinkRef 221
 GmGetCurPos 222
 GmGetDrawPage 222
 GmGetFontHeight 222
 GmGetPalette 223
 GmGetPalette4 223
 GmGetPenPos 223
 GmGetScreen 223
 GmGetScreenSize 224
 GmGetStartAddr 224
 GmGetStrWidth 224
 GmGetStrWidthCnt 225
 GmGrayArea 225
 GmGrayScreen 225
 GmIsCurActive 225
 GmLineTo 226
 GmLoadBmp 226
 GmMagnifyBmp 226
 GmMoveTo 227
 GmPutScreen 227
 GmReduceBmp 227
 GmReleaseBmp 228
 GmRevertRect 228
 GmSetBrushPattern 228
 GmSetCurBlinkRef 229
 GmSetCurBlinkTime 229
 GmSetCurPos 229
 GmSetCurWH 229
 GmSetCustomPattern 230
 GmSetDrawPage 230
 GmSetFont 230
 GmSetPalette 231
 GmSetPalette4 231
 GmSetPalettes 231
 GmSetPenBkColor 231
 GmSetPenBkColorRGB 232
 GmSetPenColor 232
 GmSetPenColorRGB 232
 GmSetPenMode 233
 GmSetPenPattern 233
 GmSetPenPos 233
 GmSetPenQuick 234

GmSetPenQuickEx 234
 GmSetPenWidth 234
 GmSetSymbol 235
 GmSetTextClip 234
 GmShiftScrn 235
 GmShowCur 235
 GmTextOut 236
 GmTextOutCnt 236
 GmTextOutLF 236
 GmTime 377
 GmVPageCopy 237
 GmVPageCopyEx 237
 GmVPageCreate 238
 GmVPageDelete 238

H

HookContMsg 318
 HookPopContMsg 319
 HookUIMsg 35
 hregGetDefPara 366
 hregNewEx 367
 hregSetDefPenWidth 367
 hregSetDefTimeOut 367
 Hypot 377

I

Index 388
 Ir_AdvanceCredit 268
 Ir_Bind 268
 Ir_ConnectIrLap269
 Ir_ConnectReq 269
 Ir_ConnectRsp 269
 Ir_DataReq 270
 Ir_DisconnectIrLap 270
 Ir_DiscoverReq 270
 Ir_End 271
 Ir_IAS_Add 271
 Ir_IAS_GetInteger 271
 Ir_IAS_GetIntLsap 272
 Ir_IAS_GetObjectID 272
 Ir_IAS_GetOctetString 272
 Ir_IAS_GetOctetStringLen 272
 Ir_IAS_GetType 273
 Ir_IAS_GetUserString 273
 Ir_IAS_GetUserStringCharSet 273
 Ir_IAS_GetUserStringLen 274
 Ir_IAS_Next 274
 Ir_IAS_Query 274
 Ir_IAS_SetDeviceName 274
 Ir_IAS_StartResult 275
 Ir_IsIrLapConnected 275
 Ir_IsMediaBusy 275
 Ir_IsNoProgress 275
 Ir_IsRemoteBusy 276
 Ir_LocalBusy 276
 Ir_MaxRxSize 276
 Ir_MaxTxSize 277

Ir_SetConTypeLMP 277
 Ir_SetConTypeTTP 277
 Ir_SetDeviceInfo 277
 Ir_Start 278
 Ir_TestReq 278
 Ir_Unbind 278
 IrLptBusy 264
 IrLptEnd 264
 IrLptSend 264
 IrLptStart 264
 IrObexAutoRecvSwitch 265
 IrObexConnect 265
 IrObexDisconnect 265
 IrObexEnd266
 IrObexGetAutoRecvMode 266
 IrObexGetData 266
 IrObexIsConnected 266
 IrObexRegister 267
 IrObexSendData 267
 IrObexSendObject 267
 IrObexSetAutoRecvMode 267
 IrObexStart 268
 Isalnum 377
 Isalpha 378
 Isascii 378
 Iscntrl 378
 Isdigit 378
 Isgraph 378
 Isleadbyte 378
 Islower 378
 IsOnCoverArea 36
 IsOnCoverRect 36
 IsPopContActive 319
 Isprint 378
 Ispunct 379
 Isspace 379
 IsTimerMsgOn 324
 Isupper 379
 Isxdigit 379

J

K

kbdNew 110
 kbdNewEx110
 kbdUnDefWordZone 110
 Keyboard 109

L

labClearCaption44
 labGetCaption 44
 labNew 44
 labNew3DEx 45
 Labs 379
 labSetCaption 45
 labSetThreeD 46

Ldexp 379
 Ldiv 379
 LeaveCriticalSection 319
 listAddItem 55
 listAddItemArray 55
 listAddSpaceBar 55
 listGetHotKeyText 56
 listGetItemActive 56
 listGetItemArray 56
 listGetItemPosition 57
 listGetItemText 57
 listGetTopItem 57
 listIsGrayItem 58
 listLastActive 58
 listMoveLine 58
 listMovePage 59
 listNew 59
 listNewEx 60
 listPopUp 60
 listPopUpEx 61
 listPopUpPro 62
 listRealHeight 62
 listRemoveItem 62
 listRemoveItemAll 63
 listSetHotKeyText 63
 listSetItemActive 63
 listSetItemPosition 64
 listSetItemText 64
 listSetTopItem 64
 listTotalItem 65
 listTotItem 65
 Localtime 379
 Log 379
 Log10 380
 Longjmp 380
 Lunar2Solar 380

M

Malloc 380
 medActiveBackSpace 138
 medAppendStr 138
 medAppendStrEx 138
 medDeleteStr 138
 medGetCurlLineNum 139
 medGetCursorCurrentPos 139
 medGetEditText 139
 medGetMaxCnt 140
 medGetModify 140
 medGetSelect 140
 medGetTextLength 140
 medGetTotalLine 141
 medGetViewPos 141
 medInfoA 141
 medInfoB 142
 medInfoC 142
 medInsertStr 142
 medIsOnFocus 143

medMove 143
 medNew 143
 medNewEx 144
 medScrollDown 144
 medScrollPageDown 144
 medScrollPageUp 145
 medScrollUp 145
 medSelectAll 145
 medSetCurPos 146
 medSetEditFont 146
 medSetEditText 146
 medSetEditTextEx 147
 medSetFocus 147
 medSetMaxCnt 147
 medSetModify 147
 medSetSelect 148
 medSetViewPos 148
 medSetViewPosEx 148
 Memchr 380
 Memcmp 381
 Memcpy 381
 Memmove 381
 Memset 381
 menuAddItem 97
 menuAddItemArray 97
 menuAddMenuHelp 97
 menuAddMenuItem 97
 menuAddMenuSection 98
 menuAddSection 98
 menuAddSectionHelp 98
 menuGetItem 99
 menuGetItemArray 99
 menuGetItemIndex 99
 menuGetItemState 100
 menuGetSectionIndex 100
 menuGrayItem 100
 menuNew 101
 menuNewEx 101
 menuPopup 102
 menuPopupMenuBar 102
 menuRemoveItem 102
 menuRemoveItemAll 103
 menuRemoveSection 103
 menuSectionFocus 103
 menuSendHotKey 104
 menuSetActiveMenu 104
 menuSetItemState 104
 menuSetSectionFocus 104
 Mktime 381
 MmDelH 343
 MmDelP 343
 MmLockH 343
 MmMaxFreeBlock 345
 MmNewH 343
 MmNewP 344
 MmResizeH 344
 MmResizeP 344
 MmSizeofH 345

MmSizeofP 345
 MmTotalFreeMem 345
 MmUnlockH 346
 Modf 381
 MsgBox 107

N

Net_Abort 304
 Net_Accept 304
 Net_CloseSocket 304
 Net_Connect 305
 Net_Dial 305
 Net_GetHostByAddr 305
 Net_GetHostByName 305
 Net_GetPop3Info 306
 Net_GetSmtpInfo 306
 Net_Listen 306
 Net_Pop3Close 307
 Net_Pop3Commit 307
 Net_Pop3Delete 307
 Net_Pop3List 307
 Net_Pop3Logoff 308
 Net_Pop3Logon 308
 Net_Pop3Noop 308
 Net_Pop3Reset 308
 Net_Pop3RetrieveMsg 309
 Net_Pop3State 309
 Net_Pop3Top 309
 Net_Pop3Uidl 310
 Net_Receive 310
 Net_ReceiveFrom 310
 Net_Select 311
 Net_Send 311
 Net_SendTo 311
 Net_SmtpClose 312
 Net_SmtpData 312
 Net_SmtpExpn 312
 Net_SmtpHello 312
 Net_SmtpHelp 313
 Net_SmtpMail 313
 Net_SmtpMessage 313
 Net_SmtpNoop 314
 Net_SmtpQuit 314
 Net_SmtpRcpt 314
 Net_SmtpReset 315
 Net_SmtpStart 314
 Net_SmtpVerify 315
 NetAbort 286
 NetAccept 286
 NetAddRoute 286
 NetBind 287
 NetCheckFD 287
 NetCloseSocket 287
 NetConnect 288
 NetDial 288
 NetGetDialInfo 288
 NetGetHostByAddr 289

NetGetHostByName 289
 NetGetIpAddr 290
 NetGetMailInfo 289
 NetGetPeerName 289
 NetGetSocketOption 290
 NetHangUp 291
 NetInitFD 291
 NetIsConnected 291
 NetListen 291
 NetPop3Close 292
 NetPop3Commit 292
 NetPop3Delete 292
 NetPop3List 293
 NetPop3Logoff 293
 NetPop3Logon 293
 NetPop3Noop 294
 NetPop3Reset 294
 NetPop3RetrieveCallBack 294
 NetPop3RetrieveMsg 294
 NetPop3State 295
 NetPop3Top 295
 NetPop3Uidl 295
 NetReceive 296
 NetReceiveFrom 296
 NetResetFD 296
 NetSelect 297
 NetSend 297
 NetSendTo 297
 NetSetBlock 298
 NetSetDialInfo 298
 NetSetFD 298
 NetSetMailInfo 299
 NetSetSocketOption 299
 NetSmtpClose 300
 NetSmtpData 300
 NetSmtpExpn 300
 NetSmtpHello 301
 NetSmtpHelp 301
 NetSmtpMail 301
 NetSmtpMessage 302
 NetSmtpNoop 302
 NetSmtpQuit 302
 NetSmtpRcpt 302
 NetSmtpReset 303
 NetSmtpStart 303
 NetSmtpVerify 303
 NetSocket 299
 NumberOfDir 178
 NumberOfFile 178

O

OSVersionStr 372

P

PbxHregEnableInk 367
 PbxHregGetParam 368
 PbxHregInit 368

PbxHregPlugIn 368
 PbxHregPlugOut 369
 PbxHregSetParam 369
 PopContainer 320
 PopContainerEnd 320
 Pow 382
 ProcessUIMsgDirect 36
 progsChangeMaxRange 92
 progsEnableTimer 92
 progsGetIncVal 92
 progsGetPercent 92
 progsGetTimer 93
 progsIncrease 93
 progsNew 93
 progsNewEx 94
 progsReset 94
 progsSetIncVal 94
 progsSetPercent 95
 progsSetTimer 95

Q

Qsort 382

R

Rand382
 Realloc 382
 ReleaseCoverRect 37
 RIsLastCoverRect 37
 RtcAddAlarm 351
 RtcDelAlarm 351
 RtcDelAllAlarm 351
 RtcDisableStopWatch 353
 RtcEnableStopWatch 353
 RtcGetDate 353
 RtcGetDateTime 353
 RtcGetTime 354
 RtcGetTime2 354
 RtcSetDate 354
 RtcSetTime 355

S

sbarAddTotalSize 113
 sbarGetMaxPos 113
 sbarGetPageSize 113
 sbarGetPos 113
 sbarGetTotalSize 114
 sbarMove 114
 sbarMoveLine 114
 sbarMovePage 115
 sbarNew 115
 sbarNewEx 116
 sbarSetPageSize 116
 sbarSetPos 116
 sbarSetTotalSize 117
 sbarShowAR 117
 sbarSubTotalSize 117

SearchComplete 160
 SearchDrawTitle 161
 SearchSaveMatched 161
 SearchSetDefMode 161
 SearchStrInStr 162
 sedActiveBackSpace 129
 sedCutSelect 129
 sedDeatchData 129
 sedDeleteStr 129
 sedGetEditText 130
 sedGetMaxCnt 130
 sedGetModify 130
 sedGetNumericFmt 130
 sedGetSelect 131
 sedGetTextLength 131
 sedInfoA 131
 sedInfoB 132
 sedMove 132
 sedNew 132
 sedNewEx 133
 sedSelectAll 133
 sedSetBeginPos 134
 sedSetCurPos 134
 sedSetEditText 134
 sedSetEditTextEx 135
 sedSetFocus 135
 sedSetMaxCnt 135
 sedSetModify 136
 sedSetNumericFmt 135
 sedSetSelect 136
 SendMsg 327
 SendMsgTop 328
 SetActiveContainer 320
 SetActiveParent 37
 SetAutoSleepTime 324
 SetContainerEntry 320
 SetCoverRect 37
 SetDiskVolumeLabel 178
 SetDoubleClickTick 333
 SetGmRefTime 382
 Setjmp 382
 SetKeyRepeatTick 330
 SetPenMoveTick 333
 SetPenRepeatTick 333
 SetTabContEntry 321
 SetTimerMsg 324
 SetTitleTabContEntry 321
 Sin 382
 Sinh 383
 SiClose 360
 SiCloseAll 360
 Sleeping 324
 SiFunc 360
 SiInit 360
 SiOpen 361
 sndClose 240
 sndControl 240
 sndGetFileInfo 241

sndGetLastError 241
 sndGetVolume 241
 sndInit 242
 sndPause 242
 SndPlay 242
 sndQueryPosition 243
 sndSetDirection 243
 sndSetMode 243
 sndSetPosition 244
 sndSetSpeed 244
 sndSetVolume 244
 sndStart 244
 sndStatus 245
 sndStop 246
 Solar2Lunar 383
 Sprintf 383
 Sqrt 383
 Srand 383
 Str2lower 384
 Str2upper 384
 Strcat 384
 Strchr 384
 Strcmp 384
 Strcpy 384
 Strcspn 85
 Strerror 385
 Stricmp 385
 Strlen 385
 Strncat 385
 Strncmp 385
 Strncpy 385
 Strpbrk 385
 Strrchr 386
 Strspn 386
 Strstr 386
 Strtod 386
 Strtok 386
 Strtol 386
 Strtoul 386
 SysDefKB 111
 SysEngKB 111
 SysGetDate 162
 SysGetHwInfo 363
 SysGetIOInfo 363
 SysGetOsInfo 364
 SysGetTime 162
 SysProcess 328
 SysSearch 163

T

tablBmpNew 67
 tablBmpOut 67
 tablClearReserve 67
 tablGetColumn 67
 tablGetReserve 68
 tablGetRow 68
 tablGridToCoord 73

tablInsertColumn 68
 tablInsertRow 69
 tablMoveColumn 69
 tablMovePageColumn 69
 tablMovePageRow 70
 tablMoveRow 70
 tablNew 70
 tablNewEx 71
 tablPopUp 71
 tablRemoveColumn 72
 tablRemoveRow 72
 tablResetActiveZone 72
 tablSetActive 73
 tablSetActiveZone 73
 tablSetActiveZoneEx 74
 tablSetNoActive 74
 tablSetReserve 74
 tablSubNew 75
 tablTextOut 75
 tablTextOutEx 75
 Tan 387
 Tanh 387
 tbrAddBtn 78
 tbrAddButton 78
 tbrGetBtnState 78
 tbrGetButtonBorder 79
 tbrGetButtonState 79
 tbrGetShowTipPos 79
 tbrGetTipBorder 80
 tbrGetTipFont 80
 tbrGetTipHandle 80
 tbrGetTipText 80
 tbrNew 81
 tbrNewEx 81
 tbrRemoveBtn 82
 tbrSetBtnState 82
 tbrSetButtonBorder 82
 tbrSetButtonState 83
 tbrSetShowTipPos 83
 tbrSetTipBorder 83
 tbrSetTipFont 84
 tbrSetTipText 84
 tbrSetToolTip 84
 tbrSubIDtoIndex 84
 tbrToolBarIsEnable 85
 Time 387
 Toascii 387
 Tolower 387
 ToneBeep 248
 ToneClose 248
 ToneGetVolume 248
 ToneOpen 248
 TonePlayMIDIFile 248
 TonePlayMusic 249
 ToneSetFrequency 249
 ToneSetValue 249
 ToneSetVolume 250
 ToneStopFrequency 250

ToneStopMusic 251
 ToneStopValue 251
 Toupper 387
 TrapGetEntry 371
 TrapGetLastApiNum 371
 TrapHookEntry 371
 treeNew 127
 treeRealHeight 127

U

uiClean 38
 uiDelete 38
 uiDeleteTimer 38
 uiEnable 38
 uiEnableTimer 39
 UIExtraProcess 39
 uiGetActive 40
 uilsActive 39
 uilsEnable 40
 uilsVisible 40
 uiNewTimer 40
 uiRedraw 41
 uiSetActive 41
 uiSetInterval 41
 uiSetLastError 42
 uiSetVisible 42
 UrtCancelSendData 254
 UrtCancelXmodemGet 254
 UrtCancelXmodemSend 254
 UrtClearRxFIFO 254
 UrtClose 254
 UrtCTSSStatus 255
 UrtImmGetData 255

UrtImmSendData 255
 UrtIsRxReady 256
 UrtIsTxAvailable 256
 UrtIsTxEmpty 256
 UrtOpen 256
 UrtRcvModeBegin 257
 UrtRcvModeEnd 257
 UrtRcvReset 257
 UrtSendData 257
 UrtSet 258
 UrtSetRTS 258
 UrtSetWay259
 UrtXGetContinue 259
 UrtXmodemGet 259
 UrtXmodemSend 259

V

W

wbClean 87
 wbCopyBmp 87
 wbNew 87
 wbNewEx 87
 wbPasteBmp 88
 wbSetDrawStyle 88
 wbSetPenColor 88
 wbSetPenColorEx 89
 wbSetPenMode 89
 wbSetPenStyle 90
 wbSetPenWidth 90

XYZ