Exit

Contents

General

Command Reference

Data Type Reference

Constants and Macros

Exit

^^^

Contents

How to Create a Module

Exit

^^^

Contents

Callback Procedures

Exit

^^^

Contents

Command Handling Commands

Exit

^^^

Contents

Window and Display Commands

Exit

^^^

Contents

Dialog Window Commands

Dialog Window <

- TosoDialogHelpMessage
- TosoDialogEnterIdle
- TosoDialogComboboxAdjust
- TosoDialogCustomButtonColor
- TosoDialogCustomButtonText
- TosoDialogCustomButtonIcon
- TosoDialogCustomListboxColor
- TosoDialogCenter

- TosoDialogSelection
- TosoDialogColor
- TosoDialogProperty
- TosoDialogXProperty
- TosoDialogFontDef
- TosoDialogDimLine
- TosoDialogDimSmall
- TosoDialogDimLarge
- TosoDialogTextStandard
- TosoDialogTextFrame
- TosoDialogBlock

- TosoDialogSelectLine
- TosoDialogSelectMultiLine
- TosoDialogSelectSystem
- TosoDialogSelectHatch
- TosoDialogSelectLayer
- TosoDialogSelectPen

- TosoDialogUpdateGuide
- TosoDialogShowProgress
- TosoDialogUpdateProgress
- TosoDialogHideProgress
- TosoDialogIsCanceled

Exit

^^^

Contents

Element Handling Commands

Exit

^^^

Contents

Unit Creation Commands

TosoBlockOpen
TosoBlockGetAddress
TosoBlockAddAttribute
TosoBlockAddEnd
TosoBlockInsert
TosoBlockClose
TosoBlockFastInsert

TosoGroupOpen
TosoGroupGetAddress
TosoGroupAddEnd
TosoGroupInsert
TosoGroupClose
TosoGroupFastInsert

Exit

^^^

Contents

Unit Editing Commands

Exit

^^^

Contents

Unit Enumeration Commands


Contents

File Handling Commands

TosoFileReadDouble
TosoFileReadCommaDouble
TosoFileReadColorref
TosoFileReadCommaColorref
TosoFileReadFontdef
TosoFileReadCommaFontdef
TosoFileReadXProperty
TosoFileReadCommaXProperty
TosoFileReadProperty
TosoFileReadCommaProperty
TosoFileReadDimLine
TosoFileReadCommaDimLine
TosoFileReadDimSmall
TosoFileReadCommaDimSmall
TosoFileReadDimLarge
TosoFileReadCommaDimLarge
TosoFileReadTextStandard
TosoFileReadCommaTextStandard
TosoFileReadTextFrame
TosoFileReadCommaTextFrame
TosoFileReadTextReference
TosoFileReadCommaTextReference
TosoFileReadClipSurface
TosoFileReadCommaClipSurface
TosoFileReadBitmapRef
TosoFileReadCommaBitmapRef
TosoFileReadHeader
TosoFileReadEndOfFile
TosoFileReadDelimiters
TosoFileReadError
TosoFileReadTotalSize
TosoFileReadCurrentSize
TosoFileReadCurrentLine

TosoFileWriteInitNull
TosoFileWriteInitDisk
TosoFileWriteInitMemory
TosoFileWriteData
TosoFileWriteTextData
TosoFileWriteFlush
TosoFileWriteExit
TosoFileWriteNewline
TosoFileWriteComma
TosoFileWriteSemi
TosoFileWriteComment
TosoFileWriteKeyword
TosoFileWriteString
TosoFileWriteCommaString
TosoFileWriteBinary
TosoFileWriteCommaBinary
TosoFileWriteBool

TosoFileWriteCommaBool
TosoFileWriteShort
TosoFileWriteCommaShort
TosoFileWriteInt
TosoFileWriteCommaInt
TosoFileWriteDouble
TosoFileWriteCommaDouble
TosoFileWriteColorref
TosoFileWriteCommaColorref
TosoFileWriteFontdef
TosoFileWriteCommaFontdef
TosoFileWriteXProperty
TosoFileWriteCommaXProperty
TosoFileWriteProperty
TosoFileWriteCommaProperty
TosoFileWriteDimLine
TosoFileWriteCommaDimLine
TosoFileWriteDimSmall
TosoFileWriteCommaDimSmall
TosoFileWriteDimLarge
TosoFileWriteCommaDimLarge
TosoFileWriteTextStandard
TosoFileWriteCommaTextStandard
TosoFileWriteTextFrame
TosoFileWriteCommaTextFrame
TosoFileWriteTextReference
TosoFileWriteCommaTextReference
TosoFileWriteClipSurface
TosoFileWriteCommaClipSurface
TosoFileWriteBitmapRef
TosoFileWriteCommaBitmapRef
TosoFileWriteHeader
TosoFileWriteEndOfFile
TosoFileWriteZeros
TosoFileWriteDelimiters
TosoFileWriteError
TosoFileWriteCurrentSize
TosoFileWriteCurrentLine


Contents

Profile and Settings Commands


Contents

Geometry and Conversion Commands


Contents

Miscellaneous Commands

Contents

Alphabetical Command List

TosoDialogSelectHatch
TosoDialogSelection
TosoDialogSelectLayer
TosoDialogSelectLine
TosoDialogSelectMultiLine
TosoDialogSelectPen
TosoDialogSelectSystem
TosoDialogShowProgress
TosoDialogTextFrame
TosoDialogTextStandard
TosoDialogUpdateGuide
TosoDialogUpdateProgress
TosoDialogXProperty
TosoDrawingGetActive
TosoDrawingGetNumber
TosoDrawingGetInfo
TosoDrawingNewFile
TosoDrawingOpenFile
TosoDrawingSaveFile
TosoDrawingSetActive
TosoDrawLineDef
TosoDrawNewObjects
TosoDrawBlock
TosoDrawWindowAll
TosoDrawWindowArea
TosoDrawWindowSelection
TosoEditDelete
TosoEditIdentCount
TosoEditIdentEnumModify
TosoEditIdentMatrix
TosoEditPointsCount
TosoEditPointsMatrix
TosoEnumAttribProc
TosoEnumerateAll
TosoEnumerateBlock
TosoEnumerateBlockAttrib
TosoEnumerateChar
TosoEnumerateIdent
TosoEnumerateInstanceAttrib
TosoEnumerateLibrary
TosoEnumeratePoints
TosoEnumerateUnit
TosoEnumIdentProc
TosoEnumObjectProc
TosoEnumPointsProc
TosoExportGetData
TosoExportGetNumber
TosoFileClose
TosoFileCopy
TosoFileCreate
TosoFileDelete

TosoFileExist
TosoFileFullPath
TosoFileGetExtension
TosoFileOpen
TosoFileRead
TosoFileReadBinary
TosoFileReadBitmapRef
TosoFileReadBool
TosoFileReadChar
TosoFileReadClipSurface
TosoFileReadColorref
TosoFileReadComma
TosoFileReadCommaBinary
TosoFileReadCommaBitmapRef
TosoFileReadCommaBool
TosoFileReadCommaClipSurface
TosoFileReadCommaColorref
TosoFileReadCommaDimLarge
TosoFileReadCommaDimLine
TosoFileReadCommaDimSmall
TosoFileReadCommaDouble
TosoFileReadCommaFontdef
TosoFileReadCommaInt
TosoFileReadCommaProperty
TosoFileReadCommaShort
TosoFileReadCommaString
TosoFileReadCommaTextFrame
TosoFileReadCommaTextReference
TosoFileReadCommaTextStandard
TosoFileReadCommaXProperty
TosoFileReadContinue
TosoFileReadCurrentLine
TosoFileReadCurrentSize
TosoFileReadData
TosoFileReadDelimiters
TosoFileReadDimLarge
TosoFileReadDimLine
TosoFileReadDimSmall
TosoFileReadDouble
TosoFileReadEndOfFile
TosoFileReadError
TosoFileReadExit
TosoFileReadFontdef
TosoFileReadHeader
TosoFileReadInitDisk
TosoFileReadInitMemory
TosoFileReadInt
TosoFileReadKeyword
TosoFileReadLastKeyword
TosoFileReadLine
TosoFileReadNextKeyword

TosoFileReadNextSection
TosoFileReadProperty
TosoFileReadTextFrame
TosoFileReadTextReference
TosoFileReadTextStandard
TosoFileReadTotalSize
TosoFileReadSemi
TosoFileReadShort
TosoFileReadString
TosoFileReadXProperty
TosoFileSetExtension
TosoFileSetPointer
TosoFileShortName
TosoFileSize
TosoFileSplitName
TosoFileWrite
TosoFileWriteBinary
TosoFileWriteBitmapRef
TosoFileWriteBool
TosoFileWriteClipSurface
TosoFileWriteComma
TosoFileWriteCommaBinary
TosoFileWriteCommaBitmapRef
TosoFileWriteCommaBool
TosoFileWriteCommaClipSurface
TosoFileWriteCommaColorref
TosoFileWriteCommaDimLarge
TosoFileWriteCommaDimLine
TosoFileWriteCommaDimSmall
TosoFileWriteCommaDouble
TosoFileWriteCommaFontdef
TosoFileWriteCommaInt
TosoFileWriteCommaProperty
TosoFileWriteCommaShort
TosoFileWriteCommaString
TosoFileWriteCommaTextFrame
TosoFileWriteCommaTextReference
TosoFileWriteCommaTextStandard
TosoFileWriteCommaXProperty
TosoFileWriteComment
TosoFileWriteColorref
TosoFileWriteCurrentLine
TosoFileWriteCurrentSize
TosoFileWriteData
TosoFileWriteDelimiters
TosoFileWriteDimLarge
TosoFileWriteDimLine
TosoFileWriteDimSmall
TosoFileWriteDouble
TosoFileWriteEndOfFile
TosoFileWriteError

TosoFileWriteExit
TosoFileWriteFlush
TosoFileWriteFontdef
TosoFileWriteHeader
TosoFileWriteInitNull
TosoFileWriteInitDisk
TosoFileWriteInitMemory
TosoFileWriteInt
TosoFileWriteKeyword
TosoFileWriteNewline
TosoFileWriteProperty
TosoFileWriteSemi
TosoFileWriteShort
TosoFileWriteString
TosoFileWriteTextData
TosoFileWriteTextFrame
TosoFileWriteTextReference
TosoFileWriteTextStandard
TosoFileWriteXProperty
TosoFileWriteZeros
TosoGeoCircumcircle
TosoGeoDistance
TosoGeoIncircle
TosoGeoIntersection
TosoGeoPerpendicular
TosoGeoRadiusFit
TosoGeoTangent
TosoGetCommandIcon
TosoGetCommandTitle
TosoGetStandardIcon
TosoGroupAddEnd
TosoGroupClose
TosoGroupFastInsert
TosoGroupGetAddress
TosoGroupInsert
TosoGroupOpen
TosoHatchGetActive
TosoHatchGetCurrentDef
TosoHatchGetDef
TosoHatchGetOrigin
TosoHatchSetActive
TosoHatchSetDef
TosoHookKeyEnd
TosoHookKeyProc
TosoHookKeyStart
TosoHookMouseEnd
TosoHookMouseProc
TosoHookMouseStart
TosoHookPositionEnd
TosoHookPositionProc
TosoHookPositionStart

TosoImportGetData
TosoImportGetNumber
TosoInitDimLarge
TosoInitDimLine
TosoInitDimSmall
TosoInitFontDef
TosoInitProperty
TosoInitTextFrame
TosoInitTextStandard
TosoInitXProperty
TosoInputCancelProc
TosoInputDisplayProc
TosoInputDrawBezier
TosoInputDrawCircle
TosoInputDrawCircleArc
TosoInputDrawEllipse
TosoInputDrawEllipseArc
TosoInputDrawEndless
TosoInputDrawFrame
TosoInputDrawLine
TosoInputDrawPoint
TosoInputDrawReference
TosoInputFinishProc
TosoInputGetGeneratedSurface
TosoInputGetIdentAddress
TosoInputGetIdentData
TosoInputGetIdentObject
TosoInputParameterProc
TosoInputPointExitProc
TosoInputPointInitProc
TosoInputPointMoveProc
TosoInstanceAddAttribute
TosoInstanceAddEnd
TosoInstanceClose
TosoInstanceFastInsert
TosoInstanceGetAddress
TosoInstanceInsert
TosoInstanceOpen
TosoLayerGetActive
TosoLayerGetCurrentDef
TosoLayerGetDef
TosoLayerSetActive
TosoLayerSetDef
TosoLibraryGetNumber
TosoLibraryGetInfo
TosoLineGetDef
TosoLineSetDef
TosoMatrixAssemble
TosoMatrixInit
TosoMatrixInvert
TosoMatrixMove

TosoMatrixMultiply
TosoMatrixRotate
TosoMatrixScale
TosoMatrixSeparate
TosoMatrixSheer
TosoModuleCommand
TosoModuleExit
TosoModuleGetData
TosoModuleGetNumber
TosoModuleInit
TosoModuleModify
TosoMultiLineGetDef
TosoMultiLineSetDef
TosoObjectAddBitmapRef
TosoObjectAddClipSurface
TosoObjectAddConstant
TosoObjectAddCurve
TosoObjectAddDimLarge
TosoObjectAddDimLine
TosoObjectAddDimSmall
TosoObjectAddEnd
TosoObjectAddOrient
TosoObjectAddPoint
TosoObjectAddTextFrame
TosoObjectAddTextLong
TosoObjectAddTextReference
TosoObjectAddTextShort
TosoObjectAddTextStandard
TosoObjectClose
TosoObjectCopyDataBlocks
TosoObjectFastInsert
TosoObjectGetAddress
TosoObjectInsert
TosoObjectOpen
TosoPageGetDef
TosoPageSetDef
TosoPenGetActive
TosoPenGetCurrentDef
TosoPenGetDef
TosoPenSetActive
TosoPenSetDef
TosoProfileDeleteKey
TosoProfileGetDrawing
TosoProfileSetDrawing
TosoProfileReadData
TosoProfileReadInt
TosoProfileReadKeyClose
TosoProfileReadKeyOpen
TosoProfileReadString
TosoProfileWriteData
TosoProfileWriteInt

TosoProfileWriteKeyClose
TosoProfileWriteKeyOpen
TosoProfileWriteString
TosoSettingGet
TosoSettingSet
TosoSystemGetActive
TosoSystemGetCurrentDef
TosoSystemGetDef
TosoSystemSetActive
TosoSystemSetDef
TosoUndoCancelProcess
TosoUndoFinishProcess
TosoUndoInitProcess
TosoUndoSetPrevious
TosoUndoUpdateLinks
TosoUserAddDataBlock
TosoUserAddEnd
TosoUserClose
TosoUserFastInsert
TosoUserGetAddress
TosoUserInsert
TosoUserOpen
TosoWindowGetActive
TosoWindowGetMode
TosoWindowGetView
TosoWindowSetActive
TosoWindowSetMode
TosoWindowSetViewAll
TosoWindowSetViewArea
TosoWindowSetViewPage
TosoWindowSetViewScale


Contents

Basic Types

Drawing Info Types


Contents

Object Info Types


Contents

Data Block Types


Contents

Unit Types

Contents

Module Interface Types

Contents

Contents

Alphabetical Type List

HATCHDEF
LAYERDEF
LINEDEF
MATRIX
MENU_DATA
MODULE_COMMAND_DATA
MODULE_DATA
MODULE_ID
MODULE_PROC
MULTILINE
MULTILINEDEF
OBJPTR
OFFSET
PAGEDEF
PENDEF
PROPERTY
STRx
SYSTEMDEF
TEXTFRAME
TEXTREFERENCE
TEXTSTANDARD
TOKEN_DATA
UNIT_ANY
UNIT_BLOCK
UNIT_HEADER
UNIT_INSTANCE
UNIT_MEMORY
UNIT_OBJECT
UNIT_USER
WINDOWDEF
XPROPERTY


Contents

Constants


Contents

Macros

Introduction (General)

All TommySoftware® 32bit applications include a powerful software interface (called *Toso Interface* - TommySoftware® Open Software Interface) that allows the integration of external modules. These modules can either be import filters, export filters or command extensions in any form.

The *Toso Interface* offers an easy way to create powerful extensions to TommySoftware® applications. The interface was designed to make it as easy as possible to create modules that fit perfectly into the application's environment. Usually, the user won't realize any substantial difference between using an internal command provided by the application or using an external command provided by a module or filter.

Import and export filters can be created with only basic knowledge of the external file format (usually TVG 4.0, see TVG 4.0 Documentation (TVG40.HLP) for documentation on that file format) and some knowledge about geometrical calculations. You will find that especially creating export filters is very easy - if required, the application does all calculations for you and passes simple lines to the export filter.

Modules, as well as import and export filters, have to have special file names: the module's file name has to end with an underscore `_` (Ansi 95), followed by the extension `.DLL`. All three types of modules have to reside in the application's main path (i.e. in the same directory as the serving application). When starting, the serving application will search for all files matching `*_.DLL` and will try to install those files as modules or filters (see Module Style Guide).

Command extension module file names should start with a three-character identification of the module's creator to avoid duplicate names. You will receive your private three-character identification at the time you contact TommySoftware® in order to receive your unique owner identifier (see Getting Your Private Owner ID).

The *Toso Interface* documentation is basically addressed to C developers with some experience in Win32 programming, i.e. it assumes knowledge of the programming language C and of basic Win32 topics. The *Toso Interface* may also be used with Basic or Pascal compilers that are able to use the supplied `TOSO40.H` and `TOSO40.LIB` files and that can create C compatible dynamic link libraries (DLLs) for Win32. Anyway, we recommend the use of the current version of Microsoft's Visual C++ - the compiler that we do use. It creates fast and robust code, and it has a powerful user interface that allows fast and easy development of complex projects.

If you are using Microsoft's Visual C++ 4.0 or higher, you can use the file `USERTYPE.DAT` that should have come together with `TOSO40.H` and `TOSO40.LIB`. Copy that file into the `\BIN` directory of your compiler's directory and from the next compiler start on, all types and definitions releated to Toso 4.0 will be highlighted in the source editor.

Most types and structures are fully described within this documentation. Some elements, however, will only be described syntactically. For a semantical description and background information on those elements, see the TVG 4.0 documentation in TVG 4.0 Documentation (TVG40.HLP). That documentation is also the place where you will find a complete list of object type handled by the *Toso Interface*, and a list of data blocks required to build these objects.

When referencing variables or data types, they will be displayed in *italic* and colored. Data types will appear in *red*, variables in *violet*. Constants will appear in capitals and YELLOW. Sections containing C source code or names will appear in dark blue, using the font `Courier`.

Technical Support and Information

The documentation of the *Toso Interface* and of the TVG 4.0 file format are supplied "as is", and they are both *only* available in electronic form and *only* in English. Developer support, however, is available in German as well as in English (see Getting Your Private Owner ID).

For technical support please send an e-mail to "support@tommysoftware.com". If you have any other questions please send your e-mail to "sales@tommysoftware.com".

On our World Wide Web site (http://www.tommysoftware.com) you can find the latest versions of the *Toso Interface* documentation and of the TVG 4.0 file format documentation as well as the latest program versions and upgrades, a collection of import/export filters and converters, the CAD/DRAW Tutorial, the CAD/DRAW Tour, additional documentation, utilities, and current information on our products. And please also try out our special Web offers. Don't miss this opportunity to make the most of the Internet!

TommySoftware®

North America, Inc.
648 Ashbury Street
San Francisco, CA 94117
U. S. A.

Phone 1-800-275-9406
Phone (415) 522 0612
Fax (415) 522 0287

CompuServe GO TOSOENG

Germany
Selchower Straße 32
D-12049 Berlin
Germany

Phone +49 30 621 5931

Fax +49 30 621 4064

CompuServe GO TOSOGER

Internet
sales@tommysoftware.com (Sales)
support@tommysoftware.com (Technical Support)
http://www.tommysoftware.com (World Wide Web)

Getting Your Private Owner ID (How to Create a Module)

When creating a "commercial" module using the *Toso Interface*, you will require your personal owner ID and a three-character identification string. Both are required to clearly identify a module and all data created by that module.

Note: If you create "personal" modules that will not be distributed, or if you are only experimenting with the *Toso Interface*, you can use a trial owner ID of `0xffff` and the identification string `XYZ` . Both will never be used by any commercial module, so there will be no conflicts on your machine. Anyway, as soon as you want to pass such a module to friends or colleagues, or even sell it, you should first get and implement your private owner ID to avoid any conflict with other modules!

You will have to state your private owner ID during the initialization of each module and within some structures passed to the serving application. The three-character identification is to be used to build the module's file name. This is necessary because module file names have to be unique. Especially if a module relies on an additional language-dependent DLL (usually named similar to the main DLL), the user will not be able to simply rename modules that use equal file names.

In order to get your private owner ID and the three-character identification, contact TommySoftware® at the following address. Please do *not* contact your local TommySoftware® subsidiary or your dealer, they will not be able to handle the request! Send your request (in English or German) either via fax or via e-mail to:

Developer Support
Fax +49 5302 1613
CompuServe 100024,1536
Internet 100024.1536@compuserve.com

Please include the following data in your request:

- Your complete name

Your company's name

The serial number of your application

Complete mail address

Phone and fax numbers

E-mail address in Internet or CompuServe

It is important that you state either a fax number or an e-mail address in Internet or CompuServe! If you do not, you will not be sent your owner ID! For registered users of a TommySoftware® application including the *Toso Interface*, this service is free of any charge, as long as all communication between you

and TommySoftware® is via fax or e-mail.

If you would prefer to get a specific three-character identification (rather than an identification created by us), please state it. If possible, we will reserve this identification for you! A good idea would be also to include a short description of the module(s) you plan to create. This allows us to inform you if such a module is already available (or in development by someone else), and it keeps us informed about what types of modules are most commonly requested. This will help us to improve the *Toso Interface* to suit exactly your needs!

Modules based on the *Toso Interface* may be developed, distributed and sold royalty-free. If you have developed a module of common interest and would like to have it distributed, contact us! If you want to distribute it on your own, we can at least include a description of it in our module catalogue.

Developer support is not included in the application's support program. Anyway, any developer who received a private owner ID can address requests concerning the *Toso Interface* and its usage to the TommySoftware® Developer Support - in most cases, you will receive a direct response from our support team. Even requests that are not directly answered will be used to build a "Frequently Asked Questions" document that is supplied in each new version of the interface. Again, remember to state either a fax number or an e-mail address in Internet or CompuServe!

For professional developers, we offer a charged phone line to experienced developers and personal training. If you are interested in this advanced support, please state this on your request. We will then supply you with all information about professional developer support.

Related Topics:

- Basic Module Code

External Commands

Export Filters

Import Filters

Handling of Coordinates

Creating Text Objects

Creating Dimension Objects

Creating Groups

Debugging Modules

Module Style Guide

Basic Module Code (How to Create a Module)

The *Toso Interface* supports three types of modules: External Commands, Import Filters and Export Filters. All three module types use similar basic code that performs initialization and termination. For this task, each module supplies 4 to 5 procedures that can be called by the serving application (sometimes referred to as "Callback Procedures"):

DllMain
: This procedure is called at the time the module is loaded into the serving's application memory, i.e. at the time the serving application itself is being loaded. Usually, this procedure's only task is to save the module's instance handle for further use.

TosoModuleInit
: This procedure is usually called immediately after the module has been loaded and supplies the module with some information about the serving application (like its interface version, its serial number etc.). In addition, it supplies the address of a MODULE_ID structure that the module should fill with information about itself, like module type (external command, import filter or export filter), module name and available commands.
Finally, this procedure is the right place to permanently allocate memory, to load settings, to initialize global data etc.

TosoModuleCommand
: This procedure is called each time the user selects a command that is to be handled by this module.

TosoModuleExit
: This procedure is called at the time the module is to be discarded from memory, i.e. at the time the serving application is terminated. This is the right time to free all memory allocated by the module and to save settings (if required).

TosoModuleModify
: This optional procedure is called each time the serving application or another module requires a module-supplied entity to be initialised or modified, e.g. by multiplying with a matrix. This procedure should only be provided if the module is capable of handling at least one module-supplied entity.

In order to enable the serving application to find these procedure, their names must be exactly as stated above. In addition, they must be "exported". When developing modules using a C compiler, this can be done by either declaring the procedures as `DLL_EXPORT` (defined as `__declspec(dllexport)`) or by stating them in the EXPORT section of a .DEF file supplied to the linker.

The following C code shows a possible implementation of the main four callback procedures for a module of type External Command. Please note the comments included!

Contents

C Source Code

```
#include        "windows.h"             // Windows definitions
#include        "windowsx.h"            // Windows definitions
#include        "toso40.h"              // Toso Interface 4.0 definitions


// Static variables used to store global handles.

HINSTANCE    hInstDLL,
             hGlobalInst;
HWND         hGlobalWnd;
```

```
MODULE_COMMAND_DATA  CommandData[TVG_MODULE_MENU_MAX];


BOOL WINAPI DllMain( HINSTANCE hInstance,
                     DWORD Reason,
                     LPVOID Dummy )
{

// This procedure's only duty is to store the module's
// instance handle. All other initialization tasks should
// be performed inside the TosoModuleInit procedure below.

  switch( Reason ) {
    case DLL_PROCESS_ATTACH:
      hInstDLL = hInstance;
      break;

    case DLL_PROCESS_DETACH:
      hInstDLL = NULL;
      break;
  }
  return( TRUE );
}


DLL_EXPORT BOOL TosoModuleInit( const LPSTR SerialNumber,
                                HINSTANCE hMainInst,
                                HWND hMainWnd,
                                int InterfaceVersion,
                                MODULE_ID* ModuleID )
{
  int           Count;

// Check whether the serving application supports at
// least the required interface version. If not, return
// immediately without calling any interface function!

  if( InterfaceVersion < TOSO_INTERFACE_VERSION ) {
    MessageBox( hMainWnd, "Wrong Interface Version!",
                          "Sample Module", MB_OK );
    return( FALSE );
  }

// Save the instance handle and the main window's handle
// of the serving application for further reference.
// If the module wants to open popup-windows for user input
// or data display, these windows should be created with
// hMainWnd as their parent window!

  hGlobalInst = hMainInst;
  hGlobalWnd  = hMainWnd;

// Set the OwnerID and the module identification. For a
// further description, see MODULE_ID.

  ModuleID->OwnerID    = 0xffff;
  ModuleID->ModuleID   = 0;
  ModuleID->ModuleCTRL = MODULECTRL_ALL;

// This module supplies external commands that shall be
// listed in the TRIMMING menu, so set ModuleType to
```

```
// MODULETYPE_TRIM.
  ModuleID->ModuleData.Type = MODULETYPE_TRIM;

// Initialize the module's name to be displayed in the
// serving application.
  ModuleID->ModuleData.InputData.CommandMode = COMMAND_DIRECT;
  ModuleID->ModuleData.MenuData.MenuEntry   = "Sample Module";
  ModuleID->ModuleData.MenuData.Description = "Sample Module >";
  ModuleID->ModuleData.IconMode = -1;

  ModuleID->CommandData = CommandData;

// Fill the CommandData arrays with the command names, menu entries
// and additional information.
  CommandData[0].InputData.CommandMode = COMMAND_DIRECT;
  CommandData[0].MenuData.MenuEntry   = "&1  Command One";
  CommandData[0].MenuData.Description = "Sample Module >Command One";
  CommandData[0].IconMode = -1;
  CommandData[0].Type = IDM_ENTRY;

  CommandData[1].InputData.CommandMode = COMMAND_DIRECT;
  CommandData[1].MenuData.MenuEntry   = "&2  Command Two";
  CommandData[1].MenuData.Description = "Sample Module >Command Two";
  CommandData[1].IconMode = -1;
  CommandData[1].Type = IDM_ENTRY;

  CommandData[2].InputData.CommandMode = COMMAND_DIRECT;
  CommandData[2].MenuData.MenuEntry   = NULL;
  CommandData[2].MenuData.Description = NULL;
  CommandData[2].IconMode = -1;
  CommandData[2].Type = IDM_ENTRY | IDM_SEPARATOR;

  CommandData[3].InputData.CommandMode = COMMAND_DIRECT;
  CommandData[3].MenuData.MenuEntry   = "&+  About";
  CommandData[3].MenuData.Description = "Sample Module >About...";
  CommandData[3].IconMode = -1;
  CommandType[3].Type = IDM_ENTRY;

  CommandData[4].Type = IDM_END;

// Return TRUE to indicate the initialization has
// been finished successfully.
  return( TRUE );
}


DLL_EXPORT BOOL TosoModuleCommand( int CommandID, int ExecMode )
{
  BOOL          Result = FALSE;

// React accordingly to a command selection.
  switch( CommandID ) {

// The user selected the command with the index 1
// (i.e. the command titled "Command One".
    case 1:
```

```
      /* Insert here code for executing "Command One" */

      Result = TRUE;
      break;

// The user selected the command with the index 2
// (i.e. the command titled "Command One".
    case 2:

      /* Insert here code for executing "Command Two" */

      Result = TRUE;
      break;

// The user selected the command with the index 4
// (i.e. the command titled "About...".
    case 4:
      MessageBox( hGlobalWnd, "Version 1.0\n\nAuthor: Bugs Bunny",
                              "Sample Module", MB_OK );
      Result = TRUE;
      break;

// CommandID is invalid, so ignore it and return FALSE.
    default:
      Result = FALSE;
      break;
  }
  return( Result );
}


DLL_EXPORT BOOL TosoModuleExit( void )
{

// Usually, this procedure's duty is to free memory and
// to save settings. For this simple module, none of these
// tasks is required, so do only return TRUE.

  return( TRUE );
}
```

Modules of type Import Filter and Export Filter use similar callback procedures with two minor differences. First, inside the TosoModuleInit procedure, the value *ModuleID->ModuleData.Type* is set to MODULETYPE_IMPORT or MODULETYPE_EXPORT.

Second, import and export filters may offer only a single command, i.e. they do not own a submenu and thus set *ModuleID->CommandName* to NULL. As a result, the TosoModuleCommand procedure will always be called with *CommandID* set to zero.

Contents

Related Topics:

Contents Getting Your Private Owner ID

External Commands (How to Create a Module)

One of the main purposes of external modules is to add new, previously unavailable commands to the serving application. Such commands are called "External Commands". There are two types of external commands:

Direct Commands

The first, easy type of command is the "direct" command that does not require any point entry by the user. Such a command can immediately be executed after the user selected that command from the module's menu. As a result, implementing such a command is very simple.

The most common "direct" command is the "About..." command that should be available in every module's (sub)menu. Once the user selected this command, the module's command processing procedure TosoModuleCommand is called with the corresponding command ID of the "About..." menu entry (let's call it IDM_ABOUT) passed in *CommandID*. Most modules will react to that command by simply displaying a message box stating the module's version and some copyright messages:

C Source Code

```
DLL_EXPORT BOOL TosoModuleCommand( int CommandID, int ExecMode )
{
  BOOL Result = FALSE;

  switch( CommandID ) {
    case IDM_ABOUT:
      MessageBox( hGlobalWnd, "Version 1.0\n\nAuthor: Stefan Malz",
                              "Sample Module", MB_OK );
      Result = TRUE;
      break;
  }
  return( Result );
}
```

Of course, a "direct" command can do a lot more than simply displaying a simple message box. It can be used to edit parameters, to load or save files, to execute complete programs, etc. One thing is common for all "direct" commands: They have to be completely terminated at the time the module exits the TosoModuleCommand procedure!

Indirect Commands

The second type of command is the "indirect" command. The implementation of an "indirect" command is more complex than a "direct" command. Nevertheless, you will find that it is rather simple once you've done it for the first time.

"Indirect" commands rely on point entry performed by the user. Any kind of point entry is very complex (think of all possibilities the user has to enter points within the serving application, like snapping, grids, orthogonal mode, direct coordinate entry, multiple windows with different views, etc.!), so this task is best observed by the serving application which has the knowledge to handle all these possibilities.

As a result, an "indirect" command requires some communication between the serving application (observing and handling the basic point entry tasks) and the module (reacting on the point entry).

In order to explain this communication, lets try to implement a very simple command: a line entry. This is

an "indirect" command expecting the user to enter two points (a start-point and an end-point). During the entry of the second point, a "rubber line" should be drawn to indicate how the resulting line will look like. After the point entry is done, the module has to create a line based on the entered points which is then inserted to the current drawing.

First, let's build a simple textual flowchart of the command "Draw Line" (having the command identifier IDM_DRAW_LINE):

Step 1: The user selects the external command "Draw Line"...

TosoModuleCommand(IDM_DRAW_LINE, MODULEEXEC_USER)

First, the module is informed that the user has selected the command "Draw Line" from the module's menu. If the module returns TRUE, the serving application will start the command "Draw Line". This results in a sequence of calls to module-owned callback procedures as described below. For each step the procedures are called in the order of their appearance.
If the module returns FALSE from this procedure, the serving application will not start the command. This allows the module to check whether it makes sense to run the command right now. If, e.g., the module displays an initial parameter editing dialog, and the user presses the "Cancel" button, the module should return FALSE.

Step 2: The entry of the start-point begins...

TosoInputPointInitProc(IDM_DRAW_LINE, 0, XPos, YPos)

The server tells the module that the entry of the first point (*PointIndex* = 0) starts. The current cursor position is passed in *XPos* and *YPos*.
Usually, this is the time to prompt for any choices that might be made for the entering of the first point. In our example, no action is required as the first point does not require any choices to be made.
If the module supplies its own guide window texts, this is the place to call TosoDialogUpdateGuide with appropriate parameters for the start-point.

TosoInputPointMoveProc(IDM_DRAW_LINE, 0, XPos, YPos)

The server tells the module that the current position of the first point (*PointIndex* = 0) is *XPos*, *YPos*.
Usually, this is the time to calculate (but not display!) all data that is required to display the current input status ("rubber lines"). In our example, no action is required as the first point does not require any input status to be drawn.

TosoInputDisplayProc(IDM_DRAW_LINE, 0, hDC, TRUE, TRUE)

The server tells the module that the input status of the first point (*PointIndex* = 0) is to be drawn in XOR-mode.
Usually, this is the time to redraw the complete input status, as both *DrawFixed* and *DrawVariable* are set to TRUE. In our example, no action is required as the first point does not require any input status to be drawn.

Step 3: Each time the user moves the cursor during the entry of the start-point...

TosoInputDisplayProc(IDM_DRAW_LINE, 0, hDC, FALSE, TRUE)

The server tells the module that the input status of the first point (*PointIndex* = 0) is to be drawn in XOR-mode.
Usually, this is the time to redraw the variable (i.e. mouse-move-dependent) part of the input status, as only *DrawVariable* is set to TRUE. In our example, no action is required as the first point does not require any input status to be drawn.

TosoInputPointMoveProc(IDM_DRAW_LINE, 0, XPos, YPos)

The server tells the module that the current position of the first point (*PointIndex* = 0) is *XPos*, *YPos*.

Usually, this is the time to calculate (but not display!) all data that is required to display the current input status ("rubber lines"). In our example, no action is required as the first point does not require any input status to be drawn.

TosoInputDisplayProc(IDM_DRAW_LINE, 0, hDC, FALSE, TRUE)
The server tells the module that the input status of the first point (*PointIndex* = 0) is to be drawn in XOR-mode.
Usually, this is the time to redraw the variable (i.e. mouse-move-dependent) part of the input status, as only *DrawVariable* is set to TRUE. In our example, no action is required as the first point does not require any input status to be drawn.

Step 4: The user finally enters the start-point...

TosoInputDisplayProc(IDM_DRAW_LINE, 0, hDC, TRUE, TRUE)
The server tells the module that the input status of the first point (*PointIndex* = 0) is to be drawn in XOR-mode.
Usually, this is the time to redraw the complete input status, as both *DrawFixed* and *DrawVariable* are set to TRUE. In our example, no action is required as the first point does not require any input status to be drawn.

TosoInputPointMoveProc(IDM_DRAW_LINE, 0, XPos, YPos)
The server tells the module that the current position of the first point (*PointIndex* = 0) is *XPos*, *YPos*.
Usually, this is the time to calculate (but not display!) all data that is required to display the current input status ("rubber lines"). In our example, no action is required as the first point does not require any input status to be drawn.

TosoInputPointExitProc(IDM_DRAW_LINE, 0, XPos, YPos)
The server tells the module that the entry of the first point (*PointIndex* = 0) has ended. The final position of the first point is passed in *XPos* and *YPos*.
Usually, this is the time to save the point coordinates and to check whether the point entered makes sense (especially if the point is used for object identification). In our example, the point coordinates are saved without any further check as the first point may be placed anywhere.

Step 5: The entry of the end-point begins...

TosoInputPointInitProc(IDM_DRAW_LINE, 1, XPos, YPos)
The server tells the module that the entry of the second point (*PointIndex* = 1) starts. The current cursor position is passed in *XPos* and *YPos*.
Usually, this is the time to prompt for any choices that might be made for the entering of the second point. In our example, no action is required as the second point does not require any choices to be made.
If the module supplies its own guide window texts, this is the place to call TosoDialogUpdateGuide with appropriate parameters for the start-point.

TosoInputPointMoveProcProc(IDM_DRAW_LINE, 1, XPos, YPos)
The server tells the module that the current position of the second point (*PointIndex* = 1) is *XPos*, *YPos*.
Usually, this is the time to calculate (but not display!) all data that is required to display the current input status ("rubber lines"). In our example, only the current position has to be saved as no further calculation is required.

TosoInputDisplayProc(IDM_DRAW_LINE, 1, hDC, TRUE, TRUE)
The server tells the module that the input status of the second point (*PointIndex* = 1) is to be drawn in XOR-mode.
Usually, this is the time to redraw the complete input status, as both *DrawFixed* and *DrawVariable* are

set to TRUE. In our example, the module has to draw a line connecting the previously entered start-point with the current cursor position.

Step 6: Each time the user moves the cursor during the entry of the end-point...

TosoInputDisplayProc(IDM_DRAW_LINE, 1, hDC, FALSE, TRUE)

The server tells the module that the input status of the second point (*PointIndex* = 1) is to be drawn in XOR-mode.
Usually, this is the time to redraw the variable (i.e. mouse-move-dependent) part of the input status, as only *DrawVariable* is set to TRUE. In our example, the module has to draw a line connecting the previously entered start-point with the current cursor position.

TosoInputPointMoveProc(IDM_DRAW_LINE, 1, XPos, YPos)

The server tells the module that the current position of the second point (*PointIndex* = 1) is *XPos*, *YPos*.
Usually, this is the time to calculate (but not display!) all data that is required to display the current input status ("rubber lines"). In our example, only the current position has to be saved as no further calculation is required.

TosoInputDisplayProc(IDM_DRAW_LINE, 1, hDC, FALSE, TRUE)

The server tells the module that the input status of the second point (*PointIndex* = 1) is to be drawn in XOR-mode.
Usually, this is the time to redraw the variable (i.e. mouse-move-dependent) part of the input status, as only *DrawVariable* is set to TRUE. In our example, the module has to draw a line connecting the previously entered start-point with the current cursor position.

Step 7: The user finally enters the end-point...

TosoInputDisplayProc(IDM_DRAW_LINE, 1, hDC, TRUE, TRUE)

The server tells the module that the input status of the second point (*PointIndex* = 1) is to be drawn in XOR-mode.
Usually, this is the time to redraw the complete input status, as both *DrawFixed* and *DrawVariable* are set to TRUE. In our example, the module has to draw a line connecting the previously entered start-point with the current cursor position.

TosoInputPointMoveProc(IDM_DRAW_LINE, 1, XPos, YPos)

The server tells the module that the current position of the second point (*PointIndex* = 1) is *XPos*, *YPos*.
Usually, this is the time to calculate (but not display!) all data that is required to display the current input status ("rubber lines"). In our example, only the current position has to be saved as no further calculation is required.

TosoInputPointExitProc(IDM_DRAW_LINE, 1, XPos, YPos)

The server tells the module that the entry of the second point (*PointIndex* = 1) has ended. The final position of the second point is passed in *XPos* and *YPos*.
Usually, this is the time to save the point coordinates and to check whether the point entered makes sense (especially if the point is used for object identification). In our example, the point coordinates are saved without any further check as the second point may be placed anywhere.

Step 8: The entry was completed, so the command has to be finished...

TosoInputFinishProc(IDM_DRAW_LINE, 2)

The server tells the module that the command was completed, the user entered two points (*PointCount* = 2).
Usually, this is the time to create or modify objects depending on the user entry. In our example, a line with the entered start-point and end-point is to be created, inserted to the current drawing and displayed.

For some commands, there is a need of one or two other callback procedures to allow the user to edit parameters of the command during its execution and to determine how many points to enter:

Step 9: During point entry, the user selects the "Edit Parameters" command...

TosoInputParameterProc(IDM_DRAW_LINE)

The server tells the module that the user wants to edit the parameters of the current command. Usually, this is the time to display a dialog window that allows viewing and editing of the current command's parameters. If this procedure returns TRUE, the process will return to step 1, allowing the user to start again with the new parameters. In our example, no parameters are available, so there is no action to be performed here.

If an external command offers parameters, these parameters will often also be offered once for editing at the time the command is selected from the menu (i.e. inside the TosoModuleCommand procedure *prior* to returning TRUE).

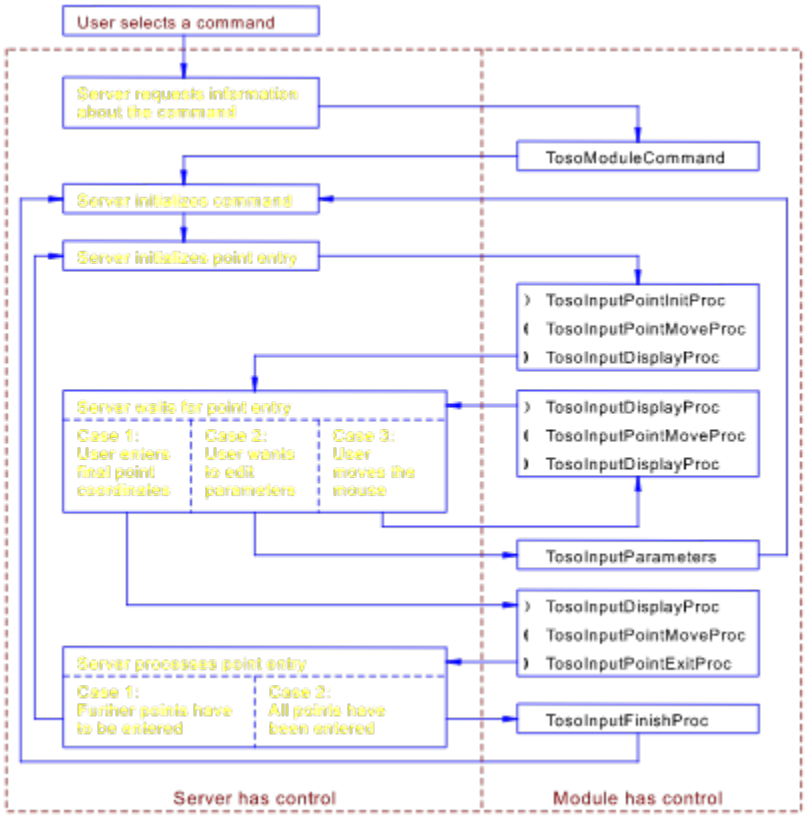
Step 10: During point entry, the user presses the right mouse button to cancel the command

TosoInputCancelProc(IDM_DRAW_LINE)

The server tells the module that the user has pressed the right mouse button. This procedure is only called if the current command requires a variable number of points to be entered (like the command **Polyline**), and if at least one point has already been entered.
Usually, this is the time to display a dialog window that asks whether to continue entering more points or to terminate the command (either with or without creating the currently entered object). In our example, the number of points is fixed, so there is no action to be performed here.

If an external command requires a variable number of points, it will often not display any dialog window at the time the user presses the right mouse button, but immediately create the object based on all previously entered points. If the user does not want that object, he can simply undo the last operation. For the implementation of a complex command requiring a variable number of points, have a look at the commands **Draw>Curve** and **Draw>Surface**.

Now, let's build a graphical flowchart based on the previous textual one. This flowchart is more common, i.e. it will not only be valid for the command "Draw Line":



Notes

The steps described above are *not* a complete list of all possible steps, only the most common ones. There are some situations where the callback procedures may be called in unexpected orders:

Contents If the user presses the ESC key to re-enter the previous point, the module will not be notified explicitly! Instead, the callback procedures will simply be called with a decremented value in the *PointIndex* parameter. In any case, the TosoInputPointInitProc will first be called with the decremented *PointIndex* parameter to allow any pre-calculation required.

So assure that this sudden fallback will not cause data loss! Many problems occur by multiple use of a single global variable. Let's assume that the value of a global variable is calculated at *PointIndex* = 0, used at *PointIndex* = 1 and re-calculated at *PointIndex* == 2. If now a fallback from *PointIndex* = 2 to *PointIndex* = 1 occurs, the content of that global variable will be invalid (because it now contains the value calculated at *PointIndex* = 2)!

Contents If the user has created a macro (using Windows' macro recorder or any similar tool) to enter a sequence of points via direct coordinate input, those points might be passed to the module without intermediate calls to the TosoInputDisplayProc callback procedure.

So assure that you do not rely on TosoInputDisplayProc being called for each point! Place any vital calculation either inside the TosoInputPointInitProc, inside the TosoInputPointMoveProc or inside the TosoInputPointExitProc.

Contents If the user or the system forces a window redraw, the input status will *not* previously be deleted! Instead, the window will be deleted, redrawn and then TosoInputDisplayProc will

be called *once*!

So assure that you do not rely on TosoInputDisplayProc being called an even number of times! As the input status should always be drawn in XOR-mode, this should usually cause no problem.

Regardless of the exceptions just stated, some assumptions may be made to allow efficient coding. They will always be guaranteed by the serving application, independent of the module's behaviour:

Prior to the first call of TosoInputDisplayProc with any given *PointIndex*, both the TosoInputPointInitProc and the TosoInputPointMoveProc have been called with the same *PointIndex*, i.e. you should calculate any data required for the display either in the TosoInputPointInitProc (if it does *not* depend on mouse movement) or in the TosoInputPointMoveProc (if it *does* depend on mouse movement).

Prior to the call to the TosoInputPointExitProc with any given *PointIndex*, the TosoInputPointMoveProc has been called with the same *PointIndex* and *exactly* the same coordinates passed in *XPos* and *YPos*, i.e. you do not have to re-calculate data that has already been calculated inside the TosoInputPointMoveProc.

If you follow all these rules, you will find that in many cases, both the TosoInputPointInitProc and the TosoInputPointExitProc will have no duty, i.e. they will not have to be implemented at all!

Following a possible implementation of the eight callback procedures for the command "Draw Line". Normally, "empty" callback procedures that do only return the "default" value do not have to be supplied (instead, the corresponding address in the *ModuleProc* entry of the MODULE_COMMAND_DATA structure is set to NULL). But for this example, all eight callback procedures have been implemented to provide a complete overview!

C Source Code

```
// Static command description data block

static COMMAND_DATA gInfo = {
  COMMAND_FIXED,
  2,
  { POINT_START, POINT_END },
  { -1,          0         },
  { NULL,        NULL      }
};

// Static coordinate array used to
// store the point coordinates

static DPOINT gPoint[2];

// Implementation of TosoInputPointInitProc:
// Server tells module that entry of point X starts

int TosoInputPointInitProc( int CommandID, int PointIndex,
                            double XPos, double YPos )
{
  return( INPUT_OK );
}

// Implementation of TosoInputPointMoveProc:
```

```
// Server informs module of new coordinates
BOOL TosoInputPointMoveProc( int CommandID, int PointIndex,
                             double XPos, double YPos )
{
  gPoint[PointIndex].x = XPos;
  gPoint[PointIndex].y = YPos;

  return( TRUE );
}

// Implementation of TosoInputPointExitProc:
// Server tells module that entry of point X has finished
int TosoInputPointExitProc( int CommandID, int PointIndex,
                            double XPos, double YPos )
{
  return( INPUT_OK );
}

// Implementation of TosoInputDisplayProc:
// Server tells module to draw/erase all
// required "rubber lines" for point X
void TosoInputDisplayProc( int CommandID, int PointIndex,
                           HDC hDrawDC, BOOL DrawFixed, BOOL DrawVariable )
{
  if( DrawVariable ) {
    if( PointIndex == 1 )
      TosoInputDrawLine( hDrawDC, gPoint[0].x, gPoint[0].y,
                                  gPoint[1].x, gPoint[1].y );
  }
}

// Implementation of TosoInputParameterProc:
// Server tells module to display parameter window
// (no action required for "Draw Line")

BOOL TosoInputParameterProc( int CommandID )
{
  return( FALSE );
}

// Implementation of TosoInputCancelProc:
// Server tells module that user pressed right mouse button
// (no action required for "Draw Line")

int TosoInputCancelProc( int CommandID )
{
  return( INPUT_FINISH );
}

// Implementation of TosoInputFinishProc:
// Server tells module to create the line

void TosoInputFinishProc( int CommandID, int PointCount )
{
  BOOL Result = TRUE;

  if( !TosoCreationStart() )
    return;
```

```
  TosoUndoInitProcess();

  TosoObjectOpen( OBJ_LINE );
  TosoObjectAddPoint( DB_POINT_START, gPoint[0].x, gPoint[0].y );
  TosoObjectAddPoint( DB_POINT_END,   gPoint[1].x, gPoint[1].y );

  if( !TosoObjectFastInsert() )
    Result = FALSE;

  if( Result ) {
    TosoUndoFinishProcess();
    TosoDrawNewObjects();
  }
  else
    TosoUndoCancelProcess();

  TosoCreationEnd();
}

// Implementation of TosoModuleCommand:
// Module either executes a command directly or
// supplies the server with information about
// an external command to start
DLL_EXPORT BOOL TosoModuleCommand( int CommandID, int ExecMode )
{
  BOOL           Result = FALSE;
  MODULE_COMMANDIDX_MODULE_COMMAND Data;

  switch( CommandID ) {
    case IDM_DRAW_LINE:
      Result = TRUE;
      break;
  }
  return( Result );
}
```

The names of the callback procedures may be freely chosen, as the address of those procedures is passed to the server in the MODULE_ID structure of the TosoModuleInit call. Anyway, it might be a good idea to always use the names stated above.

For further experience, you can find a more sophisticated example of external command implementation in the sources of `TSAMPLE_.DLL` . This implementation includes additional code for language-dependent data libraries and the implementation of a dialog window for parameter editing.

Contents

Related Topics:

Contents Getting Your Private Owner ID

Contents Basic Module Code

Export Filters (How to Create a Module)

The implementation of an export filter for the *Toso Interface* is very simple. The interface offers some enumeration procedures that can resolve data into several logic levels, allowing the developer to choose the level that fits best to the addressed export format. If required, the interface resolves *all* data, including instances, blocks and texts, into a sequence of simple polylines and, if possible, polygons. However, if the export format is "intelligent" enough, the data can be retrieved in form of complex curve data consisting of lines, Bézier curves and circular arcs, plus texts.

If the user executes an export command controlled by an external export filter, he is automatically asked to select the entities he wants to export. As a result, the export filter itself does not have to control this selection, it can simply rely on it (e.g., the procedure TosoEnumerateAll, which is usually used to export data to a file, does automatically enumerate those user-selected objects).

Simple Export Filter

For example, lets start with a very simple export filter that is only used to export a sequence of coordinates describing the nodes of polylines. Each line starts with an identifier indicating whether this is the last line of the file or not. If not, a pair of coordinates follows, separated by commas and terminated by a semicolon. Referring to the standard point types in TVG 4.0 files, the identifier is DB_POINT_ANY if another coordinate follows and DB_END if the file ends. A possible file created by this export filter could be:

```
0,10.0,20.0;
0,20.0,20.0;
0,20.0,10.0;
0,10.0,10.0;
999;
```

This file describes four points at the coordinates (10.0,20.0), 20.0,20.0), (20.0,10.0) and (10.0,10.0).

First, let's have a look at the basic command procedure of that export filter module. The command procedure is responsible for retrieving the name of the file to which the data shall be exported. It then creates this file, call the export function and then handles possible errors. This procedure is basically the same for all export filters.

C Source Code

```
// Static variable is used to report errors.
// This is more effective than returning an
// error code from every procedure.

static int gError;

// Export Filter entry point.

DLL_EXPORT BOOL TosoModuleCommand( int CommandID, int ExecMode )
{
  FILENAME        FileName;
  HANDLE          FileHandle;
  BOOL            Result;

  if( CommandID != 0 )
    return( FALSE );
```

```
// Retrieve file name of export file
// (usually by means of common dialog boxes).
  if( !ModuleGetFileName( FileName ) )
    return( FALSE );

  Result = FALSE;
  SetCursor( LoadCursor( NULL, IDC_WAIT ) );

// Create export file.
  if( TosoFileCreate( &FileHandle, FileName ) ) {

// Export data (implementation see below).
    ModuleExport( FileHandle, FileName );

// Close export file.
    TosoFileClose( FileHandle );

// Error handling.
    switch( gError ) {
      case 999:
        TosoFileDelete( gFileName );
        MessageBox( hGlobalWnd, "Export canceled.", "Export Filter", MB_OK );
        Result = TRUE;
        break;

      case 0:
        Result = TRUE;
        break;

      default:
        TosoFileDelete( gFileName );
        MessageBox( hGlobalWnd, "Export error.", "Export Filter", MB_OK );
        Result = FALSE;
        break;
    }
  }
  return( Result );
}
```

Now, let's have a look at the export procedure that is called from within the command procedure. This export procedure simply tells the serving application to pass all object data in a specified format to the module's enumeration callback procedure.

C Source Code

```
void ModuleExport( HANDLE FileHandle, const LPSTR FileName )
{
  FILENAME      FileName2;
  DUMMYSTR      DummyStr;

// Until now, no error occured, so reset gError.
  gError = 0;

// Prepare writing to the file, i.e.
```

```
// initialize the file buffer.
  if( !TosoFileWriteInitDisk( FileHandle ) ) {
    gError = 1;
    return;
  }

// Display a progress indication window. For this
// reason, assemble a string containing the file's
// name and a description of the task performed.
  TosoFileSplitName( FileName, NULL, FileName2 );
  wsprintf( DummyStr, "Exporting to file\n%s", FileName2 );
  TosoDialogShowProgress( "Export Filter", DummyStr, FALSE );

// Start the enumeration of all chosen entitites.
// The enumeration mode ENUMMODE_PLAIN causes all nested
// instances to be resolved, the mode ENUMMODE_POLYLINES
// causes simple polylines to be generated from all
// entity types.
  TosoEnumerateAll( TosoDrawingGetActive(), FLAG_USE, FLAG_USE,
                    ENUMMODE_PLAIN | ENUMMODE_LINES,
                    (TOSOENUMOBJECT_PROC) ModuleEnumCallback );

// Write the terminating line to the file.
  TosoFileWriteShort( DB_END );
  TosoFileWriteSemi();

// Finish writing to the file, i.e.
// write the file buffer to the disk.
  if( !TosoFileWriteFlush() )
    gError = 1;

  TosoFileWriteExit();

// Hide the progress indication window.
  TosoDialogHideProgress();
}
```

Finally, let's have a look at the enumeration callback procedure `ModuleEnumCallback` whose address has been passed to the TosoEnumerateAll procedure. This callback procedure is responsible for translating the enumeration data into data of the desired export file format. As the enumeration can be controlled by stating different enumeration modes, the translation will usually be a simple one-to-one translation.

(When working with complex export file formats, its a good idea to start with a very simple version of this translation by stating ENUMMODE_PLAIN combined with ENUMMODE_LINES. Once this simple translation works, refine the translation by allowing more complex data types to be passed to the callback procedure.)

C Source Code

```
BOOL ModuleEnumCallback( const ENUMDEF_DATA* EnumData )
{
  DUMMYSTR      Text1, Text2;
```

```
  int           Count;

// Check what type of data was passed. As
// ENUMMODE_POLYLINES was used, only objects of
// type CURVE, AREA or MARK will be passed. Bézier
// curves, circles, characters and other complex
// data types have been resolved to polylines.

  switch( EnumData->EnumData ) {
    case ENUMDATA_CURVE:
    case ENUMDATA_AREA:
    case ENUMDATA_MARK:

// Runs through all points of the object...

      for( Count = 0; Count < EnumData->EnumCount; Count++ ) {

// ...and write their coordinates to the file.

        TosoFileWriteShort      ( DB_POINT_ANY );
        TosoFileWriteCommaDouble( EnumData->PointPtr[Count].x );
        TosoFileWriteCommaDouble( EnumData->PointPtr[Count].y );
        TosoFileWriteSemi();

// Check whether a write error occured.

        if( TosoFileWriteError() ) {
          gError = 1;
          break;
        }

// Update the progress indication window.

        wsprintf( Text1, "Line %ld",
                  TosoFileWriteCurrentLine() );
        wsprintf( Text2, "%ld KBytes",
                  ( TosoFileWriteCurrentSize() + 1023 ) / 1024 );
        TosoDialogUpdateProgress( Text1, Text2, NOPARAM, NOPARAM );

// Check whether the user pressed the CANCEL
// button in the progress indication window.

        if( TosoDialogIsCanceled() ) {
          gError = 999;
          break;
        }
      }
      break;
  }

// If any error occured, cancel the enumeration
// by returning FALSE.

  return( !gError );
}
```

For further experience, you can find a more sophisticated version of this export filter in the sources of `EXPORT_.DLL` . This implementation includes additional code for language-dependent data libraries and the implementation of a customized common dialog window for file name retrievement.

Related Topics:


Getting Your Private Owner ID


Basic Module Code


External Commands


Import Filters


Handling of Coordinates


Creating Text Objects


Creating Dimension Objects


Creating Groups


Debugging Modules


Module Style Guide

Import Filters (How to Create a Module)

The implementation of an import filter for the *Toso Interface* is a bit more complex than creating export filters. Basically, the import file format determines the complexity of the import filter. The first duty of an import filter developer is to take some time to understand the import file format and to think about how to translate it to TVG 4.0 entities.

Simple Import Filter

For example, lets start with a very simple import filter that is capable of reading a list of coordinate pairs and converts them to markings. The import file for this filter shall be based on the data produced by the simple export filter build in the Export Filters chapter. Each line starts with an identifier indicating whether this is the last line of the file or not. If not, a pair of coordinates follows, separated by commas and terminated by a semicolon. Referring to the standard point types in TVG 4.0 files, the identifier is DB_POINT_ANY if another coordinate follows and DB_END if the file ends. A possible file to be imported by this import filter could be:

```
0,10.0,20.0;
0,20.0,20.0;
0,20.0,10.0;
0,10.0,10.0;
999;
```

This file describes four points at the coordinates (10.0,20.0), (20.0,20.0), (20.0,10.0) and (10.0,10.0).

First, let's have a look at the basic command procedure of that import filter module. The command procedure is responsible for retrieving the name of the file to be imported. It then opens this file, processes it and subsequently creates some entities using interface procedures. Finally, it handles possible errors. This procedure is basically the same for all import filters.

C Source Code

```
// Static variable is used to report errors.
// This is more effective than returning an
// error code from every procedure.
static int gError;

// Import Filter entry point.
DLL_EXPORT BOOL TosoModuleCommand( int CommandID, int ExecMode )
{
  FILENAME        FileName;
  HANDLE          FileHandle;
  BOOL            Result;

  if( CommandID != 0 )
    return( FALSE );

// Prepare the interface for object creation.
  if( !TosoCreationStart() )
    return( FALSE );

// Retrieve file name of import file
// (usually by means of common dialog boxes).
```

```
  if( !ModuleGetFileName( FileName ) )
    return( FALSE );

  Result = FALSE;
  SetCursor( LoadCursor( NULL, IDC_WAIT ) );

// Open import file.

  if( TosoFileOpen( &FileHandle, FileName ) ) {
    TosoUndoInitProcess();

// Import data (implementation see below).

    ModuleImport( FileHandle, FileName );

// Close import file.

    TosoFileClose( FileHandle );

// Error handling.

    switch( gError ) {
      case 999:

// Terminate the current undo level without
// maintaining the objects created since the
// previous call to TosoUndoInitProcess.

        TosoUndoCancelProcess();
        MessageBox( hGlobalWnd, "Import canceled.", "Import Filter", MB_OK );
        Result = TRUE;
        break;

      case 998:

// Terminate the current undo level without
// maintaining the objects created since the
// previous call to TosoUndoInitProcess.

        TosoUndoCancelProcess();
        MessageBox( hGlobalWnd, "Out Of Memory.", "Import Filter", MB_OK );
        Result = FALSE;
        break;

      case 0:

// Terminate the current undo level while
// maintaining all objects created since the
// previous call to TosoUndoInitProcess.

        TosoUndoFinishProcess();

// If blocks or instances had been modified, a call
// of the TosoUndoUpdateLinks() procedure would
// be necessary here!
// Redraw all windows to show the newly created objects.

        TosoDrawWindowAll();
        Result = TRUE;
        break;

      default:

// Terminate the current undo level without
```

```
// maintaining the objects created since the
// previous call to TosoUndoInitProcess.

        TosoUndoCancelProcess();
        wsprintf( DummyStr, "Error %d in line %ld (offset %ld bytes)",
                  gError,
                  TosoFileReadCurrentLine(),
                  TosoFileReadCurrentSize() );
        MessageBox( hGlobalWnd, DummyStr, "Import Filter", MB_OK );
        Result = FALSE;
        break;
    }
  }

// Terminate the object creation process.

  TosoCreationEnd();
  return( Result );
}
```

Now, let's have a look at the import procedure that is called from within the command procedure. This import procedure reads the data from the import file and translates it into TVG 4.0 entities.

C Source Code

```
void ModuleImport( HANDLE FileHandle, const LPSTR FileName )
{
  FILENAME      FileName2;
  DUMMYSTR      DummyStr;
  int           Count;
  double        x, y;

// Until now, no error occured, so reset gError.

  gError = 0;

// Prepare reading from the file, i.e.
// initialize the file buffer.

  if( !TosoFileReadInitDisk( FileHandle ) )
    return;

// Display a progress indication window. For this
// reason, assemble a string containing the file's
// name and a description of the task performed.

  TosoFileSplitName( FileName, NULL, FileName2 );
  wsprintf( DummyStr, "Importing from file\n%s", FileName2 );
  TosoDialogShowProgress( "Import Filter", DummyStr, TRUE );

// Try to read a coordinate pair from the file.

  if( ModuleReadCoordinate( &x, &y ) ) {
    Count = 0;

    do {

// Create an object of type OBJ_MARK (marking).

      if( Count == 0 )
        TosoObjectOpen( OBJ_MARK );
```

```
// Add one marking to the previously opened object.

      TosoObjectAddPoint( DB_POINT_MARK, x, y );
      Count++;

// Check whether the maximum number of points per object
// is reached. If so, insert the current object and
// reset the counter.

      if( Count >= POINTS_PER_OBJECT ) {
        if( !TosoObjectFastInsert() ) {
          gError = 998;
          goto _stop;
        }
        Count = 0;
      }

// Try to read further coordinate pairs.

    } while( ModuleReadCoordinate( &x, &y ) );

// If an object is still open, insert it.

    if( Count > 0 )
      if( !TosoObjectFastInsert() )
        gError = 998;

// Read the final semicolon.

    TosoFileReadSemi();
  }

_stop:

// Finish reading from the file.

  TosoFileReadExit();

// Hide the progress indication window.

  TosoDialogHideProgress();
}
```

Finally, let's implement the basic reading procedure. It first reads the identification at the beginning of each line and, if applicable, the following coordinate pair:

C Source Code

```
BOOL ModuleReadCoordinate( double* x, double* y )
{
  DUMMYSTR      Text1, Text2;
  short         Dummy;

// If any error occured, stop here.

  if( gError )
    return( FALSE );

// Update the progress indication window.

  wsprintf( Text1, "Line %ld",
```

```
            TosoFileReadCurrentLine() );
  wsprintf( Text2, "%ld KBytes",
            ( TosoFileReadCurrentSize() + 1023 ) / 1024 );
  TosoDialogUpdateProgress( Text1, Text2,
                            TosoFileReadCurrentSize(),
                            TosoFileReadTotalSize() );

// Check whether the user pressed the CANCEL
// button in the progress indication window.

  if( TosoDialogIsCanceled() ) {
    gError = 999;
    return( FALSE );
  }

// Read the first short value from the file. If its
// value is DB_END, the file ends here.

  TosoFileReadShort( &Dummy );
  if( Dummy == DB_END )
    return( FALSE );

// Read a pair of coordinates separated by commas
// and terminated by a semicolon.

  TosoFileReadCommaDouble( x );
  TosoFileReadCommaDouble( y );
  TosoFileReadSemi();

// Check whether a read error occured.

  if( TosoFileReadError() ) {
    gError = 2;
    return( FALSE );
  }

  return( TRUE );
}
```

For further experience, you can find a more sophisticated version of this import filter in the sources of `IMPORT_.DLL` . This implementation includes additional code for language-dependent data libraries and the implementation of a customized common dialog window for file name retrievement.

Related Topics:

Getting Your Private Owner ID

Basic Module Code

External Commands

Export Filters

Handling of Coordinates

Creating Text Objects

Creating Dimension Objects

Creating Groups

Debugging Modules

Module Style Guide

Handling of Coordinates (How to Create a Module)

During the development of modules, you will often have to handle with coordinates of different types. Basically, all objects created and maintained by the serving application will use the so-called "internal coordinates". These coordinates are always stated in millimeters relative to the current page's center.

They are independent all any user settings, i.e. they are not influenced by scales, units or the current origin position. Internal coordinates determine the size and position of the objects when they are printed. A square with a side length of 10 mm in internal coordinates will be printed in the same size (unless the user explicitly scales the output).

Due to their invariable definition, internal coordinates are easy to handle and to calculate with. But they are not suitable for interaction with the user, as the user is used to work with origins, scales and different units. So modules will usually have to work both in internal coordinates and in "user coordinates", depending on the user's settings of scale, unit and origin.

The circumstance that the scale, the unit and the origin are always store as a compound group within coordinate systems simplifies the work with them. The serving application helps you by calculating two matrices for each coordinate system that allow a fast and easy conversion from internal coordinates to user coordinates and vice versa.

Let's assume your module allows the user to enter a rectangle by stating the coordinates of its corner-points. This entry will be done in user coordinates, so you will have to calculate the resulting internal coordinates in order to create the rectangle. After you have read the four values (x1, y1, x2, y2), you can convert them to internal coordinates using the following code. Please note that the resulting rectangle may be a rotated or distorted rectangle, so all four corner-points have to calculated separately!

C Source Code

```
// Variables required for the calculation.

SYSTEMDEF  CurrentSystem;
DPOINT     p1, p2, p3, p4;

// Initialize the variables.

TosoSystemGetCurrentDef( &CurrentSystem );

// Convert the coordinates.

p1.x = mat_x( &CurrentSystem.SystemUnitToMM, x1, y1 );
p1.y = mat_y( &CurrentSystem.SystemUnitToMM, x1, y1 );

p2.x = mat_x( &CurrentSystem.SystemUnitToMM, x1, y2 );
p2.y = mat_y( &CurrentSystem.SystemUnitToMM, x1, y2 );

p3.x = mat_x( &CurrentSystem.SystemUnitToMM, x2, y2 );
p3.y = mat_y( &CurrentSystem.SystemUnitToMM, x2, y2 );

p4.x = mat_x( &CurrentSystem.SystemUnitToMM, x2, y1 );
p4.y = mat_y( &CurrentSystem.SystemUnitToMM, x2, y1 );
```

The calculated points can now be used to create a polygon:

C Source Code

```
// Variables required to create a polygon.
BOOL          Result;

// Create a polygon.
TosoObjectOpen( OBJ_SURFACE );
TosoObjectAddPoint( DB_POINT_START, p1.x, p1.y );
TosoObjectAddPoint( DB_POINT_END,   p2.x, p2.y );
TosoObjectAddPoint( DB_POINT_END,   p3.x, p3.y );
TosoObjectAddPoint( DB_POINT_END,   p4.x, p4.y );

// Insert the polygon into the current drawing.
Result = TosoObjectFastInsert();
```

For further information about the internal data block structure of the object OBJ_SURFACE, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

A similar method can be used for a conversion in the opposite direction. If, e.g., a module offers an information command that allows the user to identify an object whose coordinates will then be displayed, it will have to convert internal coordinates into user coordinates. A point with the coordinates (x1,y1) can be converted using the following code:

C Source Code

```
// Variables required for the calculation.
SYSTEMDEF  CurrentSystem;
DPOINT     p1;

// Initialize the variables.
TosoSystemGetCurrentDef( &CurrentSystem );

// Convert the coordinates.
p1.x = mat_x( &CurrentSystem.SystemMMToUnit, x1, y1 );
p1.y = mat_y( &CurrentSystem.SystemMMToUnit, x1, y1 );
```

Please note that the only difference is in the matrix used (*SystemMMtoUnit* instead of *SystemUnitToMM*). When converting vectors rather than points, you can simply use another macro for this calculation, the rest remains the same. A vector with the values (dx,dy) can be converted using the following code:

C Source Code

```
// Variables required for the calculation.
SYSTEMDEF  CurrentSystem;
DPOINT     v1;

// Initialize the variables.
TosoSystemGetCurrentDef( &CurrentSystem );

// Convert the vector.
v1.x = mat_dx( &CurrentSystem.SystemMMToUnit, dx, dy );
v1.y = mat_dy( &CurrentSystem.SystemMMToUnit, dx, dy );
```

Here, the only difference the usage of the macros `mat_dx` and `mat_dy` instead of `mat_x` and `mat_y`.

Using the methods described above, you should be able to perform any required coordinate transformation. For more information on how to determine transformation matrixes for frequently used modifications, see the corresponding chapter in TVG 4.0 Documentation (TVG40.HLP).

Related Topics:


Getting Your Private Owner ID


Basic Module Code


External Commands


Export Filters


Import Filters


Creating Text Objects


Creating Dimension Objects


Creating Groups


Debugging Modules


Module Style Guide

Creating Text Objects (How to Create a Module)

When implementing import filters or external commands, you will sometimes have to create complex objects that required more than simple geometrical data. The most common type of complex objects are texts. This chapter will show some examples on how to create these types of objects.

Text objects consist of three types of data: The text itself, a description of the font to be used, and the position of the text. First, let's create a standard text using the default parameters of the serving application:

C Source Code

```
// Variables required to create a text object.

BOOL          Result;
TEXTSTANDARD  Data;

// Initialize the variables.

TosoInitTextStandard( &Data, NULL );

// Create a standard text object.

TosoObjectOpen( OBJ_TEXTSTANDARD );
TosoObjectAddTextLong( "This is an example!", FALSE );
TosoObjectAddTextStandard( &Data );

// Insert the text object into the current drawing.

Result = TosoObjectFastInsert();
```

This short code sequence will result in the text "This is an example" to be displayed in the page's center, using the default font (Arial) and size (5.0 mm).

For further information about the internal data block structure of the object OBJ_TEXTSTANDARD, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

Now, let's modify this code to allow the user to edit the text and select the font and size he wants to use (modified or added code sections will be displayed in lighter blue):

C Source Code

```
// Variables required to create a text object.

BOOL          Result;
TEXTSTANDARD  Data;
STR8192       Text;

// Initialize the variables.

TosoInitTextStandard( &Data, NULL );
Text[0] = '\0';

// Display a dialog window to edit text and font.

TosoDialogTextStandard( hGlobalWnd, "Edit Text", &Data, NULL, Text );

// Create a standard text object.
```

```
TosoObjectOpen( OBJ_TEXTSTANDARD );
TosoObjectAddTextLong( Text, FALSE );
TosoObjectAddTextStandard( &Data );

// Insert the text object into the current drawing.

Result = TosoObjectFastInsert();
```

Still, the resulting text will be placed at the page's center. Usually, the module wants to place the text at a known location in the drawing - either due to calculation or due to user entry. The text's position is determined by a matrix inside the text parameters. This matrix can be edited directly:

C Source Code

```
// Variables required to create a text object.

BOOL          Result;
TEXTSTANDARD  Data;
STR8192       Text;

// Initialize the variables.

TosoInitTextStandard( &Data, NULL );
Text[0] = '\0';

// Display a dialog window to edit text and font.

TosoDialogTextStandard( hGlobalWnd, "Edit Text", &Data, NULL, Text );

// Create a standard text object.

Data.TextMatrix.m31 = -100.0;
Data.TextMatrix.m32 =   50.0;
TosoObjectOpen( OBJ_TEXTSTANDARD );
TosoObjectAddTextLong( Text, FALSE );
TosoObjectAddTextStandard( &Data );

// Insert the text object into the current drawing.

Result = TosoObjectFastInsert();
```

This modified code will result in a text placed at the coordinates (-100.0, 50.0), measured in millimeters relative to the page's center. Depending on the parameters the user chose during the TosoDialogTextStandard command, the text will be rotated, distorted and/or stretched.

For creating a reference text, i.e. a text with a reference line and a frame surrounding the text, only a few more steps are necessary:

C Source Code

```
// Variables required to create a text object.

BOOL          Result;
TEXTSTANDARD  Data;
TEXTREFERENCE Data2;
STR8192       Text;

// Initialize the variables.
```

```
TosoInitTextStandard( &Data, &Data2 );
Text[0] = '\0';

// Display a dialog window to edit text and font.

TosoDialogTextStandard( hGlobalWnd, "Edit Text", &Data, &Data2, Text );

// Create a reference text object.

Data.TextMatrix.m31 = -100.0;
Data.TextMatrix.m32 =   50.0;
TosoObjectOpen( OBJ_TEXTREFERENCE );
TosoObjectAddPoint( DB_POINT_ANY, -50.0, 0.0 );
TosoObjectAddTextLong( Text, FALSE );
TosoObjectAddTextStandard( &Data );
TosoObjectAddTextReference( &Data2 );

// Insert the text object into the current drawing.

Result = TosoObjectFastInsert();
```

This modified code will result in a reference text placed at the coordinates (-100.0, 50.0), measured in millimeters relative to the page's center. Depending on the parameters the user chose during the TosoDialogTextStandard command, the text will be rotated, distorted and/or stretched. The reference point is located at the coordinates (-50.0, 0.0), so a reference line will be drawn between the text's coordinates (-100.0, 50.0) and the reference point (-50.0, 0.0).

Finally, let's create a frame text. This is a text whose position is not determined by an insertion point, but by a surrounding parallelogram. This allows two feature that a standard text does not have. First, the text can be justified to the parallelogram's borders, and second, the text can flow oblique, i.e. the lines of text do not start at the same position:

For further information about the internal data block structure of the object OBJ_TEXTFRAME, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

The following code creates a frame text that has a similar form to the example shown above. Usually, the module will let the user determine the shape of the surrounding frame, or calculate it based on other user input.

C Source Code

```
// Variables required to create a text object.

BOOL          Result;
TEXTFRAME     Data;
STR8192       Text;

// Initialize the variables.

TosoInitTextFrame( &Data );
Text[0] = '\0';

// Display a dialog window to edit text and font.
```

```
TosoDialogTextFrame( hGlobalWnd, "Edit Text", &Data, Text );

// Create a frame text object.

TosoObjectOpen( OBJ_TEXTFRAME );
TosoObjectAddPoint( DB_POINT_ANY,  25.0, 100.0 );
TosoObjectAddPoint( DB_POINT_ANY, 125.0, 100.0 );
TosoObjectAddPoint( DB_POINT_ANY,   0.0,   0.0 );
TosoObjectAddTextLong( Text, FALSE );
TosoObjectAddTextFrame( &Data );

// Insert the text object into the current drawing.

Result = TosoObjectFastInsert();
```

For further information about the internal data block structure of the object OBJ_TEXTFRAME, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

Using the three types of text objects described above within your modules will result in great drawings, allowing the user to use high-quality fonts and numerous text effects! So do not hesitate to use them...

Related Topics:

Getting Your Private Owner ID

Basic Module Code

External Commands

Export Filters

Import Filters

Handling of Coordinates

Creating Dimension Objects

Creating Groups

Debugging Modules

Module Style Guide

Creating Dimension Objects (How to Create a Module)

When implementing import filters or external commands, you will sometimes have to create complex objects that required more than simple geometrical data. A rather common type of complex objects are dimensions. This chapter will show some examples on how to create these types of objects.

Dimension objects consist of several types of data: Geometrical data defining the measure, geometrical data defining the dimension line's position, the dimension texts, a description of the font to be used, and the position of the text. Each type of dimension has its special sequence of data blocks required to provide all this data.

First, lets create a simple length dimension using standard parameters.

C Source Code

```
// Variables required to create a dimension object.

BOOL          Result;
DIMLARGE      Data;

// Initialize the variables.

TosoInitDimLarge( &Data, TRUE );

// Create a dimension text object.

TosoObjectOpen( OBJ_DDISTANCE );
TosoObjectAddPoint( DB_POINT_START, -100.0, 0.0 );
TosoObjectAddPoint( DB_POINT_END,   100.0, 0.0 );
TosoObjectAddPoint( DB_POINT_ANY,   0.0, 0.0 );
TosoObjectAddPoint( DB_POINT_ANY,   0.0, 10.0 );
TosoObjectAddPoint( DB_POINT_ANY,   0.0, 0.0 );
TosoObjectAddPoint( DB_POINT_ANY,   0.0, 0.0 );
TosoObjectAddPoint( DB_POINT_ANY,   0.0, 0.0 );
TosoObjectAddTextShort( "", FALSE );
TosoObjectAddTextShort( "", TRUE );
TosoObjectAddTextShort( "", FALSE );
TosoObjectAddTextShort( "", FALSE );
TosoObjectAddTextShort( "", FALSE );
TosoObjectAddDimLarge( &Data, TRUE );

// Insert the text object into the current drawing.

Result = TosoObjectFastInsert();
```

This short code sequence will result in a distance dimension that measures the distance between the point (-100.0, 0.0) and (100.0, 0.0). The dimension line's position is determined by the point (0.0, 10.0).

For further information about the internal data block structure of the object OBJ_DDISTANCE, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

Related Topics:

Getting Your Private Owner ID

Creating Groups (How to Create a Module)

When creating multiple objects during the execution of one command, it will often be useful to store these objects inside a group.

```
BOOL  Result = TRUE;

TosoUndoInitProcess();

// Open a group. While a group is open, all objects and
// instances created will be added to that group.

TosoGroupOpen();

// Create the first object to be added to the group.

if( Result ) {
  TosoObjectOpen( OBJ_LINE );
  TosoObjectAddPoint( DB_POINT_START, -148.5, -105.0 );
  TosoObjectAddPoint( DB_POINT_END,    148.5,  105.0 );

  if( !TosoObjectFastInsert() )
    Result = FALSE;
}

// Create the second object to be added to the group.

if( Result ) {
  TosoObjectOpen( OBJ_LINE );
  TosoObjectAddPoint( DB_POINT_START,  148.5, -105.0 );
  TosoObjectAddPoint( DB_POINT_END,   -148.5,  105.0 );

  if( !TosoObjectFastInsert() )
    Result = FALSE;
}

// Close the group, insert it into the drawing and
// create an instance that references that group.

if( Result ) {
  if( !TosoGroupFastInsert( TRUE, NULL ) )
    Result = FALSE;
}

// Update the internal links (necessary if groups or blocks
// have been created) and draw the newly created objects.

if( Result ) {
  TosoUndoFinishProcess();
  TosoUndoUpdateLinks();
  TosoDrawNewObjects();
}
else
  TosoUndoCancelProcess();
```

Contents

Related Topics:

Debugging Modules (How to Create a Module)

During the development of an external module, you will sometime have to locate errors. Inside the module itself, this can usually be simplified by using the compiler's debug option. But what about errors that occur inside the serving application, possibly caused by illegal parameters passed through the interface?

The serving application contains a simple debug layer that can be activated by setting a special value in the application's registration database entry. In order to activate the internal debugging, start the "Registration Editor" and search for the application's keys in the HKEY_LOCAL_MACHINE subtree of the registry.

There you should find a subtree containing global settings of the serving application. If this application is CAD Release 4, you will find the subtree "SOFTWARE\TommySoftware®\CAD Release 4\English", containing several values. One of these values should be "Debugging", usually set to zero.

The internal debugging layer supports several levels of debugging which can be activated by a bitwise OR combination of the following constants:

0x00000001	Standard debugging. Enables additional message display in all standard error cases. This option will *not* slow down any operation.
0x00000002	Interface debugging. Enables additional message display when the interface detects error situations or illegal parameters. This option will *not* slow down any operation.
0x00000004	File debugging. Enables strict file format checking and message display in error cases. This option *will* slow down file operations!
0x00000008	Term debugging. Enables additional status display during term evaluation. This option will result in several message displayed for any term entry being scanned (e.g. in edit fields).

During normal module development, the "Debugging" value should be set to 0x00000003, enabling extended message display in all standard and interface error cases. All other options should only be activated if really required.

Related Topics:

Getting Your Private Owner ID

Basic Module Code

External Commands

Export Filters

Import Filters

Handling of Coordinates

Creating Text Objects

Creating Dimension Objects

Creating Groups

Module Style Guide

Module Style Guide (How to Create a Module)

Modules based on the *Toso Interface* will always work inside user interface that the serving application offers. As a result, the user assumes that modules will look and work similar to the serving application. This will help him to easily get into each new module, however powerful it is.

In order to provide a consistent user interface, external module should follow some design rules that are assembled in this Module Style Guide. Even though each module has its freedom to react in any manner, a *good* should try to fit perfectly into the complete system. This starts with the naming of the module and its commands, continues at the design of command icons and ends at the design of dialog windows.

Module Naming

Module files have to have special file names: the module's file name has to end with an underscore `_` (Ansi 95), followed by the extension `.DLL`.

Uually, import and export filter file names should start with either `IMP` (for import filters) or `EXP` (for export filters), followed by a description of the file type they can handle (e.g. `IMPDXF_.DLL` and `EXPDXF_.DLL` for DXF filters). In order to avoid duplicate names, they may also start with the module creator's three-character identification.

Command extension module file names should *always* start with the three-character identification of the module's creator. You will receive your private three-character identification at the time you contact TommySoftware® in order to receive your unique owner identifier (see Getting Your Private Owner ID).

When using language-dependent sublibraries, like all modules supplied by TommySoftware® do, name the language-dependent sublibrary similar to the module's file, but without the final exclamation mark. The same applies to help files (which are also language-dependent). If your module is named `FSBMOD1_.DLL`, the language-dependent library is to be named `FSBMOD1.DLL`, the help file `FSBMOD1.HLP`.

Resulting from the fact that you have to use a three-character identification at the beginning and an exclamation mark at the end of the file name, you have four characters left to identify the module. A good idea would be to use simple shortcuts like `DIM` (for dimensioning), `TXT` (for text), `TRM` (for trimming), or `DRW` (for drawing) to identify the type of commands the module offers, plus one digit to allow several modules for a similar purpose. Anyway, you may freely choose the four remaining characters. Just make sure that you do not use any file name twice!

Version Numbers

When displaying a module's version number (usually after the user selected the module's "About..." command), this number should include a language identification. Modules supplied by TommySoftware® use a version number with two fractional digits followed directly (without a separating space) by the language identifier. The following language identifications are pre-defined:

- `c` Chinese
- `d` German
- `e` English
- `f` French
- `i` Italian
- `j` Japanese
- `r` Russian
- `s` Spanish

A typical version number is `1.00e`, indicating the first English release. Version `1.01e` would be a minor (free) update to version `1.00e` (usually due to bug-fixing), whereas version `1.10e` would be a major (charged) update, featuring additional or more powerful commands.

Be careful with increasing your version numbers very often. Versions changing to rapidly will irritate the user! The release number (the digit before the point) should not be increased more than once in one to two years, the first fractional digit at most after two months.

Command Structure

An external module supplies a number of commands that can by called by the serving application. After being loaded into memory, the module will be asked what commands it does offer. It may return a list of up to 20 commands that will be placed in a submenu of the module popup-menu. If no submenu is required (as only one command is available), the module does supply an empty list.

The submenu may contain separators to increase the legibility of the submenu. Use them! A good value is to have a separator every four commands in average. The last command of the submenu should always be "About...", displaying a message box stating the module's name, its version and copyright messages.

In order to allow a good naming, each command may has two different names: a "short" name displayed in the menu (up to about 20 characters long) and a "long" description that is displayed in the status line while the command is executed (up to about 60 characters long). Use the opportunity to make the long command name more precise!

If a command results in a dialog window to be opened immediately after the command's selection, append ellipses points to the command's name (please do always use three single points instead of the Ansi character 133, which will cause problems on some displays).

Command Icons

All modules and filters can supply a command icon to their commands at module initialisation time. This icon will be used in the same way the serving application's icons are used.

A command icon is a monochrome bitmap with a size of 40 by 40 pixels. If a module creates its own menu, the icon for the menu itself is only 24 by 24 pixels in size. The content of a command's or menu's icon is not restricted, but try to create icons that fit into the serving application's icons! If a module supplies more than one icon, they should be placed within a single, large bitmap to reduce the module's resource size.

Message Boxes

If a module wants to display short messages, it should use the MessageBox command of WIN32. As the serving application uses the automatically subclassing CTL3D extension to draw three-dimensional dialog windows and controls on older Windows NT systems (which do not offer three-dimensional effects on their own), those message boxes will automatically use the same three-dimensional style as the serving application's boxes.

There is no need for the module to initialize the CTL3D library on its own, nor does it have to include the CTL3D_3DCHECK bitmap into its resources. Just state the serving application's main window handle (passed to the TosoModuleInit procedure) as the parent window for all message boxes, dialog windows and popup windows.

When running on Windows 95 or Windows NT 4.0, which will both offer their own three-dimensional effects, the CTL3D extension will not be activated.

Dialog Boxes

In order to give the user the impression that the serving application and the modules are forming an integrated whole it is important that the dialog boxes in the modules provide the same "look and feel" as those in the serving application. For example, all dialog boxes shoudl use the font "MS Sans Serif 8pt".

Any dialog box should provide its own help topic using hotspots. If the user presses the F1 key inside a module's dialog box, this help topic should be displayed. In order to get notice of the user pressing the F1 key, the following code segment should be placed in each dialog window's main message dispatching code:

```
BOOL CALLBACK DialogProc( HWND hDlg, UINT Message,
                          WPARAM wParam, LPARAM lParam )
{
  switch( Message ) {

      ...

    case WM_ENTERIDLE:
      return( TosoDialogEnterIdle( hDlg, wParam, lParam ) );

    default:
      if( Message == TosoDialogHelpMessage() ) {
        WinHelp( ... );
        return( TRUE );
      }

      ...

      break;
  }
  return( FALSE );
}
```

Only if this code is placed is *all* dialog windows (whether they have their own help topic or not), the help will work in all dialog windows, including those of the serving application called through the programming interface.

Help Files

When creating help files (e.g. using Word for Windows 2.0 or higher), you should use the following standard text layouts to provide a consistant look over all help files:

Topic titles (non-scrolling area)
: 22pt, 18pt or 16pt Times New Roman, bold, black, paragraph option "Keep With Next" active

Subtitles inside a topic
: 16pt, 14pt or 12pt Times New Roman, bold, black, 6pt space behind the paragraph

Standard text
: 11 pt Times New Roman, regular, black, indents on multiples of 0.25 inch (in text window) or multiples of 0.125 inch (in index window)

Names of controls (buttons radios, checkboxes etc.)
: 11 pt Times New Roman, regular, black, text in double upper quotes
(e.g.: Press the button "Cancel" to exit the dialog without saving the changes.)

Single characters

11 pt Times New Roman, regular, character in single upper quotes
(e.g.: The text will be broken after the characters ' ' and '-'.)

User input, texts in edit controls or list boxes

10.5 pt Courier, regular, dark blue, including the preceeding and trailing space!
(e.g.: Enter `1/100` to set the scale to 1:100.)

File names

10.5 pt Courier, regular, dark blue, upper case, including the preceeding and trailing space!
(e.g.: The command will create a file titled `RELEASE4.INF` in the application's directory.)

Strengthening of single words or phrases

11 pt Times New Roman, italic, black
(e.g.: This parameter *must not* be NULL!)

Extremely important notes

11 pt Times New Roman, bold italic, red
(e.g.: ***Never use this command before you called the procedure XYZ!***)

Variable names

11 pt Times New Roman, regular, dark magenta
(e.g.: The variable *LineOrientation* determines the orientation of the dimension line.)

Data type names

11 pt Times New Roman, regular, dark red
(e.g.: This macro converts a *double* value into a rounded *int* value.)

Constant names

11 pt Times New Roman, regular, yellow
(e.g.: Returns INPUT_CANCEL if the input was canceled by the user.)

C source code, only when used in complete lines

9.5 pt Courier, regular, dark blue

```
(e.g.: BOOL TosoModuleExit( void ); )
```

Please note that if you are working in Word for Windows, you have to explicitly set the standard text color to "Black"! Using the text color "Auto" will cause problems if the user has non-standard color settings in his system.

If you include graphics into your help files, try to use vector metafiles instead of bitmaps wherever possible. They usually require less memory and can be scaled without quality loss. If your graphics include fonts, be sure to only use fonts that the user will also have, i.e. Times New Roman, Arial and Wingdings! A good font to use inside graphics is Arial Regular 9 pt. This font is small, but good to read.

If you have to use bitmap (e.g. for dialog window display or screen element explanation), be sure to use transparent bitmaps were required. Help Compiler 4.0 allows the use of transparent bitmaps by using the `bmct`, `bmlt` and `bmrt` commands for bitmap inclusion.

All help files supplied by TommySoftware® will either use a single help window (if the help system is rather small like for most modules) or two help windows arranged side-to-side. All help windows use a light gray for background color of both text and non-scrolling area. The window sizes should be defined as follows (in logical units):

Primary window: x = 0, y = 0, width = 720, height = 1024
Secondary window: x = 720, y = 0, width = 304, height = 1024

If your help files uses two windows, include an "Exit" button in the secondary window that calls the "Exit()" help macro to allow an easy closing of both windows at once.

Related Topics:

Getting Your Private Owner ID

Basic Module Code

External Commands

Export Filters

Import Filters

Handling of Coordinates

Creating Text Objects

Creating Dimension Objects

Creating Groups

Debugging Modules

DllMain (Callback Procedures)

Contents **Syntax**

```
BOOL WINAPI DllMain( HINSTANCE hInstance,
                     DWORD Reason,
                     LPVOID Dummy );
```

This procedure is called once when the module is loaded from disk and once when it is discarded again. It is the standard entry point for any DLL.

Contents **Parameters**

hInstance

[*HINSTANCE*] Instance handle of the DLL. As this handle might be needed later, it should be stored in a global variable.

Reason

[*DWORD*] Supplies a flag indicating why the DLL entry routine is being called. This value can be one of the following values:

DLL_PROCESS_ATTACH

Indicates the DLL is attaching to the address space of the current process as a result of the process starting up or a call to LoadLibrary. DLLs can use this opportunity to initialize any instance data or to use the TlsAlloc function to allocate a thread local storage (TLS) index.
During initial process startup or after a call to LoadLibrary, the operating system scans the list of loaded DLLs for the process. For each DLL that has not already been called with a DLL_PROCESS_ATTACH flag, the system calls the DLL's entrypoint function. This call is made in the context of the thread that caused the process address space to change, such as the primary thread of the process or the thread that called LoadLibrary.

DLL_THREAD_ATTACH

Indicates the current process is creating a new thread. When this occurs, the system calls the entrypoint of all DLLs currently attached to the process. The call is made in the context of the new thread. DLLs can use this opportunity to initialize a TLS slot for the thread. The thread calling the DLL entrypoint with the DLL_PROCESS_ATTACH flag does not call the DLL entrypoint with the DLL_THREAD_ATTACH flag.
Note that a DLL's entrypoint is called with this flag value only by threads created after the DLL is attached to the process. When a DLL is attached by LoadLibrary, existing threads do not call the entrypoint of the newly loaded DLL.

DLL_THREAD_DETACH

Indicates a thread is exiting cleanly. If the DLL has stored a pointer to allocated memory in a TLS slot, it uses this opportunity to free the memory. The operating system calls the entrypoint of all currently loaded DLLs with this flag value. The call is made in the context of the exiting thread.
There are cases in which the entrypoint is invoked for a terminating thread even if the DLL never attached to the thread. For example, the entrypoint was never called with the DLL_THREAD_ATTACH flag in the context of the thread. This can occur in two situations:
- The thread was the initial thread in the process so the system called the entrypoint with the DLL_PROCESS_ATTACH flag.
- The thread was already running when a LoadLibrary call was made, so the system never called the entrypoint function for it.

DLL_PROCESS_DETACH

Indicates the DLL is detaching from the address space of the calling process. This results from either a clean process exit, or from a FreeLibrary call. The DLL can use this opportunity to call the TlsFree function to free any TLS indices allocated by using TlsAlloc and to free any thread local data. When a DLL detaches from a process as a result of process termination, or FreeLibrary, the operating system does not call the DLL's entrypoint with the DLL_THREAD_DETACH flag for the individual threads of the process. The only notification given to the DLL is the DLL_PROCESS_DETACH notification. DLLs can take this opportunity to clean up all resources for all threads attached and known to the DLL.

Dummy

[*LPVOID*] Specifies further aspects of DLL initialization and cleanup. If *Reason* is DLL_PROCESS_ATTACH, *Dummy* is NULL for dynamic loads, and non-NULL for static loads. If *Reason* is DLL_PROCESS_DETACH, *Dummy* is NULL if DllMain has been called via FreeLibrary, and non-NULL if DllMain has been called during process termination.

Contents

Return Value

When the system calls the DllMain function with the DLL_PROCESS_ATTACH flag, the function returns TRUE if it succeeds or FALSE if the initialization fails. If the return value is FALSE when the DllMain was invoked because the process called the LoadLibrary function, LoadLibrary returns NULL. If the return value is FALSE when the DllMain is called during process initialization, the process terminates with an error. To get extended error information, use the GetLastError function.

When the system calls the DllMain function with any flag other than DLL_PROCESS_ATTACH, the return value is ignored.

Contents

Comment

Although the Win32 documentation (as stated above) says this procedure is the right place to perform initialization tasks, modules using the *Toso Interface* should not perform any task except storing the module's instance handle. Instead, all initialization should be performed inside the TosoModuleInit procedure. This will ensure that all data will be initialized for each instance of the application running.

For an example of DllMain, see the Basic Module Code.

Contents

Related Topics:

Contents TosoModuleInit

Contents TosoModuleExit

Contents TosoModuleCommand

Contents

TosoModuleInit (Callback Procedures)

Contents

Syntax

```
BOOL TosoModuleInit( const LPSTR SerialNumber,
                     HINSTANCE hMainInst,
                     HWND hMainWnd,
                     int InterfaceVersion,
                     MODULE_ID* ModuleID )

typedef BOOL (*TOSOMODULEINIT_PROC)( const LPSTR SerialNumber,
                                     HINSTANCE hMainInst,
                                     HWND hMainWnd,
                                     int InterfaceVersion,
                                     MODULE_ID* ModuleID );
```

This procedure is called when the module is loaded into the application's memory, i.e. when the application starts. The procedure is called each time an instance of the application is started. But as every single instance of the DLL lies in another memory segment, each instance can act like being the only instance in memory.

Contents

Parameters

SerialNumber

[*const LPSTR*] This string contains the serial number of the serving application. It can be used to determine whether this module is licensed to run on that application. If the serving application is a demo version, the string *SerialNumber* will only contain the word "DEMO" in capitals. In this case, export filters should either not work at all or at least add some additional data to the exported drawing (like the word "DEMO" across the drawing).

hInstance

[*HINSTANCE*] Instance handle of the serving application. This handle can be used to retrieve data from the serving application, like bitmaps, cursors etc.

hMainWnd

[*HWND*] Window handle of the serving application's main window. This handle should be used as the parent window's handle when displaying dialog windows.

InterfaceVersion

[*int*] This value indicates the current interface version of the serving application. Compare this value with the constant TOSO_INTERFACE_VERSION. If *InterfaceVersion* is higher than TOSO_INTERFACE_VERSION, return FALSE.

ModuleID

[*MODULE_ID**] Address of a module identification structure that must be filled in by the module before returning TRUE from TosoModuleInit.

Contents

Return Value

Should return TRUE if the module has been initialised correctly. If this procedure returns FALSE, the module will immediately be removed from memory and will not be available to the user.

Contents

Comment

The time this procedure is called is a good time to perform any initialization like memory allocation, variable initialization etc. Please do *not* display dialogs or windows showing copyright etc. at this time - this will lead to extreme start-up times of the application. Instead, add either an "Infos..." button to the primary module dialog window or add a menu item "About..." to the module's submenu.

The name of the procedure has to be exactly `TosoModuleInit`. For an example of TosoModuleInit, see the Basic Module Code.

Related Topics:

DllMain

TosoModuleExit

TosoModuleCommand

TosoModuleModify

TosoModuleExit (Callback Procedures)

Syntax

```
BOOL TosoModuleExit( void );

typedef BOOL (*TOSOMODULEEXIT_PTR)( void );
```

This procedure is called when the module is discarded from the application's memory, i.e. when the application closes down. At this time, any allocated memory should be freed.

Parameters

None.

Return Value

Should return TRUE if the module may be removed from memory. If this procedure returns FALSE, the application will not terminate! However, other modules that previously returned TRUE are already removed and will not be available to the user any more. FALSE should *only* be returned if removing the module might result in harmful data loss.

Comment

The name of the procedure has to be exactly `TosoModuleExit` . For an example of TosoModuleExit, see the Basic Module Code.

Related Topics:

DllMain

TosoModuleInit

TosoModuleCommand

TosoModuleModify

TosoModuleCommand (Callback Procedures)

Contents

Syntax

```
BOOL TosoModuleCommand( int CommandID,
                        int ExecMode );

typedef BOOL (*TOSOMODULECOMMAND_PROC)( int CommandID,
                                        int ExecMode );
```

This procedure is called each time the user selects a command of this module by either selecting it in the menu, in the popup menu, via the toolbox or by pressing a key assigned to the command.

Contents

Parameters

CommandID

[*int*] This value identifies the module command selected by the user. The value of *CommandID* is either zero (if the module has only one command without a submenu) or a value between one and the number of commands list in the *CommandName* field of the module's MODULE_ID structure.

ExecMode

[*int*] This value indicates what action has to be performed concerning the selected command. Possible values are:

MODULEEXEC_USER

The command identified by *CommandID* has been selected by the user and shall be started now. If the command requires parameter input before being able to start, those parameters have to be edited *now*!

MODULEEXEC_SYSTEM

The command identified by *CommandID* is restarted by the serving application, e.g. after another temporary command has been executed. In this case, parameters should *not* be edited before starting the command!

MODULEEXEC_HELP

The user wants to view the help text for the command identified by *CommandID*.

Contents

Return Value

Should return TRUE if the command has been finished (or at least initialised) successfully, or FALSE if not. If *ExecMode* was MODULEEXEC_HELP, the return value is ignored.

Contents

Comment

The name of the procedure has to be exactly `TosoModuleCommand`. For an example of TosoModuleCommand, see the Basic Module Code.

Contents

Related Topics:

TosoModuleModify (Callback Procedures)

Syntax

```
BOOL TosoModuleModify( UNIT_USER_PTR UserObj,
                       int ModifyMode,
                       const MATRIX* Matrix );

typedef BOOL (*TOSOMODULEMODIFY_PROC)( UNIT_USER_PTR UserObj,
                                       int ModifyMode,
                                       const MATRIX* Matrix );
```

This procedure is called each time a module-supplied entity is to be modified, e.g. by multiplying it with a matrix. If the module knows the entity pointed to by *UserObj*, it should execute that modification.

Parameters

UserObj

[*UNIT_USER_PTR*] Address of the module-supplied entity that is to be modified. The module should check the values in *UserObj->Header.OwnerID* and *UserObj->UserType* on whether it knows this entity or not. If not, the module should immediately exit returning FALSE.

ModifyMode

[*int*] Value defining the action to be performed on the module-supplied entity. Defined values are:

MODIFY_INIT

The object *UserObj* shall be initialised, i.e. a new display block shall be created.

MODIFY_MATRIX

The object *UserObj* shall be multiplied with the matrix whose address is given in *Matrix*. As a result, either the entity's *DisplayMatrix* element must be modified, or the display block referenced by the module-supplied entity must be updated.

Matrix

[*MATRIX*] Address of a modification matrix that is to be applied to the module-supplied entity in some cases. Might be NULL if not needed.

Return Value

Should return TRUE if the module knows the entity and was able to modify it, or FALSE if not.

Comment

The name of the procedure has to be exactly `TosoModuleModify`.

Related Topics:

DllMain

TosoEnumAttribProc (Callback Procedures)

Contents

Syntax

```
BOOL TosoEnumAttribProc( int AttribType,
                         const LPSTR AttribName,
                         const LPSTR AttribValue );

typedef BOOL (*TOSOENUMATTRIB_PROC)( int AttribType,
                                     const LPSTR AttribName,
                                     const LPSTR AttribValue );
```

This callback procedure is called once for each existing attribute after starting an attribute enumeration by means of the TosoEnumerateInstanceAttrib or TosoEnumerateBlockAttrib procedure.

Contents

Parameters

AttribType

[*int*] Type of the enumerated attribute. Possible values are:

DB_ATTRIB_GLOBAL_TEXT
: The attribute is a global attribute, i.e. an attribute that is equal for all instances. It may contain any type of textual data.

DB_ATTRIB_GLOBAL_NUM
: The attribute is a global attribute, i.e. an attribute that is equal for all instances. It may contain only texts that represent a valid floating point number.

DB_ATTRIB_LOCAL_TEXT
: The attribute is a local attribute, i.e. an attribute that may be different for each instance. It may contain any type of textual data.

DB_ATTRIB_LOCAL_NUM
: The attribute is a local attribute, i.e. an attribute that may be different for each instance. It may contain only texts that represent a valid floating point number.

AttribName

[*const LPSTR*] This text contains the name of the attribute.

AttribValue

[*const LPSTR*] This text contains the current value of the attribute.

Contents

Return Value

Should return TRUE to continue the enumeration or FALSE to stop it.

Contents

Comment

The name `TosoEnumAttribProc` is only a placeholder, the procedure may have any name.

Contents

Related Topics:

TosoEnumObjectProc

TosoEnumIdentProc

TosoEnumPointsProc

TosoEnumObjectProc (Callback Procedures)

Syntax

```
BOOL TosoEnumObjectProc( const ENUMDEF_DATA* EnumData );

typedef BOOL (*TOSOENUMOBJECT_PROC)( const ENUMDEF_DATA* EnumData );
```

This callback procedure is called once for each enumeration step after starting an enumeration by means of the TosoEnumerateAll, TosoEnumerateLibrary, TosoEnumerateUnit or TosoEnumerateBlock procedure.

Parameters

EnumData

[*const ENUMDEF_DATA**] This structure contains information about the currently enumerated unit.

Return Value

Should return TRUE to continue the enumeration or FALSE to stop it.

Comment

The name `TosoEnumObjectProc` is only a placeholder, the procedure may have any name.

Contents

Related Topics:

TosoEnumAttribProc

TosoEnumIdentProc

TosoEnumPointsProc

TosoEnumIdentProc (Callback Procedures)

Syntax

```
BOOL TosoEnumIdentProc( UNIT_PTR UnitPtr );

typedef BOOL (*TOSOENUMIDENT_PROC)( UNIT_PTR UnitPtr );
```

This callback procedure is called once for each identified unit after starting an enumeration by means of the TosoEnumerateIdent procedure.

Parameters

UnitPtr

[*UNIT_PTR*] Address of an identified unit. Even though this value is not declared as `const`, the object should not be altered directly! If you want to modify identified objects, be sure to first create a duplicate of those objects.
Anyway, modifying an object directly in memory should only be done by advanced developers with deep knowledge of the internal memory structure of units.

Return Value

Should return TRUE to continue the enumeration or FALSE to stop it.

Comment

The name `TosoEnumIdentProc` is only a placeholder, the procedure may have any name.

Related Topics:

TosoEnumAttribProc

TosoEnumObjectProc

TosoEnumPointsProc

TosoEnumPointsProc (Callback Procedures)

Syntax

```
BOOL TosoEnumPointsProc( UNIT_PTR UnitPtr,
                         BLOCK_PTR BlockPtr );

typedef BOOL (*TOSOENUMPOINTS_PROC)( UNIT_PTR UnitPtr,
                                     BLOCK_PTR BlockPtr );
```

This callback procedure is called once for each identified point after starting an enumeration by means of the TosoEnumeratePoints procedure.

Parameters

UnitPtr

[*UNIT_PTR*] Address of the unit containing the identified point. Even though this value is not declared as `const`, the object should not be altered directly! If you want to modify identified objects, be sure to first create a duplicate of those objects.
Anyway, modifying an object directly in memory should only be done by advanced developers with deep knowledge of the internal memory structure of units.

BlockPtr

[*BLOCK_PTR*] Address of the identified point data block. Even though this value is not declared as `const`, the point should not be altered directly! If you want to modify identified points, be sure to first create a duplicate of the objects containing those points.

Return Value

Should return TRUE to continue the enumeration or FALSE to stop it.

Comment

The name `TosoEnumPointsProc` is only a placeholder, the procedure may have any name.

The enumeration will always enumerate all identified points of each object *directly* after each other, i.e. if you only want to enumerate all *objects* containing identified points, save the parameter *UnitPtr* in a static variable and ignore all directly following calls with an equal value of *UnitPtr*.

Related Topics:

TosoEnumAttribProc

TosoEnumObjectProc

Contents

TosoHookPositionProc (Callback Procedures)

Syntax

```
void TosoHookPositionProc( int WindowNum,
                           DPOINT* Position );

typedef void (*TOSOHOOKPOSITION_PROC)( int WindowNum,
                                       DPOINT* Position );
```

This callback procedure is called once each time the current cursor's position is changed.

Parameters

WindowNum

[*int*] Zero-based index of the drawing window that is responsible for the event. Valid range: 0 <= Value < TVG_WINDOW_MAX (standard windows) or Value = TVG_WINDOW_VIEW (view window).

Position

[*DPOINT**] Address of a point structure containing the cursor's new position in internal millimeters (relative to the page center, scale-independent). The module may alter these values to force a special movement of the cursor. Both the cursor and the coordinate display will react accordingly. Anyway, be sure to use this possibility only if required, as another module might also try to do so!

Return Value

None.

Comment

A hook on cursor position calculation can be established and canceled using the procedures TosoHookPositionStart and TosoHookPositionEnd.

The name `TosoHookPositionProc` is only a placeholder, the procedure may have any name.

Related Topics:

TosoHookMouseProc

TosoHookKeyProc

TosoHookMouseProc (Callback Procedures)

Syntax

```
void TosoHookMouseProc( int WindowNum,
                        UINT Message,
                        UINT Buttons,
                        int XPos,
                        int YPos );

typedef void (*TOSOHOOKMOUSE_PROC)( int WindowNum,
                                    UINT Message,
                                    UINT Buttons,
                                    int XPos,
                                    int YPos );
```

This callback procedure is called once each time a mouse event occurs.

Parameters

WindowNum

[*int*] Zero-based index of the drawing window that is responsible for the event. Valid range: 0 <= Value < TVG_WINDOW_MAX (standard windows) or Value = TVG_WINDOW_VIEW (view window).

Message

[*UINT*] Type of mouse event that occured. Possible values are:

WM_MOUSEMOVE
WM_LBUTTONDOWN
WM_LBUTTONUP
WM_MBUTTONDOWN
WM_MBUTTONUP
WM_RBUTTONDOWN
WM_RBUTTONUP

For a description of these standard mouse events, refer to the windows SDK documentation.

Buttons

[*UINT*] Current button status. Indicates whether various virtual keys are down. This parameter can be any combination of the following values:

MK_CONTROL
Set if the CTRL key is down.

MK_LBUTTON
Set if the left mouse button is down.

MK_MBUTTON
Set if the middle mouse button is down.

MK_RBUTTON
Set if the right mouse button is down.

MK_SHIFT
Set if the SHIFT key is down.

XPos,
YPos

[*int*] Current mouse position in pixels, relative to the upper left corner of the drawing window's client area.

Contents

Return Value

None.

Contents

Comment

A hook on mouse events can be established and canceled using the procedures TosoHookMouseStart and TosoHookMouseEnd.

The name `TosoHookMouseProc` is only a placeholder, the procedure may have any name.

Contents

Related Topics:

Contents TosoHookPositionProc

Contents TosoHookKeyProc

TosoHookKeyProc (Callback Procedures)

Contents Syntax

```
void TosoHookKeyProc( int WindowNum,
                      UINT Message,
                      int Key,
                      long Data );

typedef void (*TOSOHOOKKEY_PROC)( int WindowNum,
                                  UINT Message,
                                  int Key,
                                  long Data );
```

This callback procedure is called once each time a key event occurs.

Contents Parameters

WindowNum

[*int*] Zero-based index of the drawing window that is responsible for the event. Valid range: 0 <= Value < TVG_WINDOW_MAX (standard windows) or Value = TVG_WINDOW_VIEW (view window).

Message

[*UINT*] Type of key event that occured. Possible values are:

WM_KEYDOWN
WM_KEYUP

For a description of these standard key events, refer to the windows SDK documentation.

Key

[*int*] Virtual key code of the key that has been pressed or released.

Data

[*long*] Specifies the repeat count, scan code, extended-key flag, context code, previous key-state flag, and transition-state flag, as shown in the following table:

Bit	*Meaning*
0-15	Specifies the repeat count. The value is the number of times the keystroke is repeated as a result of the user holding down the key.
16-23	Specifies the scan code. The value depends on the original equipment manufacturer (OEM).
24	Specifies whether the key is an extended key, such as a function key or a key on the numeric keypad. The value is 1 if it is an extended key; otherwise, it is 0.
25-28	Reserved.
29	Specifies the context code.
30	Specifies the previous key state.
31	Specifies the transition state.

Contents Return Value

None.

Contents **Comment**

A hook on key events can be established and canceled using the procedures TosoHookKeyStart and TosoHookKeyEnd.

The name `TosoHookKeyProc` is only a placeholder, the procedure may have any name.

Contents

Related Topics:

Contents TosoHookPositionProc

Contents TosoHookMouseProc

TosoInputPointInitProc (Callback Procedures)

Syntax

```
int TosoInputPointInitProc( int CommandID,
                            int PointIndex,
                            double XPos,
                            double YPos );

typedef int (*TOSOINPUTPOINTINIT_PROC)( int CommandID,
                                        int PointIndex,
                                        double XPos,
                                        double YPos );
```

This procedure is called during the execution of an external command, i.e. a command that is handled by an external module.

Parameters

CommandID
[*int*] Identifier of the current command. Valid range: 0 <= Value < TVG_MODULE_MENU_MAX.

PointIndex
[*int*] Zero-based index of the point that is currently entered. Valid range: 0 <= Value < TVG_INPUT_MAX.

XPos
YPos
[*double*] Coordinates of the current mouse position in internal millimeters (relative to the page center, scale-independent).

Return Value

Should return one of the following values:

INPUT_OK
Point initialisation successful, continue point entry.

INPUT_CANCEL
Cancel the complete command (usually due to user termination).

INPUT_NEXT
Continue immediately with the next point.

INPUT_FINISH
Finish the command (even though not all points have been entered!).

Comment

This procedure is called each time the entry of a point is initialised. At this time, the module could initialize data required for further handling of this point input, or it can display a dialog window to allow the user to select parameters or a special way to continue.

If the command requires a variable number of points, this procedure must return INPUT_FINISH if the

command shall be finished now. Returning INPUT_OK will continue with the next point entry.

The name `TosoInputPointInitProc` is only a placeholder, the procedure may have any name.

Related Topics:


TosoInputPointMoveProc


TosoInputPointExitProc


TosoInputDisplayProc


TosoInputParameterProc


TosoInputCancelProc


TosoInputFinishProc

TosoInputPointMoveProc (Callback Procedures)

Syntax

```
BOOL TosoInputPointMoveProc( int CommandID,
                             int PointIndex,
                             double XPos,
                             double YPos );

typedef BOOL (*TOSOINPUTPOINTMOVE_PROC)( int CommandID,
                                         int PointIndex,
                                         double XPos,
                                         double YPos );
```

This procedure is called during the execution of an external command, i.e. a command that is handled by an external module.

Parameters

CommandID

[*int*] Identifier of the current command. Valid range: 0 <= Value < TVG_MODULE_MENU_MAX.

PointIndex

[*int*] Zero-based index of the point that is currently entered. Valid range: 0 <= Value < TVG_INPUT_MAX.

XPos

YPos

[*double*] Coordinates of the current mouse position in internal millimeters (relative to the page center, scale-independent).

Return Value

Should return TRUE if the current status is valid, i.e. the current point might be set to the given coordinate. Else return FALSE.

As long as this function returns FALSE, the user cannot enter the current point by pressing the mouse button or by direct coordinate entry. This is to avoid invalid point coordinates.

Comment

This procedure is called each time the current mouse coordinate is changed by either moving the mouse or by entering a coordinate via direct coordinate entry. At this time, the module should store the coordinates and perform all calculations required to display the current input status.

If, e.g., the current command is "Circumcircle", and the third point on the circle is currently entered, this procedure should calculate the center-point of the circle and its radius in order to have these values available for displaying that circle.

In order to allow a flicker-free display while moving the mouse, only short-time calculations should be done during mouse movement. Time-consuming calculations should be performed after the point entry in

finished, i.e. when TosoInputPointExitProc is called.

The name `TosoInputPointMoveProc` is only a placeholder, the procedure may have any name.

Related Topics:


TosoInputPointInitProc


TosoInputPointExitProc


TosoInputDisplayProc


TosoInputParameterProc


TosoInputCancelProc


TosoInputFinishProc

TosoInputPointExitProc (Callback Procedures)

Syntax

```
int TosoInputPointExitProc( int CommandID,
                            int PointIndex,
                            double XPos,
                            double YPos );

typedef int (*TOSOINPUTPOINTEXIT_PROC)( int CommandID,
                                        int PointIndex,
                                        double XPos,
                                        double YPos );
```

This procedure is called during the execution of an external command, i.e. a command that is handled by an external module.

Parameters

CommandID
: [*int*] Identifier of the current command. Valid range: 0 <= Value < TVG_MODULE_MENU_MAX.

PointIndex
: [*int*] Zero-based index of the point that is currently entered. Valid range: 0 <= Value < TVG_INPUT_MAX.

XPos
YPos
: [*double*] Coordinates of the current mouse position in internal millimeters (relative to the page center, scale-independent).

Return Value

Should return one of the following values:

INPUT_OK
: Point may be accepted, continue entry of next point entry (or finish command, if the current point was the final point).

INPUT_CANCEL
: Cancel the complete command (usually due to user termination).

INPUT_ERROR
: Cancel the complete command (usually due to calculation overflow, invalid data etc.).

INPUT_OVERFLOW
: Too many points entered, ignore this point.

INPUT_FINISH
: Finish the command (even though not all points have been entered!).

Comment

This procedure is called each time the entry of a point is finished by either clicking with the mouse or by

entering a coordinate via direct coordinate entry. At this time, the module can perform calculations that would have too time-consuming to do them during point entry. Also, it should store the given coordinates, as they are the final and definite coordinates of the current point.

If the command requires a variable number of points, this procedure must return INPUT_FINISH if the command shall be finished now. Returning INPUT_OK will continue with the next point entry.

The name `TosoInputPointExitProc` is only a placeholder, the procedure may have any name.

Related Topics:


TosoInputPointInitProc


TosoInputPointMoveProc


TosoInputDisplayProc


TosoInputParameterProc


TosoInputCancelProc


TosoInputFinishProc

TosoInputDisplayProc (Callback Procedures)

Contents Syntax

```
void TosoInputDisplayProc( int CommandID,
                           int PointIndex,
                           HDC hDrawDC,
                           BOOL DrawFixed,
                           BOOL DrawVariable );

typedef void (*TOSOINPUTDISPLAY_PROC)( int CommandID,
                                       int PointIndex,
                                       HDC hDrawDC,
                                       BOOL DrawFixed,
                                       BOOL DrawVariable );
```

This procedure is called during the execution of an external command, i.e. a command that is handled by an external module.

Contents Parameters

CommandID
[*int*] Identifier of the current command. Valid range: 0 <= Value < TVG_MODULE_MENU_MAX.

PointIndex
[*int*] Zero-based index of the point that is currently entered. Valid range: 0 <= Value < TVG_INPUT_MAX.

hDrawDC
[*HDC*] Window device context of the window in which the current input status is to be drawn. Direct all output during this procedure to *hDrawDC* without changing its pen and brush settings!

DrawFixed
[*BOOL*] Indicates whether the "fixed" part of the current input status should be drawn or not. The "fixed" part is that part that is independent from mouse movement.
If, e.g., the current command is "Polyline", moving the mouse will only influence the last line of that polyline. So the "fixed" part of this polyline are all lines but the last one.

DrawVariable
[*BOOL*] Indicates whether the "variable" part of the current input status should be drawn or not. The "variable" part is that part that depends on mouse movement.
If, e.g., the current command is "Polyline", moving the mouse will only influence the last line of that polyline. So the "variable" part of this polyline is only its last line.

Contents Return Value

None.

Contents Comment

This procedure is called each time a redraw of the current input status is required (either due to mouse movement or due to a WM_PAINT message). At this time, the module should draw the current input status depending on *DrawFixed* and *DrawVariable*. As no mouse coordinates are passed to this procedure, it's the module's duty to store the required coordinates statically within the TosoInputPointMoveProc.

This procedure will only be called if the last call to TosoInputPointMoveProc returned TRUE, i.e. if the current input status is valid.

The name `TosoInputDisplayProc` is only a placeholder, the procedure may have any name.

Contents

Related Topics:

Contents TosoInputPointInitProc

Contents TosoInputPointMoveProc

Contents TosoInputPointExitProc

Contents TosoInputParameterProc

Contents TosoInputCancelProc

Contents TosoInputFinishProc

TosoInputParameterProc (Callback Procedures)

Contents Syntax

```
BOOL TosoInputParameterProc( int CommandID );

typedef void (*TOSOINPUTPARAMETER_PROC)( int CommandID );
```

This procedure is called during the execution of an external command, i.e. a command that is handled by an external module.

Contents Parameters

CommandID

[*int*] Identifier of the current command. Valid range: 0 <= Value < TVG_MODULE_MENU_MAX.

Contents Return Value

Should return TRUE to inducate that changes in the parameters force a re-start of the command. This re-start will then automatically be done by the serving application. If FALSE is returned, the serving application continues like this procedure had never been called.

Contents Comment

This procedure is usually called each time the application's command "Change Parameters" is called.

In most cases, it might be useful to also call this procedure from within the module before starting a command's execution. The command itself should then only be started if the user does not cancel the dialog.

The name `TosoInputParameterProc` is only a placeholder, the procedure may have any name.

Contents

Related Topics:

Contents TosoInputPointInitProc

Contents TosoInputPointMoveProc

Contents TosoInputPointExitProc

Contents TosoInputDisplayProc

TosoInputCancelProc

TosoInputFinishProc

TosoInputCancelProc (Callback Procedures)

Contents Syntax

```
int TosoInputCancelProc( int CommandID,
                         int PointCount );

typedef int (*TOSOINPUTCANCEL_PROC)( int CommandID,
                                     int PointCount );
```

This procedure is called during the execution of an external command, i.e. a command that is handled by an external module.

Contents Parameters

CommandID
[*int*] Identifier of the current command. Valid range: 0 <= Value < TVG_MODULE_MENU_MAX.

PointCount
[*int*] Number of points that have already been entered. Valid range: 0 <= Value < TVG_INPUT_MAX.

Contents Return Value

Should return one of the following values:

INPUT_OK
Continue the entry of the current point after calling TosoInputPointInitProc again to update calculations (usually is the user altered parameters for the current point).

INPUT_IGNORE
Ignore the pressing of the right mouse button, i.e. continue the entry of the current point.

INPUT_CANCEL
Cancel the complete command (usually due to user termination).

INPUT_FINISH
Finish the command, i.e. call TosoInputFinishProc.

Contents Comment

This procedure is called if the user presses the right mouse button to cancel the command during point entry. The procedure will only be called if the command requires a variable number of point entries (i.e. if the value *InputData.CommandMode* value of the MODULE_COMMAND_DATA structure was set to COMMAND_VARIABLE) and if at least one point has already been entered. At this time, the module should e.g. ask whether to terminate the point entry (either canceling or finishing the command) or to edit parameters of the current point.

The name `TosoInputCancelProc` is only a placeholder, the procedure may have any name.

Related Topics:

 TosoInputPointInitProc

 TosoInputPointMoveProc

 TosoInputPointExitProc

 TosoInputDisplayProc

 TosoInputParameterProc

 TosoInputFinishProc

TosoInputFinishProc (Callback Procedures)

Syntax

```
void TosoInputFinishProc( int CommandID,
                          int PointCount );

typedef void (*TOSOINPUTFINISH_PROC)( int CommandID,
                                      int PointCount );
```

This procedure is called during the execution of an external command, i.e. a command that is handled by an external module.

Parameters

CommandID

[*int*] Identifier of the current command. Valid range: 0 <= Value < TVG_MODULE_MENU_MAX.

PointCount

[*int*] Number of points entered before the command was ended. Valid range: 0 <= Value < TVG_INPUT_MAX.

Return Value

None.

Comment

This procedure is called if either all required points have been entered, or if TosoInputPointInitProc or TosoInputPointExitProc returned INPUT_FINISH, or if the command does not require any point entry at all. At this time, the module should perform all tasks to finish the command execution, like e.g. creating new objects or modifying existing object.

The name `TosoInputFinishProc` is only a placeholder, the procedure may have any name.

Related Topics:

TosoInputPointInitProc

TosoInputPointMoveProc

TosoInputPointExitProc

TosoInputDisplayProc

TosoCommandInternal (Command Handling)

Contents

Syntax

```
void TosoCommandInternal( int CommandID );
```

Executes a standard command of the serving application.

Contents

Parameters

CommandID

[*int*] Identification of the application's internal command to executed. For a list of all command IDs, see Command IDs.

Contents

Return Value

None.

Contents

Comment

If the executed command is a permanent command, i.e. a command that is active until the user selects another command, the procedure will return immediately after initialising the command! Only "direct" commands that do not require point entry will be completely finished when TosoCommandInternal returns.

Contents

Related Topics:

No directly related topic

TosoInputDrawPoint (Command Handling)

Syntax

```
void TosoInputDrawPoint( HDC hDrawDC,
                         double x,
                         double y );
```

Draws a single point marker (diagonally crossed lines) at the given position.

Parameters

hDrawDC

[*HDC*] Device context to draw to. This should be the device context passed to the current TosoInputDisplayProc.

x

y

[*double*] Coordinates of the point in internal millimeters (relative to the page center, scale-independent).

Return Value

None.

Comment

This command may only be called from within a TosoInputDisplayProc, using the device context passed to that procedure!

Related Topics:

TosoInputDrawLine

TosoInputDrawEndless

TosoInputDrawBezier

TosoInputDrawFrame

TosoInputDrawCircle

TosoInputDrawCircleArc

TosoInputDrawEllipse

TosoInputDrawEllipseArc

TosoInputDrawReference

TosoInputDrawLine (Command Handling)

Contents

Syntax

```
void TosoInputDrawLine( HDC hDrawDC,
                        double x1,
                        double y1,
                        double x2,
                        double y2 );
```

Draws a straight line between the given points.

Contents

Parameters

hDrawDC

[*HDC*] Device context to draw to. This should be the device context passed to the current TosoInputDisplayProc.

x1

y1

[*double*] Coordinates of the line's start-point in internal millimeters (relative to the page center, scale-independent).

x2

y2

[*double*] Coordinates of the line's end-point in internal millimeters (relative to the page center, scale-independent).

Contents

Return Value

None.

Contents

Comment

This command may only be called from within a TosoInputDisplayProc, using the device context passed to that procedure!

Contents

Related Topics:

Contents TosoInputDrawPoint

Contents TosoInputDrawEndless

Contents TosoInputDrawBezier

TosoInputDrawEndless (Command Handling)

Contents

Syntax

```
void TosoInputDrawEndless( HDC hDrawDC,
                           double x1,
                           double y1,
                           double x2,
                           double y2 );
```

Draws an endless line passing through the given points.

Contents

Parameters

hDrawDC

[*HDC*] Device context to draw to. This should be the device context passed to the current TosoInputDisplayProc.

x1

y1

[*double*] Coordinates of the line's first definition point in internal millimeters (relative to the page center, scale-independent).

x2

y2

[*double*] Coordinates of the line's second definition point in internal millimeters (relative to the page center, scale-independent).

Contents

Return Value

None.

Contents

Comment

This command may only be called from within a TosoInputDisplayProc, using the device context passed to that procedure!

Contents

Related Topics:

Contents TosoInputDrawPoint

Contents TosoInputDrawLine

Contents TosoInputDrawBezier

TosoInputDrawBezier (Command Handling)

Contents

Syntax

```
void TosoInputDrawBezier( HDC hDrawDC,
                          double x1,
                          double y1,
                          double x2,
                          double y2,
                          double x3,
                          double y3,
                          double x4,
                          double y4 );
```

Draws a bézier curve spanned by the given points.

Contents

Parameters

hDrawDC

[*HDC*] Device context to draw to. This should be the device context passed to the current TosoInputDisplayProc.

x1
y1

[*double*] Coordinates of the bezier curve's start-point in internal millimeters (relative to the page center, scale-independent).

x2
y2

[*double*] Coordinates of the bezier curve's first control point in internal millimeters (relative to the page center, scale-independent).

x3
y3

[*double*] Coordinates of the bezier curve's second control point in internal millimeters (relative to the page center, scale-independent).

x4
y4

[*double*] Coordinates of the bezier curve's end-point in internal millimeters (relative to the page center, scale-independent).

Contents

Return Value

None.

Contents

Comment

This command may only be called from within a TosoInputDisplayProc, using the device context passed to that procedure!

Contents

Related Topics:

TosoInputDrawPoint

TosoInputDrawLine

TosoInputDrawEndless

TosoInputDrawFrame

TosoInputDrawCircle

TosoInputDrawCircleArc

TosoInputDrawEllipse

TosoInputDrawEllipseArc

TosoInputDrawReference

TosoInputDrawFrame (Command Handling)

Contents

Syntax

```
void TosoInputDrawFrame( HDC hDrawDC,
                         double x1,
                         double y1,
                         double x2,
                         double y2,
                         double x3,
                         double y3,
                         double x4,
                         double y4 );
```

Draws a crossed quadrangle spanned by the given points.

Contents

Parameters

hDrawDC

[*HDC*] Device context to draw to. This should be the device context passed to the current TosoInputDisplayProc.

x1
y1

[*double*] Coordinates of the quadrangle's first corner-point in internal millimeters (relative to the page center, scale-independent).

x2
y2

[*double*] Coordinates of the quadrangle's second corner-point in internal millimeters (relative to the page center, scale-independent).

x3
y3

[*double*] Coordinates of the quadrangle's third corner-point in internal millimeters (relative to the page center, scale-independent).

x4
y4

[*double*] Coordinates of the quadrangle's fourth corner-point in internal millimeters (relative to the page center, scale-independent).

Contents

Return Value

None.

Contents

Comment

This command may only be called from within a TosoInputDisplayProc, using the device context passed to that procedure!

Contents

Related Topics:

TosoInputDrawPoint

TosoInputDrawLine

TosoInputDrawEndless

TosoInputDrawBezier

TosoInputDrawCircle

TosoInputDrawCircleArc

TosoInputDrawEllipse

TosoInputDrawEllipseArc

TosoInputDrawReference

TosoInputDrawCircle (Command Handling)

Contents

Syntax

```
void TosoInputDrawCircle( HDC hDrawDC,
                          double x1,
                          double y1,
                          double x2,
                          double y2 );
```

Draws a circle determined by its center-point and a point on the circle.

Contents

Parameters

hDrawDC

[*HDC*] Device context to draw to. This should be the device context passed to the current TosoInputDisplayProc.

x1
y1

[*double*] Coordinates of the circle's center-point in internal millimeters (relative to the page center, scale-independent).

x2
y2

[*double*] Coordinates of the circle's radius definition point in internal millimeters (relative to the page center, scale-independent).

Contents

Return Value

None.

Contents

Comment

This command may only be called from within a TosoInputDisplayProc, using the device context passed to that procedure!

Contents

Related Topics:

Contents TosoInputDrawPoint

Contents TosoInputDrawLine

Contents TosoInputDrawEndless

TosoInputDrawCircleArc (Command Handling)

Contents

Syntax

```
void TosoInputDrawCircleArc( HDC hDrawDC,
                             double x1,
                             double y1,
                             double x2,
                             double y2,
                             double x3,
                             double y3,
                             double x4,
                             double y4,
                             BOOL Positive );
```

Draws a circular arc determined by its center-point, a point on the circle and two angle definition points.

Contents

Parameters

hDrawDC

[*HDC*] Device context to draw to. This should be the device context passed to the current TosoInputDisplayProc.

x1
y1

[*double*] Coordinates of the circle's center-point in internal millimeters (relative to the page center, scale-independent).

x2
y2

[*double*] Coordinates of the circle's radius definition point in internal millimeters (relative to the page center, scale-independent).

x3
y3

[*double*] Coordinates of the circular arc's start-angle definition point in internal millimeters (relative to the page center, scale-independent).

x4
y4

[*double*] Coordinates of the circular arc's end-angle definition point in internal millimeters (relative to the page center, scale-independent).

Positive

[*BOOL*] If this value is TRUE, the arc is drawn in positive direction, i.e. counterclockwise. Else it is drawn in negative direction, i.e. clockwise.

Contents

Return Value

None.

Contents

Comment

This command may only be called from within a TosoInputDisplayProc, using the device context passed to that procedure!

Contents

Related Topics:

Contents TosoInputDrawPoint

Contents TosoInputDrawLine

Contents TosoInputDrawEndless

Contents TosoInputDrawBezier

Contents TosoInputDrawFrame

Contents TosoInputDrawCircle

Contents TosoInputDrawEllipse

Contents TosoInputDrawEllipseArc

Contents TosoInputDrawReference

TosoInputDrawEllipse (Command Handling)

Contents

Syntax

```
void TosoInputDrawEllipse( HDC hDrawDC,
                           double x1,
                           double y1,
                           double x2,
                           double y2,
                           double x3,
                           double y3 );
```

Draws an ellipse determined by its center-point and to spanning vectors' end-points.

Contents

Parameters

hDrawDC

[*HDC*] Device context to draw to. This should be the device context passed to the current TosoInputDisplayProc.

x1
y1

[*double*] Coordinates of the ellipse's center-point in internal millimeters (relative to the page center, scale-independent).

x2
y2

[*double*] Coordinates of the ellipse's first vector end-point in internal millimeters (relative to the page center, scale-independent).

x3
y3

[*double*] Coordinates of the ellipse's second vector end-point in internal millimeters (relative to the page center, scale-independent).

Contents

Return Value

None.

Contents

Comment

This command may only be called from within a TosoInputDisplayProc, using the device context passed to that procedure!

Contents

Related Topics:

Contents TosoInputDrawPoint

TosoInputDrawEllipseArc (Command Handling)

Contents **Syntax**

```
void TosoInputDrawEllipseArc( HDC hDrawDC,
                              double x1,
                              double y1,
                              double x2,
                              double y2,
                              double x3,
                              double y3,
                              double x4,
                              double y4,
                              double x5,
                              double y5,
                              BOOL Positive );
```

Draws an ellipse arc determined by its center-point, its spanning vectors' end-points and two angles.

Contents **Parameters**

hDrawDC

[*HDC*] Device context to draw to. This should be the device context passed to the current TosoInputDisplayProc.

x1
y1

[*double*] Coordinates of the ellipse's center-point in internal millimeters (relative to the page center, scale-independent).

x2
y2

[*double*] Coordinates of the ellipse's first vector end-point in internal millimeters (relative to the page center, scale-independent).

x3
y3

[*double*] Coordinates of the ellipse's second vector end-point in internal millimeters (relative to the page center, scale-independent).

x4
y4

[*double*] Coordinates of the ellipse arc's start-angle definition point in internal millimeters (relative to the page center, scale-independent).

x5
y5

[*double*] Coordinates of the ellipse arc's end-angle definition point in internal millimeters (relative to the page center, scale-independent).

Positive

[*BOOL*] If this value is TRUE, the arc is drawn in positive direction, i.e. counterclockwise. Else it is drawn in negative direction, i.e. clockwise.

Contents **Return Value**

None.

Contents **Comment**

This command may only be called from within a TosoInputDisplayProc, using the device context passed to that procedure!

Contents

Related Topics:

Contents TosoInputDrawPoint

Contents TosoInputDrawLine

Contents TosoInputDrawEndless

Contents TosoInputDrawBezier

Contents TosoInputDrawFrame

Contents TosoInputDrawCircle

Contents TosoInputDrawCircleArc

Contents TosoInputDrawEllipse

Contents TosoInputDrawReference

TosoInputDrawReference (Command Handling)

Syntax

```
void TosoInputDrawReference( HDC hDrawDC,
                             const GEO_OBJECT* GeoObject );
```

Draws a reference object based on the object pointed to by *GeoObject*.

Parameters

hDrawDC

[*HDC*] Device context to draw to. This should be the device context passed to the current TosoInputDisplayProc.

GeoObject

[*const GEO_OBJECT**] Reference object to be drawn.

Return Value

None.

Comment

This command may only be called from within a TosoInputDisplayProc, using the device context passed to that procedure!

Related Topics:

TosoInputDrawPoint

TosoInputDrawLine

TosoInputDrawEndless

TosoInputDrawBezier

TosoInputDrawFrame

TosoInputDrawCircle

TosoInputGetIdentData (Command Handling)

Contents Syntax

```
int TosoInputGetIdentData( int DrawingNum,
                           int PointIndex,
                           GEO_OBJECT* GeoObject );
```

Retrieves a standard object resulting from object-independent identification.

Contents Parameters

DrawingNum

[*int*] Zero-based index of the drawing in which the object shall be seeked. Valid range: 0 <= Value < TVG_DRAWING_MAX.

PointIndex

[*int*] Zero-based point index. This index states during which point's entry the identification was done. If, e.g., *PointIndex* is 1, the application will search for the object identified by entering the point with index 1, i.e. the second point. Valid range: 0 <= Value < TVG_INPUT_MAX.

GeoObject

[*GEO_OBJECT**] Address of a buffer to receive the retrieved identified object.

Contents Return Value

Returns the retrieved object's type or -1 if no valid object is found. Following object types are possible:

OBJ_LINE
Straight line.

OBJ_CIRCLE
Circle.

OBJ_ARC
Circular arc.

OBJ_ELLIPSE
Ellipse.

OBJ_EARC
Elliptical arc.

OBJ_GEOLINE
Geometry line (optionally printed object).

OBJ_GEOCIRCLE
Geometry circle (optionally printed object).

OBJ_GEOELLIPSE
Geometry ellipse (optionally printed object).

Contents Comment

The possible types of the returned object are limited by the respective point definition. Complex objects

like curves or dimensioning are automatically split into simple objects before returning one of those simple objects.

Object-independent identification is forced by using point types whose names begin with POINT_P. For a complete list of point types, see Point Types.

Related Topics:

TosoInputGetIdentObject

TosoInputGetIdentAddress

TosoInputGetGeneratedSurface

TosoInputGetIdentObject (Command Handling)

Syntax

```
int TosoInputGetIdentObject( int DrawingNum,
                             int PointIndex,
                             GEO_OBJECT* GeoObject );
```

Retrieves a standard object resulting from object-dependent identification.

Parameters

DrawingNum
: [*int*] Zero-based index of the drawing in which the object shall be seeked. Valid range: 0 <= Value < TVG_DRAWING_MAX.

PointIndex
: [*int*] Zero-based point index. This index states during which point's entry the identification was done. If, e.g., *PointIndex* is 1, the application will search for the object identified by entering the point with index 1, i.e. the second point. Valid range: 0 <= Value < TVG_INPUT_MAX.

GeoObject
: [*GEO_OBJECT**] Address of a buffer to receive the retrieved identified object.

Return Value

Returns the retrieved object's type or -1 if no valid object is found. Following object types are possible:

OBJ_LINE
: Straight line.

OBJ_CIRCLE
: Circle.

OBJ_ARC
: Circular arc.

OBJ_SECTOR
: Circular sector.

OBJ_SEGMENT
: Circular segment.

OBJ_ELLIPSE
: Ellipse.

OBJ_EARC
: Elliptical arc.

OBJ_ESECTOR
: Elliptical sector.

OBJ_ESEGMENT
: Elliptical segment.

OBJ_GEOLINE
: Geometry line (optionally printed object).

OBJ_GEOCIRCLE
Geometry circle (optionally printed object).

OBJ_GEOELLIPSE
Geometry ellipse (optionally printed object).

Contents

Comment

This procedure will only work if the identified object itself has one of the types listed as possible return values. In these cases, retrieving the identified object in form of a GEO_OBJECT is usually easier than retrieving the object's address and enumerating it.

Object-dependent identification is forced by using point types whose names begin with POINT_ID_ (but not with POINT_ID_MULTI). For a complete list of point types, see Point Types.

Contents

Related Topics:

Contents TosoInputGetIdentData

Contents TosoInputGetIdentAddress

Contents TosoInputGetGeneratedSurface

TosoInputGetIdentAddress (Command Handling)

Syntax

```
UNIT_PTR TosoInputGetIdentAddress( int DrawingNum,
                                   int PointIndex );
```

Retrieves the address of the unit identified by object-dependent identification. This will only work for single-unit identification.

Parameters

DrawingNum

[*int*] Zero-based index of the drawing in which the unit shall be seeked. Valid range: 0 <= Value < TVG_DRAWING_MAX.

PointIndex

[*int*] Zero-based point index. This index states during which point's entry the identification was done. If, e.g., *PointIndex* is 1, the application will search for the object identified by entering the point with index 1, i.e. the second point. Valid range: 0 <= Value < TVG_INPUT_MAX.

Return Value

Address of the identified unit or NULL, if no valid unit was found.

Comment

Object-dependent identification is forced by using point types whose names begin with POINT_ID_ (but not with POINT_ID_MULTI). For a complete list of point types, see Point Types.

Contents

Related Topics:

Contents TosoInputGetIdentData

Contents TosoInputGetIdentObject

Contents TosoInputGetGeneratedSurface

TosoInputGetGeneratedSurface (Command Handling)

Contents Syntax

```
int TosoInputGetGeneratedSurface( int DrawingNum,
                                  int PointIndex,
                                  int PointCount,
                                  UNIT_PTR ResultObj,
                                  int Bytes );
```

Generates and return a surface based on an object selection and several reference points. This will only work if the current command forced a multi-point identification.

Contents Parameters

DrawingNum

[*int*] Zero-based index of the drawing in which the unit shall be seeked. Valid range: 0 <= Value < TVG_DRAWING_MAX.

PointIndex

[*int*] Zero-based point index. This index states at which point entry the first reference point that was entered. Valid range: 0 <= Value < TVG_INPUT_DEF_MAX.

PointCount

[*int*] Number of total reference points, including the point defined by *PointIndex*. Valid range: 1 <= Value < TVG_INPUT_MAX.

ResultObj

[*UNIT_PTR*] Address of a buffer that is to receive the generated surface. If either *ResultObj* is NULL or *Bytes* is zero, this procedure will only return the number of bytes required to store the generated surface.

Bytes

[*int*] Size of the buffer pointed to by *ResultObj*, in bytes. If either *ResultObj* is NULL or *Bytes* is zero, this procedure will only return the number of bytes required to store the generated surface.

Contents Return Value

Number of bytes copied to *ResultObj*, if successful. If either *ResultObj* is NULL or *Bytes* is zero, this procedure will only return the number of bytes required to store the generated surface without actually returning the surface. If -1 is returned, an error occured. If 0 is returned, there was no surface to be generated (usually because the selected object did not form an closed surface).

Contents Comment

Multi-unit identification is forced by using point types whose names begin with POINT_ID_MULTI (but not with POINT_ID_MULTIPOINT). For a complete list of point types, see Point Types.

On how the generation of surfaces works, see the description of the command Trim > Trim Surface > Generate of the serving application.

Related Topics:


TosoInputGetIdentData


TosoInputGetIdentObject


TosoInputGetIdentAddress

TosoDrawingGetActive (Window and Display)

Syntax

```
int TosoDrawingGetActive( void );
```

Retrieves the index of the currently active drawing.

Parameters

None.

Return Value

Returns a zero-based index of the currently active drawing. Valid range: 0 <= Value < DRAWING_MAX.

Comment

Currently, the serving application can only handle one drawing at a time, so the returned value is always zero. This is subject to change, so be sure to always use `TosoDrawingGetActive` instead of a fix drawing index of zero!

Related Topics:

- TosoDrawingSetActive
- TosoWindowGetMode
- TosoWindowSetMode
- TosoWindowGetActive
- TosoWindowSetActive
- TosoWindowGetView
- TosoWindowSetViewAll
- TosoWindowSetViewPage

TosoDrawingSetActive (Window and Display)

Contents Syntax

```
BOOL TosoDrawingSetActive( int DrawingNum );
```

Activates the given drawing, i.e. brings its window(s) to the top and directs all further input to that drawing.

Contents Parameters

DrawingNum

[*int*] Zero-based index of the drawing that shall be activated. Valid range: 0 <= Value < TVG_DRAWING_MAX.

Contents Return Value

Returns TRUE if successful, else FALSE.

Contents Comment

Currently, the serving application can only handle one drawing at a time, so the returned value will be FALSE if *DrawingNum* is not zero. This is subject to change, so be prepared to handle multiple drawings!

Contents

Related Topics:

Contents TosoDrawingGetActive

Contents TosoWindowGetMode

Contents TosoWindowSetMode

Contents TosoWindowGetActive

Contents TosoWindowSetActive

Contents TosoWindowGetView

Contents TosoWindowSetViewAll

TosoWindowSetViewPage

TosoWindowSetViewScale

TosoWindowSetViewArea

TosoWindowGetMode (Window and Display)

Contents Syntax

```
int TosoWindowGetMode( int DrawingNum );
```

Retrieves the current window arrangement mode of the given drawing.

Contents Parameters

DrawingNum

[*int*] Zero-based index of the drawing whose window mode shall be returned. Valid range: 0 <= Value < TVG_DRAWING_MAX.

Contents Return Value

Returns the current window arrangement mode. Possible values are:

WINMODE_1
: 1 window (full size).

WINMODE_2X
: 2 windows (equal size, left and right).

WINMODE_2Y
: 2 windows (equal size, top and bottom).

WINMODE_3X
: 3 windows (large window left, 2 small windows right)

WINMODE_3Y
: 3 windows (large window top, 2 small windows bottom)

WINMODE_4
: 4 windows (equal size, 2 × 2)

WINMODE_4X
: 4 windows (large window left, 3 small windows right)

WINMODE_4Y
: 4 windows (large window top, 3 small windows bottom)

Contents Comment

None.

Contents

Related Topics:

Contents TosoDrawingGetActive

TosoWindowSetMode (Window and Display)

Contents Syntax

```
BOOL TosoWindowSetMode( int DrawingNum, int Mode );
```

Sets the window arrangement mode of the given drawing.

Contents Parameters

DrawingNum

[*int*] Zero-based index of the drawing whose window mode shall be set. Valid range: 0 <= Value < TVG_DRAWING_MAX.

Mode

[*int*] This value determines the new number and arrangement of windows visible on the screen. Possible values are:

WINMODE_1
: 1 window (full size).

WINMODE_2X
: 2 windows (equal size, left and right).

WINMODE_2Y
: 2 windows (equal size, top and bottom).

WINMODE_3X
: 3 windows (large window left, 2 small windows right)

WINMODE_3Y
: 3 windows (large window top, 2 small windows bottom)

WINMODE_4
: 4 windows (equal size, 2 × 2)

WINMODE_4X
: 4 windows (large window left, 3 small windows right)

WINMODE_4Y
: 4 windows (large window top, 3 small windows bottom)

Contents Return Value

Return TRUE if successful, else FALSE.

Contents Comment

None.

Contents

Related Topics:

TosoWindowGetActive (Window and Display)

Contents **Syntax**

```
int TosoWindowGetActive( int DrawingNum );
```

Retrieves the currently active window of the given drawing. This window does not necessarily have to be visible!

Contents **Parameters**

DrawingNum

[*int*] Zero-based index of the drawing whose active window shall be returned. Valid range: 0 <= Value < TVG_DRAWING_MAX.

Contents **Return Value**

Returns the index of the currently active window.

Contents **Comment**

None.

Contents

Related Topics:

Contents TosoDrawingGetActive

Contents TosoDrawingSetActive

Contents TosoWindowGetMode

Contents TosoWindowSetMode

Contents TosoWindowSetActive

Contents TosoWindowGetView

Contents TosoWindowSetViewAll

TosoWindowSetViewPage

TosoWindowSetViewScale

TosoWindowSetViewArea

TosoWindowSetActive (Window and Display)

Contents **Syntax**

```
BOOL TosoWindowSetActive( int DrawingNum, int WindowNum );
```

Sets the active window of the given drawing. This window does not necessarily have to be visible!

Contents **Parameters**

DrawingNum

[*int*] Zero-based index of the drawing whose active window shall be set. Valid range: 0 <= Value < TVG_WINDOW_MAX (standard windows) or Value = TVG_WINDOW_VIEW (view window).

WindowNum

[*int*] Zero-based index of the window that shall be activated. The windows are numbered in reading direction, i.e. left to right and top to bottom. Valid range: 0 <= Value < TVG_WINDOW_MAX (standard windows) or Value = TVG_WINDOW_VIEW (view window).

Contents **Return Value**

Return TRUE if successful, else FALSE.

Contents **Comment**

None.

Contents

Related Topics:

Contents TosoDrawingGetActive

Contents TosoDrawingSetActive

Contents TosoWindowGetMode

Contents TosoWindowSetMode

Contents TosoWindowGetActive

Contents TosoWindowGetView

TosoWindowGetView (Window and Display)

Syntax

```
BOOL TosoWindowGetView( int DrawingNum,
                        int WindowNum,
                        DPOINT* Center,
                        double* Zoom );
```

Retrieves the current view of the drawing that is visible in the given window.

Parameters

DrawingNum

[*int*] Zero-based index of the drawing whose view shall be returned. Valid range: 0 <= Value < TVG_DRAWING_MAX.

WindowNum

[*int*] Zero-based index of the window whose view shall be returned. The windows are numbered in reading direction, i.e. left to right and top to bottom. Valid range: 0 <= Value < TVG_WINDOW_MAX (standard windows) or Value = TVG_WINDOW_VIEW (view window).

Center

[*DPOINT**] Address of a point buffer to receive the current view's center point.

Zoom

[*double*] Address of a double buffer to receive the current view's zoom factor.

Return Value

Return TRUE if successful, else FALSE.

Comment

None.

Contents

Related Topics:

TosoDrawingGetActive

TosoDrawingSetActive

TosoWindowGetMode

TosoWindowSetMode

TosoWindowSetViewAll (Window and Display)

Contents Syntax

```
BOOL TosoWindowSetViewAll( int DrawingNum,
                           int WindowNum );
```

Alters the view in the given window so that all entities of the drawing will be visible.

Contents Parameters

DrawingNum

[*int*] Zero-based index of the drawing whose view shall be altered. Valid range: 0 <= Value < TVG_DRAWING_MAX.

WindowNum

[*int*] Zero-based index of the window whose view shall be altered. The windows are numbered in reading direction, i.e. left to right and top to bottom. Valid range: 0 <= Value < TVG_WINDOW_MAX (standard windows) or Value = TVG_WINDOW_VIEW (view window).

Contents Return Value

Return TRUE if successful, else FALSE.

Contents Comment

If no entity does exist in the stated drawing, the complete page will be displayed. Geometry lines will be ignore during the calculation of the view as they are endless.

Contents

Related Topics:

Contents TosoDrawingGetActive

Contents TosoDrawingSetActive

Contents TosoWindowGetMode

Contents TosoWindowSetMode

Contents TosoWindowGetActive

Contents TosoWindowSetActive

TosoWindowSetViewPage (Window and Display)

Contents **Syntax**

```
BOOL TosoWindowSetViewPage( int DrawingNum,
                            int WindowNum );
```

Alters the view in the given window so that the complete page will be visible.

Contents **Parameters**

DrawingNum

[*int*] Zero-based index of the drawing whose view shall be altered. Valid range: 0 <= Value < TVG_DRAWING_MAX.

WindowNum

[*int*] Zero-based index of the window whose view shall be altered. The windows are numbered in reading direction, i.e. left to right and top to bottom. Valid range: 0 <= Value < TVG_WINDOW_MAX (standard windows) or Value = TVG_WINDOW_VIEW (view window).

Contents **Return Value**

Return TRUE if successful, else FALSE.

Contents **Comment**

None.

Contents

Related Topics:

Contents TosoDrawingGetActive

Contents TosoDrawingSetActive

Contents TosoWindowGetMode

Contents TosoWindowSetMode

Contents TosoWindowGetActive

Contents TosoWindowSetActive

TosoWindowSetViewScale (Window and Display)

Contents **Syntax**

```
BOOL TosoWindowSetViewScale( int DrawingNum,
                             int WindowNum,
                             const DPOINT* Center,
                             double Zoom );
```

Alters the view in the given window so that the given point is in the window's center, and sets the zoom to the given value.

Contents **Parameters**

DrawingNum

[*int*] Zero-based index of the drawing whose view shall be altered. Valid range: 0 <= Value < TVG_DRAWING_MAX.

WindowNum

[*int*] Zero-based index of the window whose view shall be altered. The windows are numbered in reading direction, i.e. left to right and top to bottom. Valid range: 0 <= Value < TVG_WINDOW_MAX (standard windows) or Value = TVG_WINDOW_VIEW (view window).

Center

[*DPOINT*] Coordinates of the point that shall be displayed in the center of the drawing. The coordinates are in internal mm relative to the page's center.

Zoom

[*double*] Zoom factor to be used for display.

Contents **Return Value**

Return TRUE if successful, else FALSE.

Contents **Comment**

None.

Contents

Related Topics:

Contents TosoDrawingGetActive

Contents TosoDrawingSetActive

Contents TosoWindowGetMode

TosoWindowSetViewArea (Window and Display)

Contents Syntax

```
BOOL TosoWindowSetViewArea( int DrawingNum,
                            int WindowNum,
                            const DPOINT* Point1,
                            const DPOINT* Point2 );
```

Alters the view in the given window so that the rectangular region defined by the two corner-points will be visible.

Contents Parameters

DrawingNum

[*int*] Zero-based index of the drawing whose view shall be altered. Valid range: 0 <= Value < TVG_DRAWING_MAX.

WindowNum

[*int*] Zero-based index of the window whose view shall be altered. The windows are numbered in reading direction, i.e. left to right and top to bottom. Valid range: 0 <= Value < TVG_WINDOW_MAX (standard windows) or Value = TVG_WINDOW_VIEW (view window).

Point1,
Point2

[*DPOINT*] Corner-points of a rectangular region of the drawing that shall be completely visible. The coordinates are in internal mm relative to the page's center.

Contents Return Value

Return TRUE if successful, else FALSE.

Contents Comment

None.

Contents

Related Topics:

Contents TosoDrawingGetActive

Contents TosoDrawingSetActive

Contents TosoWindowGetMode

Contents TosoWindowSetMode

TosoDrawWindowAll (Window and Display)

Syntax

```
void TosoDrawWindowAll( void );
```

Redraws all currently visible drawing windows completely.

Parameters

None.

Return Value

None.

Comment

None.

Related Topics:

TosoDrawWindowArea

TosoDrawWindowSelection

TosoDrawNewObjects

TosoDrawLineDef

TosoDrawBlock

TosoDrawWindowArea (Window and Display)

Syntax

```
void TosoDrawWindowArea( const DRECT* Data );
```

Redraws a rectangular area in all currently visible drawing windows.

Parameters

Data

[*DRECT*] Size of the area to be redrawn. The values *Data->x1* and *Data->y1* contain the coordinates of the lower left corner, *Data->x2* and *Data->y2* the coordinates of the upper right corner of the area in internal millimeters.

Return Value

None.

Comment

None.

Related Topics:

TosoDrawWindowAll

TosoDrawWindowSelection

TosoDrawNewObjects

TosoDrawLineDef

TosoDrawBlock

TosoDrawWindowSelection (Window and Display)

Contents Syntax

```
void TosoDrawWindowSelection( BOOL ClearFlag );
```

Redraws an area containing all currently selected objects in all drawing windows, and optionally clears the objects' selection flag FLAG_USE.

Contents Parameters

ClearFlag

[*BOOL*] If this value is TRUE, the objects' selection flag (FLAG_USE) will be deleted before redraw, i.e. the objects will not be marked as "selected" any longer.

Contents Return Value

None.

Contents Comment

None.

Contents

Related Topics:

Contents TosoDrawWindowAll

Contents TosoDrawWindowArea

Contents TosoDrawNewObjects

Contents TosoDrawLineDef

Contents TosoDrawBlock

TosoDrawNewObjects (Window and Display)

Syntax

```
void TosoDrawNewObjects( void );
```

Redraws all newly created objects. This procedure may either be called within a TosoUndoInitProcess and TosoUndoFinishProcess bracket or after calling TosoUndoFinishProcess.

Parameters

None.

Return Value

None.

Comment

If some of the newly created objects are instances, or if blocks or groups have been created, make sure to call TosoUndoUpdateLinks *before* calling this procedure!

Related Topics:

TosoDrawWindowAll

TosoDrawWindowArea

TosoDrawWindowSelection

TosoDrawLineDef

TosoDrawBlock

TosoDrawLineDef (Window and Display)

Syntax

```
void TosoDrawLineDef( HWND hWindow,
                      const LINEDEF* LineDef );
```

Displays a line type ("line pattern") within the given window. The complete client area of the given window will be used.

Parameters

hWindow

[*HWND*] Handle of the window or control into whose client area the line pattern shall be drawn.

LineDef

[*const LINEDEF*] Line pattern definition to be drawn.

Return Value

None.

Comment

This procedure is usually used to display a line pattern inside a static control inside a dialog window. If the given line pattern definition is invalid, the window is cleared.

Related Topics:

TosoDrawWindowAll

TosoDrawWindowArea

TosoDrawWindowSelection

TosoDrawNewObjects

TosoDrawBlock

TosoDrawBlock (Window and Display)

Syntax

```
void TosoDrawBlock( HWND hWindow,
                    int DrawingNum,
                    const LPSTR BlockName,
                    const LPSTR LibraryName );
```

Displays a block within the given window. The complete client area of the window will be used.

Parameters

hWindow

[*HWND*] Handle of the window or control into whose client area the line pattern shall be drawn.

DrawingNum

[*int*] Zero-based index of the drawing in which the block is located (ignored if *LibraryName* is not equal to TVG_BLOCK_ID). Valid range: 0 <= Value < TVG_DRAWING_MAX.

BlockName

[*const LPSTR*] Name of the block to be drawn.

LibraryName

[*const LPSTR*] Name of the library the block is located in.

Return Value

None.

Comment

This procedure is usually used to display a block inside a static control inside a dialog window. If the stated block is not available in the cache, the window will be cleared.

Related Topics:

TosoDrawWindowAll

TosoDrawWindowArea

TosoDrawWindowSelection

TosoDrawNewObjects

Contents

TosoDialogHelpMessage (Dialog Window)

Contents Syntax

```
UINT TosoDialogHelpMessage( void );
```

Returns the global message ID that is used to inform a dialog window that the user wants help, i.e. that he has pressed the F1 button.

Contents Parameters

None.

Contents Return Value

Value of the global help message ID.

Contents Comment

The value returned by this procedure can be checked in the `default` case of a dialog window's main message switch:

```
BOOL CALLBACK DialogProc( HWND hDlg, UINT Message,
                          WPARAM wParam, LPARAM lParam )
{
  switch( Message ) {

      ...

    case WM_ENTERIDLE:
      return( TosoDialogEnterIdle( hDlg, wParam, lParam ) );

    default:
      if( Message == TosoDialogHelpMessage() ) {
        WinHelp( ... );
        return( TRUE );
      }
      break;
  }
  return( FALSE );
}
```

Contents

Related Topics:

Contents TosoDialogEnterIdle

Contents TosoDialogComboboxAdjust

TosoDialogEnterIdle (Dialog Window)

Contents Syntax

```
BOOL TosoDialogEnterIdle( HWND hDlg,
                          WPARAM wParam,
                          LPARAM lParam );
```

Handles the WM_ENTERIDLE message a dialog window, allowing the user to press F1 in order to get help on the current dialog window.

Contents Parameters

hDlg
[*HWND*] Handle of the dialog window that received the WM_ENTERIDLE message.

wParam
[*WPARAM*] First parameter that was passed with the WM_ENTERIDLE message.

lParam
[*LPARAM*] Second parameter that was passed with the WM_ENTERIDLE message.

Contents Return Value

Returns TRUE if the WM_ENTERIDLE message was processed, FALSE if not. This value should be returned from the dialog window's callback procedure to inform the system whether or not this message was processed.

Contents Comment

This procedure should be placed in the `WM_ENTERIDLE` case of all dialog windows' main message switches:

```
BOOL CALLBACK DialogProc( HWND hDlg, UINT Message,
                          WPARAM wParam, LPARAM lParam )
{
  switch( Message ) {

      ...

    case WM_ENTERIDLE:
      return( TosoDialogEnterIdle( hDlg, wParam, lParam ) );

    default:
      if( Message == TosoDialogHelpMessage() ) {
        WinHelp( ... );
        return( TRUE );
      }
      break;
  }
  return( FALSE );
}
```

Contents

Related Topics:

Contents TosoDialogHelpMessage

Contents TosoDialogComboboxAdjust

Contents TosoDialogCustomButtonColor

Contents TosoDialogCustomButtonText

Contents TosoDialogCustomButtonIcon

Contents TosoDialogCustomListboxColor

Contents TosoDialogCenter

TosoDialogComboboxAdjust (Dialog Window)

Syntax

```
void TosoDialogComboboxAdjust( HWND hDlg,
                               int CtlID );
```

Adjusts the size of a combo box according to the number of its entries. This enables the user to get a fast overview of all available choices in that combo box, and minimizes the need to scroll.

Parameters

hDlg

[*HWND*] Handle of the dialog window that contains the comobox control.

CtlID

[*int*] Identification number of the combobox control.

Return Value

None.

Comment

This procedure should be called in the `WM_INITDIALOG` case of all dialog windows that contain combo boxes for each of these combo boxes:

```
BOOL CALLBACK DialogProc( HWND hDlg, UINT Message,
                          WPARAM wParam, LPARAM lParam )
{
  switch( Message ) {
    case WM_INITDIALOG:
      ...

      for( Count = FirstCombo; Count <= LastCombo; Count++ )
        TosoDialogComboboxAdjust( hDlg, Count );

      ...

  }
  return( FALSE );
}
```

This also applies to sub-classed common dialog windows!

Related Topics:

TosoDialogHelpMessage

TosoDialogCustomButtonColor (Dialog Window)

Contents Syntax

```
void TosoDialogCustomButtonColor( const LPDRAWITEMSTRUCT DrawItem,
                                  COLORREF Color,
                                  BOOL Active );
```

Draws a custom button showing a color field.

Contents Parameters

DrawItem

[*const LPDRAWITEMSTRUCT*] Address of a structure that contains information about the custom button.

Color

[*COLORREF*] Color to be displayed within the button.

Active

[*BOOL*] If *Active* is TRUE, the button is displayed in permanently pressed state, which is an extension to the item states that are defined in Win32. The serving application uses this option to indicate the currently active setting in a row of option buttons.

Contents Return Value

None.

Contents Comment

This procedure will only use the *itemState*, *hDC* and *rcItem* elements of the *DRAWITEMSTRUCT* structure. If you need to draw a custom button outside a standard window or dialog, you can create your own *DRAWITEMSTRUCT* and fill in those three values.

In order to include your own custom controls, use buttons with the "Owner-Draw" style. Then, inside your dialog window command procedure, place the following code into the main message switch in order to draw the control:

```
case WM_DRAWITEM:
  {
    LPDRAWITEMSTRUCT pDrawItem;

    pDrawItem = (LPDRAWITEMSTRUCT) GET_LPARAM( wParam, lParam );
    switch( DrawItemPtr->CtlID ) {
      case IDD_CUSTOMCONTROL:
        TosoDialogCustomButtonColor( pDrawItem, Color, FALSE );
        return( TRUE );
    }
  }
  return( FALSE );
```

Whether the user pressed this button or not can be checked by including the control's ID in the standard

WM_COMMAND case. Such a custom control will behave like any standard button.

Related Topics:


TosoDialogHelpMessage


TosoDialogEnterIdle


TosoDialogComboboxAdjust


TosoDialogCustomButtonText


TosoDialogCustomButtonIcon


TosoDialogCustomListboxColor


TosoDialogCenter

TosoDialogCustomButtonText (Dialog Window)

Contents Syntax

```
void TosoDialogCustomButtonText( const LPDRAWITEMSTRUCT DrawItem,
                                 const LPSTR Text,
                                 BOOL Active );
```

Draws a custom button showing a left-aligned text and a right-aligned arrow.

Contents Parameters

DrawItem

[*const LPDRAWITEMSTRUCT*] Address of a structure that contains information about the custom button.

Text

[*const LPSTR*] Address of the text to be displayed in the button.

Active

[*BOOL*] If *Active* is TRUE, the button is displayed in permanently pressed state, which is an extension to the item states that are defined in Win32. The serving application uses this option to indicate the currently active setting in a row of option buttons.

Contents Return Value

None.

Contents Comment

This procedure will only use the *itemState*, *hDC* and *rcItem* elements of the *DRAWITEMSTRUCT* structure. If you need to draw a custom button outside a standard window or dialog, you can create your own *DRAWITEMSTRUCT* and fill in those three values.

In order to include your own custom controls, use buttons with the "Owner-Draw" style. Then, inside your dialog window command procedure, place the following code into the main message switch in order to draw the control:

```
case WM_DRAWITEM:
  {
    LPDRAWITEMSTRUCT pDrawItem;

    pDrawItem = (LPDRAWITEMSTRUCT) GET_LPARAM( wParam, lParam );
    switch( DrawItemPtr->CtlID ) {
      case IDD_CUSTOMCONTROL:
        TosoDialogCustomButtonText( pDrawItem, Text, FALSE );
        return( TRUE );
    }
  }
  return( FALSE );
```

Whether the user pressed this button or not can be checked by including the control's ID in the standard

WM_COMMAND case. Such a custom control will behave like any standard button.

Related Topics:


TosoDialogHelpMessage


TosoDialogEnterIdle


TosoDialogComboboxAdjust


TosoDialogCustomButtonColor


TosoDialogCustomButtonIcon


TosoDialogCustomListboxColor


TosoDialogCenter

TosoDialogCustomButtonIcon (Dialog Window)

Contents Syntax

```
void TosoDialogCustomButtonIcon( const LPDRAWITEMSTRUCT DrawItem,
                                 HICON hIcon,
                                 BOOL Active );
```

Draws a custom button showing an icon.

Contents Parameters

DrawItem

[*const LPDRAWITEMSTRUCT*] Address of a structure that contains information about the custom button.

hIcon

[*HICON*] Handle of the icon to be displayed within the button. This should be a 32 by 32 pixel icon!

Active

[*BOOL*] If *Active* is TRUE, the button is displayed in permanently pressed state, which is an extension to the item states that are defined in Win32. The serving application uses this option to indicate the currently active setting in a row of option buttons.

Contents Return Value

None.

Contents Comment

This procedure will only use the *itemState*, *hDC* and *rcItem* elements of the *DRAWITEMSTRUCT* structure. If you need to draw a custom button outside a standard window or dialog, you can create your own *DRAWITEMSTRUCT* and fill in those three values.

In order to include your own custom controls, use buttons with the "Owner-Draw" style. Then, inside your dialog window command procedure, place the following code into the main message switch in order to draw the control:

```
case WM_DRAWITEM:
  {
    LPDRAWITEMSTRUCT pDrawItem;

    pDrawItem = (LPDRAWITEMSTRUCT) GET_LPARAM( wParam, lParam );
    switch( DrawItemPtr->CtlID ) {
      case IDD_CUSTOMCONTROL:
        TosoDialogCustomButtonIcon( pDrawItem, hIcon, FALSE );
        return( TRUE );
    }
  }
  return( FALSE );
```

Whether the user pressed this button or not can be checked by including the control's ID in the standard

WM_COMMAND case. Such a custom control will behave like any standard button.

Related Topics:


TosoDialogHelpMessage


TosoDialogEnterIdle


TosoDialogComboboxAdjust


TosoDialogCustomButtonColor


TosoDialogCustomButtonText


TosoDialogCustomListboxColor


TosoDialogCenter

TosoDialogCustomListboxColor (Dialog Window)

Contents Syntax

```
void TosoDialogCustomListboxColor( const LPDRAWITEMSTRUCT DrawItem,
                                   COLORREF Color,
                                   const LPSTR Text );
```

Draws a listbox item showing a color field and a following text.

Contents Parameters

DrawItem

[*const LPDRAWITEMSTRUCT*] Address of a structure that contains information about the listbox item.

Color

[*COLORREF*] Color to be displayed in a field at the beginning of the item line.

Text

[*const LPSTR*] Address of the text to be displayed following the color field.

Contents Return Value

None.

Contents Comment

This procedure will only use the *itemState*, *hDC* and *rcItem* elements of the *DRAWITEMSTRUCT* structure. If you need to draw a listbox item outside a standard window or dialog, you can create your own *DRAWITEMSTRUCT* and fill in those three values.

In order to include your own custom controls, use listboxes with the "Owner-Draw" style. Then, inside your dialog window command procedure, place the following code into the main message switch in order to initialize and draw the control:

```
case WM_MEASUREITEM:
  {
    LPMEASUREITEMSTRUCT     MeasureItemPtr;

    MeasureItemPtr = (LPMEASUREITEMSTRUCT) GET_LPARAM( wParam, lParam );
    switch( MeasureItemPtr->CtlID ) {
      case IDD_CUSTOMCONTROL:
        MeasureItemPtr->itemHeight = ItemHeight;
        break;
  }
  return( TRUE );

case WM_COMPAREITEM:
  {
    LPCOMPAREITEMSTRUCT     CompareItemPtr;

    CompareItemPtr = (LPCOMPAREITEMSTRUCT) GET_LPARAM( wParam, lParam );
```

```
      switch( CompareItemPtr->CtlID ) {
        case IDD_CUSTOMCONTROL:

          // Compare two items in order to build a sorted listbox...

          break;
      }
      return( 0 );

    case WM_DRAWITEM:
      {
        LPDRAWITEMSTRUCT        DrawItemPtr;

        DrawItemPtr = (LPDRAWITEMSTRUCT) GET_LPARAM( wParam, lParam );
        switch( DrawItemPtr->CtlID ) {
          case IDD_CUSTOMCONTROL:
            TosoDialogCustomListboxColor( DrawItemPtr, Color, Text );
            return( TRUE );
        }
      }
      return( FALSE );
```

Whether the user selects a listbox item from such a listbox can be checked by including the control's ID in the standard WM_COMMAND case. Such a custom control will behave like any standard listbox.

Related Topics:


TosoDialogHelpMessage


TosoDialogEnterIdle


TosoDialogComboboxAdjust


TosoDialogCustomButtonColor


TosoDialogCustomButtonText


TosoDialogCustomButtonIcon


TosoDialogCenter

TosoDialogCenter (Dialog Window)

Syntax

```
void TosoDialogCenter( HWND hWindow );
```

Centers a dialog box relative to the application's main window.

Parameters

hWindow

[*HWND*] Handle of the dialog window to be centered.

Return Value

None.

Comment

This procedure should be called in the `WM_INITDIALOG` case of all dialog windows of the module in order to achieve a common dialog window positioning:

```
BOOL CALLBACK DialogProc( HWND hDlg, UINT Message,
                          WPARAM wParam, LPARAM lParam )
{
  switch( Message ) {
    case WM_INITDIALOG:
      TosoDialogCenter( hDlg );

      ...

  }
  return( FALSE );
}
```

This also applies to sub-classed common dialog windows!

Related Topics:

TosoDialogHelpMessage

TosoDialogEnterIdle

TosoDialogComboboxAdjust

TosoDialogSelection (Dialog Window)

Syntax

```
BOOL TosoDialogSelection( HWND hWindow,
                          const LPSTR Caption )
```

Shows and processes the dialog that allows editing the current selection parameters.

Parameters

hWindow

[*HWND*] Handle of the dialog window's parent window.

Caption

[*const LPSTR*] Title of the dialog window to be displayed. Be careful that the given title fits into the dialog window's caption.

Return Value

Returns TRUE if the dialog window was ended by pressing the OK button *and* anything inside the window was changed, else FALSE.

Comment

None.

Related Topics:

TosoDialogColor

TosoDialogProperty

TosoDialogXProperty

TosoDialogFontDef

TosoDialogDimLine

TosoDialogDimSmall

TosoDialogColor (Dialog Window)

Syntax

```
BOOL TosoDialogColor( HWND hWindow,
                      const LPSTR Caption,
                      COLORREF* Data );
```

Shows and processes a dialog for color selection (including custom colors).

Parameters

hWindow

[*HWND*] Handle of the dialog window's parent window.

Caption

[*const LPSTR*] Title of the dialog window to be displayed. Be careful that the given title fits into the dialog window's caption.

Data

[*COLORREF**] Address of a color value. This color value is used to initialize the dialog window, and it will receive the selected color if the user ends the dialog by pressing the OK button.

Return Value

Returns TRUE if the dialog window was ended by pressing the OK button *and* anything inside the window was changed, else FALSE.

Comment

None.

Related Topics:

TosoDialogSelection

TosoDialogProperty

TosoDialogXProperty

TosoDialogFontDef

TosoDialogDimLine

TosoDialogDimSmall

TosoDialogDimLarge

TosoDialogTextStandard

TosoDialogTextFrame

TosoDialogBlock

TosoDialogProperty (Dialog Window)

Syntax

```
BOOL TosoDialogProperty( HWND hWindow,
                         const LPSTR Caption,
                         PROPERTY* Data );
```

Shows and processes a dialog for standard property editing (without pen, transmission, and layer).

Parameters

hWindow

[*HWND*] Handle of the dialog window's parent window.

Caption

[*const LPSTR*] Title of the dialog window to be displayed. Be careful that the given title fits into the dialog window's caption.

Data

[*PROPERTY**] Address of a property set. This property set is used to initialize the dialog window, and it will receive the selected property set if the user ends the dialog by pressing the OK button.

Return Value

Returns TRUE if the dialog window was ended by pressing the OK button *and* anything inside the window was changed, else FALSE.

Comment

None.

Related Topics:

TosoDialogSelection

TosoDialogColor

TosoDialogXProperty

TosoDialogFontDef

TosoDialogDimLine

TosoDialogDimSmall

TosoDialogDimLarge

TosoDialogTextStandard

TosoDialogTextFrame

TosoDialogBlock

TosoDialogXProperty (Dialog Window)

Syntax

```
BOOL TosoDialogXProperty( HWND hWindow,
                          const LPSTR Caption,
                          XPROPERTY* Data );
```

Shows and processes a dialog for extended property editing (including pen, transmission, and layer).

Parameters

hWindow

[*HWND*] Handle of the dialog window's parent window.

Caption

[*const LPSTR*] Title of the dialog window to be displayed. Be careful that the given title fits into the dialog window's caption.

Data

[*XPROPERTY**] Address of an extended property set. This extended property set is used to initialize the dialog window, and it will receive the selected extended property set if the user ends the dialog by pressing the OK button.

Return Value

Returns TRUE if the dialog window was ended by pressing the OK button *and* anything inside the window was changed, else FALSE.

Comment

None.

Related Topics:

TosoDialogSelection

TosoDialogColor

TosoDialogProperty

TosoDialogFontDef

TosoDialogDimLine

TosoDialogDimSmall

TosoDialogDimLarge

TosoDialogTextStandard

TosoDialogTextFrame

TosoDialogBlock

TosoDialogFontDef (Dialog Window)

Syntax

```
BOOL TosoDialogFontDef( HWND hWindow,
                        const LPSTR Caption,
                        FONTDEF* Data );
```

Shows and processes a dialog for font definition editing.

Parameters

hWindow

[*HWND*] Handle of the dialog window's parent window.

Caption

[*const LPSTR*] Title of the dialog window to be displayed. Be careful that the given title fits into the dialog window's caption.

Data

[*FONTDEF**] Address of a font definition. This font definition is used to initialize the dialog window, and it will receive the selected font definition if the user ends the dialog by pressing the OK button.

Return Value

Returns TRUE if the dialog window was ended by pressing the OK button *and* anything inside the window was changed, else FALSE.

Comment

None.

Related Topics:

TosoDialogSelection

TosoDialogColor

TosoDialogProperty

TosoDialogXProperty

TosoDialogDimLine

TosoDialogDimSmall

TosoDialogDimLarge

TosoDialogTextStandard

TosoDialogTextFrame

TosoDialogBlock

TosoDialogDimLine (Dialog Window)

Syntax

```
BOOL TosoDialogDimLine( HWND hWindow,
                        const LPSTR Caption,
                        DIMLINE* Data,
                        int UseFlag );
```

Shows and processes a dialog for dimension line parameter editing.

Parameters

hWindow

[*HWND*] Handle of the dialog window's parent window.

Caption

[*const LPSTR*] Title of the dialog window to be displayed. Be careful that the given title fits into the dialog window's caption.

Data

[*DIMLINE**] Address of a dimension line parameter set. These parameters will be used to initialize the dialog window, and they will receive the selected parameters if the user ends the dialog by pressing the OK button.

UseFlag

[*int*] This parameter is a bitwise OR combination of several flags that enable single settings of this dialog window. For each flag that is set, one control in the dialog window is enabled. By this means, the module can control which settings can be altered by the user. The following flags are defined:

DIMMODE_ARROWSTARTFORM
: If this flag is set, the control corresponding to the *ArrowStartForm* element will be enabled.

DIMMODE_ARROWSTARTMODE
: If this flag is set, the control corresponding to the *ArrowStartMode* element will be enabled.

DIMMODE_ARROWENDFORM
: If this flag is set, the control corresponding to the *ArrowEndForm* element will be enabled.

DIMMODE_ARROWENDMODE
: If this flag is set, the control corresponding to the *ArrowEndMode* element will be enabled.

Return Value

Returns TRUE if the dialog window was ended by pressing the OK button *and* anything inside the window was changed, else FALSE.

Comment

None.

Related Topics:

TosoDialogSelection

TosoDialogColor

TosoDialogProperty

TosoDialogXProperty

TosoDialogFontDef

TosoDialogDimSmall

TosoDialogDimLarge

TosoDialogTextStandard

TosoDialogTextFrame

TosoDialogBlock

TosoDialogDimSmall (Dialog Window)

Syntax

```
BOOL TosoDialogDimSmall( HWND hWindow,
                         const LPSTR Caption,
                         DIMSMALL* Data,
                         BOOL UseGlobal,
                         inr UseFlag );
```

Shows and processes a dialog for small dimension parameter editing.

Parameters

hWindow

[*HWND*] Handle of the dialog window's parent window.

Caption

[*const LPSTR*] Title of the dialog window to be displayed. Be careful that the given title fits into the dialog window's caption.

Data

[*DIMSMALL**] Address of a small dimension parameter set. These parameters will be used to initialize the dialog window, and they will receive the selected parameters if the user ends the dialog by pressing the OK button.

UseGlobal

[*BOOL*] If this value is TRUE, the elements *TextFont*, *TextSize1*, *TextSize2*, *NumAccuracy* and *System* of the structure pointed to by *Data* will not be used to initialized the dialog window, instead, the current global dimension parameters will be used. If the user edits these values, the changes will be stored in the global dimension parameters as well as in the structure pointed to by *Data*.

UseFlag

[*int*] This parameter is a bitwise OR combination of several flags that enable single settings of this dialog window. For each flag that is set, one control in the dialog window is enabled. By this means, the module can control which settings can be altered by the user. The following flags are defined:

DIMMODE_NUMREFRESH

If this flag is set, the control corresponding to the *NumRefresh* element will be enabled.

DIMMODE_NUMROTATE

If this flag is set, the control corresponding to the *NumRotate* element will be enabled.

DIMMODE_GLOBALPARAM

If this flag is set, the global dimension parameter button will be enabled (allowing to edit the *TextFont*, *TextSize1*, *TextSize2*, *NumAccuracy* and *System* elements).

Return Value

Returns TRUE if the dialog window was ended by pressing the OK button *and* anything inside the window was changed, else FALSE.

Comment

None.

Related Topics:

 TosoDialogSelection

 TosoDialogColor

 TosoDialogProperty

 TosoDialogXProperty

 TosoDialogFontDef

 TosoDialogDimLine

 TosoDialogDimLarge

 TosoDialogTextStandard

 TosoDialogTextFrame

 TosoDialogBlock

TosoDialogDimLarge (Dialog Window)

Contents Syntax

```
BOOL TosoDialogDimLarge( HWND hWindow,
                         const LPSTR Caption,
                         DIMLARGE* Data,
                         BOOL UseGlobal,
                         int UseFlag );
```

Shows and processes a dialog for large dimension parameter editing.

Contents Parameters

hWindow

[*HWND*] Handle of the dialog window's parent window.

Caption

[*const LPSTR*] Title of the dialog window to be displayed. Be careful that the given title fits into the dialog window's caption.

Data

[*DIMLARGE**] Address of a large dimension parameter set. These parameters will be used to initialize the dialog window, and they will receive the selected parameters if the user ends the dialog by pressing the OK button.

UseGlobal

[*BOOL*] If this value is TRUE, the elements *TextFont*, *TextSize1*, *TextSize2*, *NumAccuracy* and *System* of the structure pointed to by *Data* will not be used to initialized the dialog window, instead, the current global dimension parameters will be used. If the user edits these values, the changes will be stored in the global dimension parameters as well as in the structure pointed to by *Data*.

UseFlag

[*int*] This parameter is a bitwise OR combination of several flags that enable single settings of this dialog window. For each flag that is set, one control in the dialog window is enabled. By this means, the module can control which settings can be altered by the user. The following flags are defined:

DIMMODE_NUMREFRESH

If this flag is set, the control corresponding to the *NumRefresh* element will be enabled.

DIMMODE_NUMCENTERED

If this flag is set, the control corresponding to the *NumCentered* element will be enabled.

DIMMODE_NUMTIGHT

If this flag is set, the control corresponding to the *NumTight* element will be enabled.

DIMMODE_NUMROTATE

If this flag is set, the control corresponding to the *NumRotate* element will be enabled.

DIMMODE_ARROWSTARTFORM

If this flag is set, the control corresponding to the *ArrowStartForm* element will be enabled.

DIMMODE_ARROWSTARTMODE

If this flag is set, the control corresponding to the *ArrowStartMode* element will be enabled.

DIMMODE_ARROWENDFORM

If this flag is set, the control corresponding to the *ArrowEndForm* element will be enabled.

DIMMODE_ARROWENDMODE

If this flag is set, the control corresponding to the *ArrowEndMode* element will be enabled.

DIMMODE_EXTSTARTDISPLAY
If this flag is set, the control corresponding to the *ExtStartDisplay* element will be enabled.

DIMMODE_EXTENDDISPLAY
If this flag is set, the control corresponding to the *ExtEndDisplay* element will be enabled.

DIMMODE_LINEDISPLAY
If this flag is set, the control corresponding to the *LineDisplay* element will be enabled.

DIMMODE_LINEORIENTATION
If this flag is set, the control corresponding to the *LineOrientation* element will be enabled.

DIMMODE_LINETYPE
If this flag is set, the control corresponding to the *LineType* element will be enabled.

DIMMODE_LINEDISTMODE
If this flag is set, the control corresponding to the *LineDistMode* element will be enabled.

DIMMODE_LINEDISTANCE
If this flag is set, the control corresponding to the *LineDistance* element will be enabled.

DIMMODE_GLOBALPARAM
If this flag is set, the global dimension parameter button will be enabled (allowing to edit the *TextFont*, *TextSize1*, *TextSize2*, *NumAccuracy* and *System* elements).

Return Value

Returns TRUE if the dialog window was ended by pressing the OK button *and* anything inside the window was changed, else FALSE.

Comment

None.

Related Topics:


TosoDialogSelection


TosoDialogColor


TosoDialogProperty


TosoDialogXProperty


TosoDialogFontDef

TosoDialogTextStandard (Dialog Window)

Contents

Syntax

```
BOOL TosoDialogTextStandard( HWND hWindow,
                             const LPSTR Caption,
                             TEXTSTANDARD* Data1,
                             TEXTREFERENCE* Data2,
                             LPSTR EditText );
```

Shows and processes a dialog for standard text editing (including text parameters).

Contents

Parameters

hWindow

[*HWND*] Handle of the dialog window's parent window.

Caption

[*const LPSTR*] Title of the dialog window to be displayed. Be careful that the given title fits into the dialog window's caption.

Data1

[*TEXTSTANDARD**] Address of a standard text parameter set. If this address is not NULL, the standard text parameters will be used to initialize the dialog window, and they will receive the selected standard text parameters if the user ends the dialog by pressing the OK button.

Data2

[*TEXTREFERENCE**] Address of a reference text parameter set. If this address is not NULL, the reference text parameters will be used to initialize the dialog window, and they will receive the selected reference text parameters if the user ends the dialog by pressing the OK button.

EditText

[*LPSTR*] Address of a text. If this address is not NULL, the text will be used to initialize the dialog window, and it will receive the edited text if the user ends the dialog by pressing the OK button. Allow at least NAME_LENGTH_TEXTLONG bytes!

Contents

Return Value

Returns TRUE if the dialog window was ended by pressing the OK button *and* anything inside the window was changed, else FALSE.

Contents

Comment

Depending on the values of *Data1*, *Data2* and *EditText*, the dialog does have an edit control, controls to edit the standard parameters and/or a button to branch to reference text parameter editing. If e.g. only *EditText* is non-NULL, a simple text editing control appears.

Contents

Related Topics:

TosoDialogTextFrame (Dialog Window)

Syntax

```
BOOL TosoDialogTextFrame( HWND hWindow,
                          const LPSTR Caption,
                          TEXTFRAME* Data,
                          LPSTR EditText );
```

Shows and processes a dialog for frame text editing (including text parameters).

Parameters

hWindow
: [*HWND*] Handle of the dialog window's parent window.

Caption
: [*const LPSTR*] Title of the dialog window to be displayed. Be careful that the given title fits into the dialog window's caption.

Data
: [*TEXTFRAME**] Address of a frame text parameter set. If this address is not NULL, the frame text parameters will be used to initialize the dialog window, and they will receive the selected frame text parameters if the user ends the dialog by pressing the OK button.

EditText
: [*LPSTR*] Address of a text. If this address is not NULL, the text will be used to initialize the dialog window, and it will receive the edited text if the user ends the dialog by pressing the OK button. Allow at least NAME_LENGTH_TEXTLONG bytes!

Return Value

Returns TRUE if the dialog window was ended by pressing the OK button *and* anything inside the window was changed, else FALSE.

Comment

Depending on the values of *Data* and *EditText*, the dialog does have an edit control and/or controls to edit the parameters. If only *EditText* is non-NULL, a simple text editing control appears.

Related Topics:

TosoDialogSelection

TosoDialogColor

TosoDialogProperty

TosoDialogBlock (Dialog Window)

Contents

Syntax

```
BOOL TosoDialogBlock( HWND hWindow,
                      const LPSTR Caption,
                      LPSTR BlockName,
                      LPSTR LibraryName,
                      BOOL LibChangePossible );
```

Shows and processes a dialog for block selection.

Contents

Parameters

hWindow

[*HWND*] Handle of the dialog window's parent window.

Caption

[*const LPSTR*] Title of the dialog window to be displayed. Be careful that the given title fits into the dialog window's caption.

BlockName

[*LPSTR*] Address of a block name. This block name will be used to initialize the dialog window, and it will receive the selected block name if the user ends the dialog by pressing the OK button. Allow at least NAME_LENGTH_BLOCK bytes!

LibraryName

[*LPSTR*] Address of a library name. This library name will be used to initialize the dialog window, and it will receive the selected library name if the user ends the dialog by pressing the OK button. Allow at least NAME_LENGTH_TITLE bytes!

LibChangePossible

[*BOOL*] If *LibChangePossible* is FALSE, the user is not allowed to change the preset library name. This can used to force the user to select a block (and *not* a smybol).

Contents

Return Value

Returns TRUE if the dialog window was ended by pressing the OK button *and* anything inside the window was changed, else FALSE.

Contents

Comment

None.

Contents

Related Topics:

Contents TosoDialogSelection

Contents TosoDialogColor

TosoDialogSelectLine (Dialog Window)

Contents Syntax

```
void TosoDialogSelectLine( HWND hWnd,
                           const RECT* ButtonRect,
                           int Current,
                           int BasicID,
                           int ExtraID,
                           int EditID );
```

Displays a popup menu offering all available line patterns for user selection. Resulting from the user selection, a command message will be sent to the parent window or dialog window of the selection popup menu.

Contents Parameters

hWnd

[*HWND*] Handle of the window or dialog window that is the parent of the selection, i.e. that will receive the selection messages.

ButtonRect

[*const RECT**] Address of a rectangle structure that contains the bounding rectangle of a button that is used to open the selection. The appearing popup menu will be aligned to that button. If *ButtonRect* is NULL, the popup menu will be center to the current mouse cursor position.

Current

[*int*] Index of the currently active line pattern. This index will be used to set a checkmark in front of the corresponding menu entry. Set this value to -1 in order to set the checkmark in front of the entry "Current" (if *ExtraID* is non-zero) or to any value below -1 in order to set no checkmark at all.

BasicID

[*int*] Basic command ID of the popup menu's entries. If the user selects the line pattern with the index zero, the resulting command ID will be *BasicID*. If the line pattern's index is 5, the resulting command ID will be *BasicID+5* and so on.

ExtraID

[*int*] Command ID of the popup menu's "Current" entry. If this value is non-zero, the popup menu will contain the entry "Current". If the user selects that entry, the resulting command ID will be *ExtraID*. If *ExtraID* is zero, the popup menu will not contain the entry "Current".

EditID

[*int*] Command ID of the popup menu's "Edit..." entry. If this value is non-zero, the popup menu will contain the entry "Edit...". If the user selects that entry, the resulting command ID will be *EditID*. If *EditID* is zero, the popup menu will not contain the entry "Edit...".

Contents Return Value

None.

Contents Comment

The user selection will be passed to the parent window using a WM_COMMAND message and a command ID based on *BasicID*, *ExtraID* and *EditID*. The user can cancel the menu by pressing the ESC

or the ALT key or by clicking outside the menu. In this case, no command ID will be passed to the parent handle, i.e. one may not rely on always receiving a command ID after calling this procedure!

Related Topics:

TosoDialogSelectMultiLine

TosoDialogSelectSystem

TosoDialogSelectHatch

TosoDialogSelectLayer

TosoDialogSelectPen

TosoDialogSelectMultiLine (Dialog Window)

Contents

Syntax

```
void TosoDialogSelectMultiLine( HWND hWnd,
                                const RECT* ButtonRect,
                                int Current,
                                int BasicID,
                                int ExtraID,
                                int EditID );
```

Displays a popup menu offering all available line sequences for user selection. Resulting from the user selection, a command message will be sent to the parent window or dialog window of the selection popup menu.

Contents

Parameters

hWnd

[*HWND*] Handle of the window or dialog window that is the parent of the selection, i.e. that will receive the selection messages.

ButtonRect

[*const RECT**] Address of a rectangle structure that contains the bounding rectangle of a button that is used to open the selection. The appearing popup menu will be aligned to that button. If *ButtonRect* is NULL, the popup menu will be center to the current mouse cursor position.

Current

[*int*] Index of the currently active line sequence. This index will be used to set a checkmark in front of the corresponding menu entry. Set this value to -1 in order to set the checkmark in front of the entry "Current" (if *ExtraID* is non-zero) or to any value below -1 in order to set no checkmark at all.

BasicID

[*int*] Basic command ID of the popup menu's entries. If the user selects the line sequence with the index zero, the resulting command ID will be *BasicID*. If the line sequence's index is 5, the resulting command ID will be *BasicID+5* and so on.

ExtraID

[*int*] Command ID of the popup menu's "Current" entry. If this value is non-zero, the popup menu will contain the entry "Current". If the user selects that entry, the resulting command ID will be *ExtraID*. If *ExtraID* is zero, the popup menu will not contain the entry "Current".

EditID

[*int*] Command ID of the popup menu's "Edit..." entry. If this value is non-zero, the popup menu will contain the entry "Edit...". If the user selects that entry, the resulting command ID will be *EditID*. If *EditID* is zero, the popup menu will not contain the entry "Edit...".

Contents

Return Value

None.

Contents

Comment

The user selection will be passed to the parent window using a WM_COMMAND message and a command ID based on *BasicID*, *ExtraID* and *EditID*. The user can cancel the menu by pressing the ESC

or the ALT key or by clicking outside the menu. In this case, no command ID will be passed to the parent handle, i.e. one may not rely on always receiving a command ID after calling this procedure!

Related Topics:

TosoDialogSelectLine

TosoDialogSelectSystem

TosoDialogSelectHatch

TosoDialogSelectLayer

TosoDialogSelectPen

TosoDialogSelectSystem (Dialog Window)

Contents

Syntax

```
void TosoDialogSelectSystem( HWND hWnd,
                             const RECT* ButtonRect,
                             int Current,
                             int BasicID,
                             int ExtraID,
                             int EditID );
```

Displays a popup menu offering all available coordinate systems for user selection. Resulting from the user selection, a command message will be sent to the parent window or dialog window of the selection popup menu.

Contents

Parameters

hWnd

[*HWND*] Handle of the window or dialog window that is the parent of the selection, i.e. that will receive the selection messages.

ButtonRect

[*const RECT**] Address of a rectangle structure that contains the bounding rectangle of a button that is used to open the selection. The appearing popup menu will be aligned to that button. If *ButtonRect* is NULL, the popup menu will be center to the current mouse cursor position.

Current

[*int*] Index of the currently active coordinate system. This index will be used to set a checkmark in front of the corresponding menu entry. Set this value to -1 in order to set the checkmark in front of the entry "Current" (if *ExtraID* is non-zero) or to any value below -1 in order to set no checkmark at all.

BasicID

[*int*] Basic command ID of the popup menu's entries. If the user selects the coordinate system with the index zero, the resulting command ID will be *BasicID*. If the coordinate system's index is 5, the resulting command ID will be *BasicID+5* and so on.

ExtraID

[*int*] Command ID of the popup menu's "Current" entry. If this value is non-zero, the popup menu will contain the entry "Current". If the user selects that entry, the resulting command ID will be *ExtraID*. If *ExtraID* is zero, the popup menu will not contain the entry "Current".

EditID

[*int*] Command ID of the popup menu's "Edit..." entry. If this value is non-zero, the popup menu will contain the entry "Edit...". If the user selects that entry, the resulting command ID will be *EditID*. If *EditID* is zero, the popup menu will not contain the entry "Edit...".

Contents

Return Value

None.

Contents

Comment

The user selection will be passed to the parent window using a WM_COMMAND message and a command ID based on *BasicID*, *ExtraID* and *EditID*. The user can cancel the menu by pressing the ESC

or the ALT key or by clicking outside the menu. In this case, no command ID will be passed to the parent handle, i.e. one may not rely on always receiving a command ID after calling this procedure!

Related Topics:

TosoDialogSelectLine

TosoDialogSelectMultiLine

TosoDialogSelectHatch

TosoDialogSelectLayer

TosoDialogSelectPen

TosoDialogSelectHatch (Dialog Window)

Contents

Syntax

```
void TosoDialogSelectHatch( HWND hWnd,
                            const RECT* ButtonRect,
                            int Current,
                            int BasicID,
                            int ExtraID,
                            int EditID );
```

Displays a popup menu offering all available hatching types for user selection. Resulting from the user selection, a command message will be sent to the parent window or dialog window of the selection popup menu.

Contents

Parameters

hWnd

[*HWND*] Handle of the window or dialog window that is the parent of the selection, i.e. that will receive the selection messages.

ButtonRect

[*const RECT**] Address of a rectangle structure that contains the bounding rectangle of a button that is used to open the selection. The appearing popup menu will be aligned to that button. If *ButtonRect* is NULL, the popup menu will be center to the current mouse cursor position.

Current

[*int*] Index of the currently active hatching type. This index will be used to set a checkmark in front of the corresponding menu entry. Set this value to -1 in order to set the checkmark in front of the entry "Current" (if *ExtraID* is non-zero) or to any value below -1 in order to set no checkmark at all.

BasicID

[*int*] Basic command ID of the popup menu's entries. If the user selects the hatching type with the index zero, the resulting command ID will be *BasicID*. If the hatching type's index is 5, the resulting command ID will be *BasicID+5* and so on.

ExtraID

[*int*] Command ID of the popup menu's "Current" entry. If this value is non-zero, the popup menu will contain the entry "Current". If the user selects that entry, the resulting command ID will be *ExtraID*. If *ExtraID* is zero, the popup menu will not contain the entry "Current".

EditID

[*int*] Command ID of the popup menu's "Edit..." entry. If this value is non-zero, the popup menu will contain the entry "Edit...". If the user selects that entry, the resulting command ID will be *EditID*. If *EditID* is zero, the popup menu will not contain the entry "Edit...".

Contents

Return Value

None.

Contents

Comment

The user selection will be passed to the parent window using a WM_COMMAND message and a command ID based on *BasicID*, *ExtraID* and *EditID*. The user can cancel the menu by pressing the ESC

or the ALT key or by clicking outside the menu. In this case, no command ID will be passed to the parent handle, i.e. one may not rely on always receiving a command ID after calling this procedure!

Related Topics:

TosoDialogSelectLine

TosoDialogSelectMultiLine

TosoDialogSelectSystem

TosoDialogSelectLayer

TosoDialogSelectPen

TosoDialogSelectLayer (Dialog Window)

Syntax

```
void TosoDialogSelectLayer( HWND hWnd,
                            const RECT* ButtonRect,
                            int Current,
                            int BasicID,
                            int ExtraID,
                            int EditID );
```

Displays a popup menu offering all available layers for user selection. Resulting from the user selection, a command message will be sent to the parent window or dialog window of the selection popup menu.

Parameters

hWnd

[*HWND*] Handle of the window or dialog window that is the parent of the selection, i.e. that will receive the selection messages.

ButtonRect

[*const RECT**] Address of a rectangle structure that contains the bounding rectangle of a button that is used to open the selection. The appearing popup menu will be aligned to that button. If *ButtonRect* is NULL, the popup menu will be center to the current mouse cursor position.

Current

[*int*] Index of the currently active layer. This index will be used to set a checkmark in front of the corresponding menu entry. Set this value to -1 in order to set the checkmark in front of the entry "Current" (if *ExtraID* is non-zero) or to any value below -1 in order to set no checkmark at all.

BasicID

[*int*] Basic command ID of the popup menu's entries. If the user selects the layer with the index zero, the resulting command ID will be *BasicID*. If the layer's index is 5, the resulting command ID will be *BasicID+5* and so on.

ExtraID

[*int*] Command ID of the popup menu's "Current" entry. If this value is non-zero, the popup menu will contain the entry "Current". If the user selects that entry, the resulting command ID will be *ExtraID*. If *ExtraID* is zero, the popup menu will not contain the entry "Current".

EditID

[*int*] Command ID of the popup menu's "Edit..." entry. If this value is non-zero, the popup menu will contain the entry "Edit...". If the user selects that entry, the resulting command ID will be *EditID*. If *EditID* is zero, the popup menu will not contain the entry "Edit...".

Return Value

None.

Comment

The user selection will be passed to the parent window using a WM_COMMAND message and a command ID based on *BasicID*, *ExtraID* and *EditID*. The user can cancel the menu by pressing the ESC or the ALT key or by clicking outside the menu. In this case, no command ID will be passed to the parent

handle, i.e. one may not rely on always receiving a command ID after calling this procedure!

Related Topics:

TosoDialogSelectLine

TosoDialogSelectMultiLine

TosoDialogSelectSystem

TosoDialogSelectHatch

TosoDialogSelectPen

TosoDialogSelectPen (Dialog Window)

Contents Syntax

```
void TosoDialogSelectPen( HWND hWnd,
                          const RECT* ButtonRect,
                          int Current,
                          int BasicID,
                          int ExtraID,
                          int EditID );
```

Displays a popup menu offering all available pens for user selection. Resulting from the user selection, a command message will be sent to the parent window or dialog window of the selection popup menu.

Contents Parameters

hWnd

[*HWND*] Handle of the window or dialog window that is the parent of the selection, i.e. that will receive the selection messages.

ButtonRect

[*const RECT**] Address of a rectangle structure that contains the bounding rectangle of a button that is used to open the selection. The appearing popup menu will be aligned to that button. If *ButtonRect* is NULL, the popup menu will be center to the current mouse cursor position.

Current

[*int*] Index of the currently active pen. This index will be used to set a checkmark in front of the corresponding menu entry. Set this value to -1 in order to set the checkmark in front of the entry "Current" (if *ExtraID* is non-zero) or to any value below -1 in order to set no checkmark at all.

BasicID

[*int*] Basic command ID of the popup menu's entries. If the user selects the pen with the index zero, the resulting command ID will be *BasicID*. If the pen's index is 5, the resulting command ID will be *BasicID+5* and so on.

ExtraID

[*int*] Command ID of the popup menu's "Current" entry. If this value is non-zero, the popup menu will contain the entry "Current". If the user selects that entry, the resulting command ID will be *ExtraID*. If *ExtraID* is zero, the popup menu will not contain the entry "Current".

EditID

[*int*] Command ID of the popup menu's "Edit..." entry. If this value is non-zero, the popup menu will contain the entry "Edit...". If the user selects that entry, the resulting command ID will be *EditID*. If *EditID* is zero, the popup menu will not contain the entry "Edit...".

Contents Return Value

None.

Contents Comment

The user selection will be passed to the parent window using a WM_COMMAND message and a command ID based on *BasicID*, *ExtraID* and *EditID*. The user can cancel the menu by pressing the ESC or the ALT key or by clicking outside the menu. In this case, no command ID will be passed to the parent

handle, i.e. one may not rely on always receiving a command ID after calling this procedure!

Related Topics:

TosoDialogSelectLine

TosoDialogSelectMultiLine

TosoDialogSelectSystem

TosoDialogSelectHatch

TosoDialogSelectLayer

TosoDialogUpdateGuide (Dialog Window)

Contents **Syntax**

```
void TosoDialogUpdateGuide( int GuideMode,
                            LPSTR GuideText );
```

Displays the given texts in the guide window (if it is visible at the moment).

Contents **Parameters**

GuideMode

[*int*] This value is a bitwise OR combination of some predefined flags that define frequently used texts, plus an optional flag that allows the use of a module-defined additional text. Available flags are:

GUIDE_SINGLE_OBJ
: Standard texts for single object identification.

GUIDE_MULTI_OBJ
: Standard texts for multiple object choosing. Should be combined with GUIDE_SINGLE_OBJ.

GUIDE_SINGLE_POINT
: Standard texts for single point identification.

GUIDE_MULTI_POINT
: Standard texts for multiple point choosing. Should be combined with GUIDE_SINGLE_POINT.

GUIDE_PARAMETER
: Standard text for commands that allow parameter editing.

GUIDE_ARC_DIRECTION
: Standard text for commands that make use of the arc direction.

GUIDE_ARC_MODE
: Standard text for commands that make use of the arc mode.

GUIDE_COORDINATES
: Standard text for point entries that make use of direct coordinate entry.

GUIDE_SNAP
: Standard text for point entries that make use of snap functions.

GUIDE_GRID
: Standard text for point entries that make use of grid functions.

GUIDE_DUPLICATE
: Standard text for point entries that make use of the duplicate function. Should *not* be combined with GUIDE_MULTILINE.

GUIDE_MULTILINE
: Standard text for point entries that make use of the multiline function. Should *not* be combined with GUIDE_DUPLICATE.

GUIDE_ESC
: Standard text for commands that require more than one point and that allow to re-enter previously entered points.

GUIDE_RBUTTON1

Standard text for commands any point entries and that can be canceled by pressing the right mouse button. Should *not* be combined with GUIDE_RBUTTON2.

GUIDE_RBUTTON2
Standard text for commands that require a variable number of point entries and that is finished by pressing the right mouse button. Should *not* be combined with GUIDE_RBUTTON1.

GUIDE_ENTRY
Standard text displayed during coordinate entry. Should *not* be combined with any flag except with either GUIDE_DUPLICATE or GUIDE_MULTILINE.

GUIDE_DEFAULT
If this flag is set, all currently "useful" flags of those described above will be set.

GUIDE_EXTERNAL
If this flag is set, and GuideText points to a valid text, that text will be displayed in the guide window. If additional texts are to be display since other flags are set, this external text will be displayed at the end of the list.

GuideText
[*LPSTR*] Address of a text to be displayed in the guide window. The text can be a multi-paragraph, multi-line text. Each line is to be ended by a '\n' character, each paragraph by a '\0' character. The complete text must be ended by a double '\0'. Each new paragraph will start with a red square. A valid guide text would be:

```
"Move in 45° steps: Press SHIFT\0Define steps: SHIFT+ESC\0\0"
```

The text whose address is passed in *GuideText* must remain valid until TosoDialogUpdateGuide is called with another value of *GuideText*! To avoid problems, call this procedure with a *GuideMode* of GUIDE_DEFAULT and *GuideText* set to NULL as soon as the custon display is no longer required.

Return Value

None.

Comment

This procedure will usually be called inside the TosoInputPointInitProc called during command processing. Each time a new point entry is started, the content of the guide window is reset, so the module should call TosoDialogUpdateGuide at *each* call of the TosoInputPointInitProc.

In most cases, its best to combine GUIDE_DEFAULT to get the default messages, GUIDE_EXTERNAL to add a module-defined text and maybe some other mode to explicitely activate messages that would normally not be active.

Related Topics:

TosoDialogShowProgress

TosoDialogShowProgress (Dialog Window)

Syntax

```
BOOL TosoDialogShowProgress( const LPSTR Title,
                             const LPSTR Text,
                             BOOL HasPercentBar );
```

Displays a non-modal progress indication dialog box and disables the application's window.

Parameters

Title

[*const LPSTR*] Title of the dialog window to be displayed. Be careful that the given title fits into the dialog window's caption.

Text

[*const LPSTR*] Text to be displayed in the upper line of the progress indicator. The text may contain one linefeed. This text should be used to describe the current action, like "Importing from file\nXYZ". When using filenames here, be sure to display only the file's base name and *not* the complete path. You can use TosoFileSplitName to retrieve a file's base name.

HasPercentBar

[*BOOL*] If *HasPercentBar* is TRUE, the progress indicator will be equipped with a percent bar.

Return Value

Returns TRUE if successful, or FALSE if unable to display the progress indicator (due to memory limitations).

Comment

The middle and lower line of the progress indicator will be initialized with an empty text, the percent bar (if existing) will be initialized with 0%.

Related Topics:

TosoDialogUpdateGuide

TosoDialogUpdateProgress

TosoDialogHideProgress

TosoDialogIsCanceled

TosoDialogUpdateProgress (Dialog Window)

Contents

Syntax

```
void TosoDialogUpdateProgress( const LPSTR Text1,
                               const LPSTR Text2,
                               long Done,
                               long Total );
```

Initialises the text controls and, if existing, the progress indicator of a non-modal progress indication box.

Contents

Parameters

Text1

[*const LPSTR*] Text to be displayed in the middle line of the progress indicator. This text should be used to display a numeric indicator like "Object 25 of 100" or at least "Object 25". If *Text1* is NULL, this parameter is ignored, i.e. the current content of the middle line will remain unchanged.

Text2

[*const LPSTR*] Text to be displayed in the lower line of the progress indicator. This text should be used to display a size indicator like "30 KBytes". If *Text2* is NULL, this parameter is ignored, i.e. the current content of the lower line will remain unchanged.

Done

[*long*] Number of calculation steps already executed. This value is used to calculate the percent bar's filling level. If *Done* is less than zero, this parameter is ignored, the percent bar may change if *Total* is valid.

Total

[*long*] Number of total calculation steps. This value is used to calculate the percent bar's filling level. If *Total* is 0, the percent bar will be empty. If *Total* is less than zero, this parameter is ignored, the percent bar may change if *Done* is valid.

Contents

Return Value

None.

Contents

Comment

The progress indicator management uses timer functions to achieve a "smooth" movement of the percent bar. In average, the content of the progress indicator is updated every 1/10 second. Due to this timer management, feel free to call this procedure *each* time *any* value changes, there will be no markable performance loss caused by permanent dialog window updates!
At the time the progress indicator's content is updated, the application's message queue is checked for a message indicating that the user either pressed the "Cancel" button of the progress indicator or pressed the ESC key to cancel. Use TosoDialogIsCanceled to determine whether this has occured.

Contents

Related Topics:

TosoDialogHideProgress (Dialog Window)

Syntax

```
void TosoDialogHideProgress( void );
```

Hides the non-modal progress indication dialog box and enables the application's window.

Parameters

None.

Return Value

None.

Comment

None.

Related Topics:

TosoDialogUpdateGuide

TosoDialogShowProgress

TosoDialogUpdateProgress

TosoDialogIsCanceled

TosoDialogIsCanceled (Dialog Window)

Syntax

```
BOOL TosoDialogIsCanceled( void );
```

Checks whether the user has pressed the cancel button in a previously actived progress indicator.

Parameters

None.

Return Value

Returns TRUE if the "Cancel" button of a currently open or previously opened progress indicator has been pressed, else FALSE.

Comment

This procedure will work even after the progress indicator has been closed by means of TosoDialogHideProgress. It will return the correct value until TosoDialogShowProgress is called again.

Related Topics:

TosoDialogUpdateGuide

TosoDialogShowProgress

TosoDialogUpdateProgress

TosoDialogHideProgress

TosoPageGetDef (Element Handling)

Syntax

```
BOOL TosoPageGetDef( int DrawingNum,
                     PAGEDEF* Data );
```

Retrieves the current page attributes of the drawing.

Parameters

DrawingNum

[*int*] Zero-based index of the drawing whose page definition shall be returned. Valid range: 0 <= Value < TVG_DRAWING_MAX.

Data

[*PAGEDEF**] Address of a buffer to retrieve the desired page definition.

Return Value

Returns TRUE if the page definition is defined, else FALSE.

Comment

None.

Related Topics:

- TosoPageSetDef
- TosoHatchGetCurrentDef
- TosoHatchGetOrigin
- TosoHatchGetDef
- TosoHatchSetDef
- TosoHatchGetActive
- TosoHatchSetActive

Contents

TosoPageSetDef (Element Handling)

Syntax

```
BOOL TosoPageSetDef( int DrawingNum,
                     const PAGEDEF* Data );
```

Parameters

DrawingNum

[*int*] Zero-based index of the drawing whose page definition shall be set. Valid range: 0 <= Value < TVG_DRAWING_MAX.

Data

[*const PAGEDEF**] Address of a buffer containing the new page definition.

Sets the current page attributes of the drawing.

Return Value

Returns TRUE if the page definition has been set successfully, else FALSE.

Comment

None.

Related Topics:

- TosoPageGetDef
- TosoHatchGetCurrentDef
- TosoHatchGetOrigin
- TosoHatchGetDef
- TosoHatchSetDef
- TosoHatchGetActive
- TosoHatchSetActive

Contents

TosoHatchGetCurrentDef (Element Handling)

Syntax

```
BOOL TosoHatchGetCurrentDef( HATCHDEF* Data );
```

Copies the currently active hatching type definition into the buffer pointed to by *Data*.

Parameters

Data

[*HATCHDEF**] Address of a buffer to retrieve the desired hatch type definition.

Return Value

Returns TRUE if successful, else FALSE.

Comment

This procedure could also be implemented as the following code segment:

```
return( TosoHatchGetDef( TosoDrawingGetActive(),
                         TosoHatchGetActive( TosoDrawingGetActive() ),
                         Data );
```

Anyway, the implementation of this procedure is faster and more compact.

Related Topics:

TosoPageGetDef

TosoPageSetDef

TosoHatchGetOrigin

TosoHatchGetDef

TosoHatchSetDef

TosoHatchGetActive

TosoHatchGetOrigin (Element Handling)

Syntax

```
BOOL TosoHatchGetOrigin( int DrawingNum,
                         DPOINT* Data );
```

Copies the current hatching origin of the given drawing into the buffer pointed to by *Data*.

Parameters

DrawingNum

[*int*] Zero-based index of the drawing whose hatching origin shall be returned. Valid range: 0 <= Value < TVG_DRAWING_MAX.

Data

[*DPOINT**] Address of a buffer to retrieve the desired hatching origin.

Return Value

Returns TRUE if successful, else FALSE.

Comment

None.

Contents

Related Topics:

- TosoPageGetDef
- TosoPageSetDef
- TosoHatchGetCurrentDef
- TosoHatchGetDef
- TosoHatchSetDef
- TosoHatchGetActive
- TosoHatchSetActive

Contents

TosoHatchGetDef (Element Handling)

Syntax

```
BOOL TosoHatchGetDef( int DrawingNum,
                      int Index,
                      HATCHDEF* Data );
```

Copies the desired hatching type definition into the buffer pointed to by *Data*, if defined.

Parameters

DrawingNum

[*int*] Zero-based index of the drawing whose hatch type definition shall be returned. Valid range: 0 <= Value < TVG_DRAWING_MAX.

Index

[*int*] Zero-based index of the hatch type definition that shall be returned. Valid range: 0 <= Value <= TVG_HATCH_MAX.

Data

[*HATCHDEF**] Address of a buffer to retrieve the desired hatch type definition.

Return Value

Returns TRUE if the desired hatch type is defined, else FALSE.

Comment

None.

Related Topics:

TosoPageGetDef

TosoPageSetDef

TosoHatchGetCurrentDef

TosoHatchGetOrigin

TosoHatchSetDef

TosoHatchSetDef (Element Handling)

Syntax

```
BOOL TosoHatchSetDef( int DrawingNum,
                      int Index,
                      const HATCHDEF* Data );
```

Replaces the desired hatch type definition with the new definition pointed to by *Data*.

Parameters

DrawingNum

[*int*] Zero-based index of the drawing whose hatch type definition shall be set. Valid range: 0 <= Value < TVG_DRAWING_MAX.

Index

[*int*] Zero-based index of the hatch type definition that shall be set. Valid range: 0 <= Value <= TVG_HATCH_MAX.

Data

[*const HATCHDEF**] Address of a buffer containing the new hatch type definition.

Return Value

Returns TRUE if the desired hatch type was set successfully, else FALSE.

Comment

In order to delete a hatch type definition, replace it with a hatching type where *Data->HatchName* is set to "".

Contents

Related Topics:

- TosoPageGetDef
- TosoPageSetDef
- TosoHatchGetCurrentDef
- TosoHatchGetOrigin
- TosoHatchGetDef

TosoHatchGetActive (Element Handling)

Contents Syntax

```
int TosoHatchGetActive( int DrawingNum );
```

Retrieves the currently active hatch type of the given drawing.

Contents Parameters

DrawingNum

[*int*] Zero-based index of the drawing whose hatch type shall be returned. Valid range: 0 <= Value < TVG_DRAWING_MAX.

Contents Return Value

Returns the index of the currently active hatch type.

Contents Comment

None.

Contents

Related Topics:

Contents TosoPageGetDef

Contents TosoPageSetDef

Contents TosoHatchGetCurrentDef

Contents TosoHatchGetOrigin

Contents TosoHatchGetDef

Contents TosoHatchSetDef

Contents TosoHatchSetActive

Contents TosoMultiLineGetDef

TosoHatchSetActive (Element Handling)

Syntax

```
BOOL TosoHatchSetActive( int DrawingNum,
                         int Index );
```

Sets a new active hatch type in the given drawing.

Parameters

DrawingNum

[*int*] Zero-based index of the drawing whose hatch type shall be set. Valid range: 0 <= Value < TVG_DRAWING_MAX.

Index

[*int*] Zero-based index of the hatch type that shall be set active. Valid range: 0 <= Value <= TVG_HATCH_MAX.

Return Value

Return TRUE if successful, else FALSE.

Comment

None.

Related Topics:

TosoPageGetDef

TosoPageSetDef

TosoHatchGetCurrentDef

TosoHatchGetOrigin

TosoHatchGetDef

TosoHatchSetDef

TosoMultiLineGetDef (Element Handling)

Syntax

```
BOOL TosoMultiLineGetDef( int DrawingNum,
                          int Index,
                          MULTILINEDEF* Data );
```

Copies the desired line sequence definition into the buffer pointed to by *Data*, if defined.

Parameters

DrawingNum

[*int*] Zero-based index of the drawing whose line sequence definition shall be returned. Valid range: 0 <= Value < TVG_DRAWING_MAX.

Index

[*int*] Zero-based index of the line sequence definition that shall be returned. Valid range: 0 <= Value <= TVG_MULTILINE_MAX.

Data

[*MULTILINEDEF**] Address of a buffer to retrieve the desired line sequence definition.

Return Value

Returns TRUE if the desired line sequence is defined, else FALSE.

Comment

None.

Related Topics:

TosoPageGetDef

TosoPageSetDef

TosoHatchGetCurrentDef

TosoHatchGetOrigin

TosoHatchGetDef

TosoMultiLineSetDef (Element Handling)

Syntax

```
BOOL TosoMultiLineSetDef( int DrawingNum,
                          int Index,
                          const MULTILINE* Data );
```

Replaces the desired line sequence definition with the new definition pointed to by *Data*.

Parameters

DrawingNum

[*int*] Zero-based index of the drawing whose line sequence definition shall be set. Valid range: 0 <= Value < TVG_DRAWING_MAX.

Index

[*int*] Zero-based index of the line sequence definition that shall be set. Valid range: 0 <= Value <= TVG_MULTILINE_MAX.

Data

[*const MULTILINEDEF**] Address of a buffer containing the new line sequence definition.

Return Value

Returns TRUE if the desired line sequence was set successfully, else FALSE.

Comment

In order to delete a line sequence definition, replace it with a line sequence where *Data->MultiLineName* is set to "".

Contents

Related Topics:

- TosoPageGetDef
- TosoPageSetDef
- TosoHatchGetCurrentDef
- TosoHatchGetOrigin
- TosoHatchGetDef

TosoSystemGetCurrentDef (Element Handling)

Syntax

```
BOOL TosoSystemGetCurrentDef( SYSTEMDEF* Data );
```

Copies the currently active coordinate system definition into the buffer pointed to by *Data*.

Parameters

Data

[*SYSTEMDEF**] Address of a buffer to retrieve the desired coordinate system definition.

Return Value

Returns TRUE if successful, else FALSE.

Comment

This procedure could also be implemented as the following code segment:

```
return( TosoSystemGetDef( TosoDrawingGetActive(),
                          TosoSystemGetActive( TosoDrawingGetActive(),
                          TosoWindowGetActive( TosoDrawingGetActive() ),
                          Data ) );
```

Anyway, the implementation of this procedure is faster and more compact.

Related Topics:

TosoPageGetDef

TosoPageSetDef

TosoHatchGetCurrentDef

TosoHatchGetOrigin

TosoHatchGetDef

TosoHatchSetDef

TosoSystemGetDef (Element Handling)

Syntax

```
BOOL TosoSystemGetDef( int DrawingNum,
                       int Index,
                       SYSTEMDEF* Data );
```

Copies the desired coordinate system definition into the buffer pointed to by *Data*, if defined.

Parameters

DrawingNum

[*int*] Zero-based index of the drawing whose coordinate system definition shall be returned. Valid range: 0 <= Value < TVG_DRAWING_MAX.

Index

[*int*] Zero-based index of the coordinate system definition that shall be returned. Valid range: 0 <= Value <= TVG_SYSTEM_MAX.

Data

[*SYSTEMDEF**] Address of a buffer to retrieve the desired coordinate system definition.

Return Value

Returns TRUE if the coordinate system is defined, else FALSE.

Comment

None.

Related Topics:

TosoPageGetDef

TosoPageSetDef

TosoHatchGetCurrentDef

TosoHatchGetOrigin

TosoHatchGetDef

TosoSystemSetDef (Element Handling)

Syntax

```
BOOL TosoSystemSetDef( int DrawingNum,
                       int Index,
                       const SYSTEMDEF* Data );
```

Replaces the desired coordinate system definition with the new definition pointed to by *Data*.

Parameters

DrawingNum

[*int*] Zero-based index of the drawing whose coordinate system definition shall be set. Valid range: 0 <= Value < TVG_DRAWING_MAX.

Index

[*int*] Zero-based index of the coordinate system definition that shall be set. Valid range: 0 <= Value <= TVG_SYSTEM_MAX.

Data

[*const SYSTEMDEF**] Address of a buffer containing the new coordinate system definition.

Return Value

Returns TRUE if the desired coordinate system definition was set successfully, else FALSE.

Comment

In order to delete a coordinate system definition, replace it with a coordinate system definition where *Data->SystemName* is set to "".

Contents

Related Topics:

TosoPageGetDef

TosoPageSetDef

TosoHatchGetCurrentDef

TosoHatchGetOrigin

TosoHatchGetDef

TosoSystemGetActive (Element Handling)

Contents Syntax

```
int TosoSystemGetActive( int DrawingNum,
                         int WindowNum );
```

Retrieves the currently coordinate system of the given drawing window.

Contents Parameters

DrawingNum

[*int*] Zero-based index of the drawing whose active coordinate system shall be returned. Valid range: 0 <= Value < TVG_DRAWING_MAX.

WindowNum

[*int*] Zero-based index of the window whose active coordinate system shall be returned. The windows are numbered in reading direction, i.e. left to right and top to bottom. Valid range: 0 <= Value < TVG_WINDOW_MAX (standard windows) or Value = TVG_WINDOW_VIEW (view window).

Contents Return Value

Returns the index of the currently active coordinate system.

Contents Comment

None.

Contents

Related Topics:

Contents TosoPageGetDef

Contents TosoPageSetDef

Contents TosoHatchGetCurrentDef

Contents TosoHatchGetOrigin

Contents TosoHatchGetDef

Contents TosoHatchSetDef

TosoSystemSetActive (Element Handling)

Syntax

```
BOOL TosoSystemSetActive( int DrawingNum,
                          int WindowNum,
                          int Index );
```

Sets a new active coordinate system in the given drawing window.

Parameters

DrawingNum

[*int*] Zero-based index of the drawing whose active coordinate system shall be set. Valid range: 0 <= Value < TVG_DRAWING_MAX.

WindowNum

[*int*] Zero-based index of the window whose active coordinate system shall be set. The windows are numbered in reading direction, i.e. left to right and top to bottom. Valid range: 0 <= Value < TVG_WINDOW_MAX (standard windows) or Value = TVG_WINDOW_VIEW (view window).

Index

[*int*] Zero-based index of the coordinate system that shall be set active. Valid range: 0 <= Value <= TVG_SYSTEM_MAX.

Return Value

Return TRUE if successful, else FALSE.

Comment

None.

Related Topics:

TosoPageGetDef

TosoPageSetDef

TosoHatchGetCurrentDef

TosoHatchGetOrigin

TosoHatchGetDef

TosoPenGetCurrentDef (Element Handling)

Syntax

```
BOOL TosoPenGetCurrentDef( PENDEF* Data );
```

Copies the currently active pen definition definition into the buffer pointed to by *Data*.

Parameters

Data

[*PENDEF**] Address of a buffer to retrieve the desired pen definition.

Return Value

Returns TRUE if successful, else FALSE.

Comment

This procedure could also be implemented as the following code segment:

```
return( TosoPenGetDef( TosoDrawingGetActive(),
                       TosoPenGetActive( TosoDrawingGetActive() ),
                       Data );
```

Anyway, the implementation of this procedure is faster and more compact.

Related Topics:

TosoPageGetDef

TosoPageSetDef

TosoHatchGetCurrentDef

TosoHatchGetOrigin

TosoHatchGetDef

TosoHatchSetDef

TosoPenGetDef (Element Handling)

Syntax

```
BOOL TosoPenGetDef( int DrawingNum,
                    int Index,
                    PENDEF* Data );
```

Copies the desired pen definition into the buffer pointed to by *Data*, if defined.

Parameters

DrawingNum

[*int*] Zero-based index of the drawing whose pen definition shall be returned. Valid range: 0 <= Value < TVG_DRAWING_MAX.

Index

[*int*] Zero-based index of the pen definition that shall be returned. Valid range: 0 <= Value <= TVG_PEN_MAX.

Data

[*PENDEF**] Address of a buffer to retrieve the desired pen definition.

Return Value

Returns TRUE if the desired pen is defined, else FALSE.

Comment

None.

Related Topics:

TosoPageGetDef

TosoPageSetDef

TosoHatchGetCurrentDef

TosoHatchGetOrigin

TosoHatchGetDef

TosoPenSetDef (Element Handling)

Syntax

```
BOOL TosoPenSetDef( int DrawingNum,
                    int Index,
                    const PENDEF* Data );
```

Replaces the desired pen definition with the new definition pointed to by *Data*.

Parameters

DrawingNum

[*int*] Zero-based index of the drawing whose pen definition shall be set. Valid range: 0 <= Value < TVG_DRAWING_MAX.

Index

[*int*] Zero-based index of the pen definition that shall be set. Valid range: 0 <= Value <= TVG_PEN_MAX.

Data

[*const PENDEF**] Address of a buffer containing the new pen definition.

Return Value

Returns TRUE if the desired pen definition was set successfully, else FALSE.

Comment

In order to delete a pen definition, replace it with a pen definition where *Data->PenName* is set to "".

Related Topics:

TosoPageGetDef

TosoPageSetDef

TosoHatchGetCurrentDef

TosoHatchGetOrigin

TosoHatchGetDef

TosoPenGetActive (Element Handling)

Syntax

```
int TosoPenGetActive( int DrawingNum );
```

Retrieves the currently active pen of the given drawing.

Parameters

DrawingNum

[*int*] Zero-based index of the drawing whose active pen shall be returned. Valid range: 0 <= Value < TVG_DRAWING_MAX.

Return Value

Returns the index of the currently active pen.

Comment

None.

Related Topics:

TosoPageGetDef

TosoPageSetDef

TosoHatchGetCurrentDef

TosoHatchGetOrigin

TosoHatchGetDef

TosoHatchSetDef

TosoHatchGetActive

TosoHatchSetActive

TosoPenSetActive (Element Handling)

Contents Syntax

```
BOOL TosoPenSetActive( int DrawingNum,
                       int Index );
```

Sets a new active pen in the given drawing.

Contents Parameters

DrawingNum

[*int*] Zero-based index of the drawing whose active pen shall be set. Valid range: 0 <= Value < TVG_DRAWING_MAX.

Index

[*int*] Zero-based index of the pen that shall be set active. Valid range: 0 <= Value <= TVG_PEN_MAX.

Contents Return Value

Return TRUE if successful, else FALSE.

Contents Comment

None.

Contents

Related Topics:

Contents TosoPageGetDef

Contents TosoPageSetDef

Contents TosoHatchGetCurrentDef

Contents TosoHatchGetOrigin

Contents TosoHatchGetDef

Contents TosoHatchSetDef

TosoLineGetDef (Element Handling)

Syntax

```
BOOL TosoLineGetDef( int DrawingNum,
                     int Index,
                     LINEDEF* Data );
```

Copies the desired line pattern definition into the buffer pointed to by *Data*, if defined.

Parameters

DrawingNum

[*int*] Zero-based index of the drawing whose line pattern definition shall be returned. Valid range: 0 <= Value < TVG_DRAWING_MAX.

Index

[*int*] Zero-based index of the line pattern definition that shall be returned. Valid range: 0 <= Value <= TVG_LINE_MAX.

Data

[*LINEDEF**] Address of a buffer to retrieve the desired line pattern definition.

Return Value

Returns TRUE if the desired line pattern is defined, else FALSE.

Comment

None.

Related Topics:

TosoPageGetDef

TosoPageSetDef

TosoHatchGetCurrentDef

TosoHatchGetOrigin

TosoHatchGetDef

TosoLineSetDef (Element Handling)

Syntax

```
BOOL TosoLineSetDef( int DrawingNum,
                     int Index,
                     const LINEDEF* Data );
```

Replaces the desired line pattern definition with the new definition pointed to by *Data*.

Parameters

DrawingNum

[*int*] Zero-based index of the drawing whose line pattern definition shall be set. Valid range: 0 <= Value < TVG_DRAWING_MAX.

Index

[*int*] Zero-based index of the line pattern definition that shall be set. Valid range: 0 <= Value <= TVG_LINE_MAX.

Data

[*const LINEDEF**] Address of a buffer containing the new line pattern definition.

Return Value

Returns TRUE if the desired line pattern definition was set successfully, else FALSE.

Comment

In order to delete a line pattern definition, replace it with a line pattern definition where *Data->LineName* is set to "".

Contents

Related Topics:

Contents TosoPageGetDef

Contents TosoPageSetDef

Contents TosoHatchGetCurrentDef

Contents TosoHatchGetOrigin

Contents TosoHatchGetDef

TosoLayerGetCurrentDef (Element Handling)

Syntax

```
BOOL TosoPenGetCurrentDef( LAYERDEF* Data );
```

Copies the currently active layer definition definition into the buffer pointed to by *Data*.

Parameters

Data

[*LAYERDEF**] Address of a buffer to retrieve the desired layer definition.

Return Value

Returns TRUE if successful, else FALSE.

Comment

This procedure could also be implemented as the following code segment:

```
return( TosoLayerGetDef( TosoDrawingGetActive(),
                         TosoLayerGetActive( TosoDrawingGetActive() ),
                         Data ) );
```

Anyway, the implementation of this procedure is faster and more compact.

Related Topics:

- TosoPageGetDef
- TosoPageSetDef
- TosoHatchGetCurrentDef
- TosoHatchGetOrigin
- TosoHatchGetDef
- TosoHatchSetDef

TosoLayerGetDef (Element Handling)

Syntax

```
BOOL TosoLayerGetDef( int DrawingNum,
                      int Index,
                      LAYERDEF* Data );
```

Copies the desired layer definition into the buffer pointed to by *Data*, if defined.

Parameters

DrawingNum

[*int*] Zero-based index of the drawing whose layer definition shall be returned. Valid range: 0 <= Value < TVG_DRAWING_MAX.

Index

[*int*] Zero-based index of the layer definition that shall be returned. Valid range: 0 <= Value <= TVG_LAYER_MAX.

Data

[*LAYERDEF**] Address of a buffer to retrieve the desired layer definition.

Return Value

Returns TRUE if the desired layer is defined, else FALSE.

Comment

None.

Related Topics:

TosoPageGetDef

TosoPageSetDef

TosoHatchGetCurrentDef

TosoHatchGetOrigin

TosoHatchGetDef

TosoLayerSetDef (Element Handling)

Syntax

```
BOOL TosoLayerSetDef( int DrawingNum,
                      int Index,
                      const LAYERDEF* Data );
```

Replaces the desired layer definition with the new definition pointed to by *Data*.

Parameters

DrawingNum

[*int*] Zero-based index of the drawing whose layer definition shall be set. Valid range: 0 <= Value < TVG_DRAWING_MAX.

Index

[*int*] Zero-based index of the layer definition that shall be set. Valid range: 0 <= Value <= TVG_LAYER_MAX.

Data

[*const LAYERDEF**] Address of a buffer containing the new layer definition.

Return Value

Returns TRUE if the desired layer definition was set successfully, else FALSE.

Comment

In order to delete a layer definition, replace it with a layer definition where *Data->LayerName* is set to "".

Contents

Related Topics:

Contents TosoPageGetDef

Contents TosoPageSetDef

Contents TosoHatchGetCurrentDef

Contents TosoHatchGetOrigin

Contents TosoHatchGetDef

TosoLayerGetActive (Element Handling)

Contents **Syntax**

```
int TosoLayerGetActive( int DrawingNum );
```

Retrieves the currently active layer of the given drawing window.

Contents **Parameters**

DrawingNum

[*int*] Zero-based index of the drawing whose active layer shall be returned. Valid range: 0 <= Value < TVG_DRAWING_MAX.

Contents **Return Value**

Returns the index of the currently active layer.

Contents **Comment**

None.

Contents

Related Topics:

Contents TosoPageGetDef

Contents TosoPageSetDef

Contents TosoHatchGetCurrentDef

Contents TosoHatchGetOrigin

Contents TosoHatchGetDef

Contents TosoHatchSetDef

Contents TosoHatchGetActive

Contents TosoHatchSetActive

TosoLayerSetActive (Element Handling)

Syntax

```
BOOL TosoLayerSetActive( int DrawingNum,
                         int Index );
```

Sets a new active layer in the given drawing window.

Parameters

DrawingNum

[*int*] Zero-based index of the drawing whose active layer shall be set. Valid range: 0 <= Value < TVG_DRAWING_MAX.

Index

[*int*] Zero-based index of the layer that shall be set active. Valid range: 0 <= Value <= TVG_LAYER_MAX.

Return Value

Return TRUE if successful, else FALSE.

Comment

None.

Related Topics:

TosoPageGetDef

TosoPageSetDef

TosoHatchGetCurrentDef

TosoHatchGetOrigin

TosoHatchGetDef

TosoHatchSetDef

TosoCreationStart (Unit Creation)

Syntax

```
BOOL TosoCreationStart( void );
```

Initialises the unit creation buffers and data. This function must be called before *any* access to any of the other unit creation functions. As all buffers are only available once, the creation process should last as short as possible.

Parameters

None.

Return Value

Returns TRUE if creation may start. If FALSE is returned, this indicates that either creation has already been started already or that there is not enough memory to allocate the required buffers.

Comment

None.

Related Topics:

TosoCreationEnd

TosoUndoInitProcess

TosoUndoCancelProcess

TosoUndoFinishProcess

TosoUndoUpdateLinks

TosoUndoSetPrevious

TosoCreationEnd (Unit Creation)

Syntax

```
void TosoCreationEnd( void );
```

Ends a unit creation or modification previously started by calling TosoCreationStart. After closing the interface, no unit creation commands may be called unless the creation is started again.

Parameters

None.

Return Value

None.

Comment

None.

Related Topics:

- TosoCreationStart
- TosoUndoInitProcess
- TosoUndoCancelProcess
- TosoUndoFinishProcess
- TosoUndoUpdateLinks
- TosoUndoSetPrevious

TosoUndoInitProcess (Unit Creation)

Contents

Syntax

```
void TosoUndoInitProcess( void );
```

Starts a new undo level. All objects, instances and blocks created after this command will be assigned to that undo level. These objects will not be valid until TosoUndoFinishProcess is called.

Contents

Parameters

None.

Contents

Return Value

None.

Contents

Comment

The number of concurrent undo levels depends on user settings in the serving application. Usually, each command execution should use exactly *one* undo level.

Contents

Related Topics:

Contents TosoCreationStart
Contents TosoCreationEnd
Contents TosoUndoCancelProcess
Contents TosoUndoFinishProcess
Contents TosoUndoUpdateLinks
Contents TosoUndoSetPrevious

TosoUndoCancelProcess (Unit Creation)

Syntax

```
void TosoUndoCancelProcess( void );
```

Cancels a previously started undo level. All objects, instances and blocks created since the last TosoUndoInitProcess call are deleted, objects deleted since then are undeleted.

Parameters

None.

Return Value

None.

Comment

None.

Related Topics:

- TosoCreationStart
- TosoCreationEnd
- TosoUndoInitProcess
- TosoUndoFinishProcess
- TosoUndoUpdateLinks
- TosoUndoSetPrevious

TosoUndoFinishProcess (Unit Creation)

Syntax

```
void TosoUndoFinishProcess( void );
```

Validates a previously started undo level. All objects, instances and blocks created since the last TosoUndoInitProcess call now valid, objects deleted since then are permanently deleted.

Parameters

None.

Return Value

None.

Comment

None.

Related Topics:

TosoCreationStart

TosoCreationEnd

TosoUndoInitProcess

TosoUndoCancelProcess

TosoUndoUpdateLinks

TosoUndoSetPrevious

TosoUndoUpdateLinks (Unit Creation)

Syntax

```
void TosoUndoUpdateLinks( void );
```

Recalculates the internal links between instances, blocks and attributes.

Parameters

None.

Return Value

None.

Comment

This procedure should be called after TosoUndoFinishProcess if any changes in blocks or instances occured.

Related Topics:

- TosoCreationStart
- TosoCreationEnd
- TosoUndoInitProcess
- TosoUndoCancelProcess
- TosoUndoFinishProcess
- TosoUndoSetPrevious

TosoUndoSetPrevious (Unit Creation)

Syntax

```
void TosoUndoSetPrevious( void );
```

Sets the FLAG_PREVIOUS flag in all objects created in the latest undo level, allowing the user to choose all these objects by simply pressing the F11 key.

Parameters

None.

Return Value

None.

Comment

This procedure should be called after TosoUndoFinishProcess if the module created a large number of separate objects within one command execution.

Related Topics:

TosoCreationStart

TosoCreationEnd

TosoUndoInitProcess

TosoUndoCancelProcess

TosoUndoFinishProcess

TosoUndoUpdateLinks

TosoObjectOpen (Unit Creation)

Contents Syntax

```
UNIT_OBJECT_PTR TosoObjectOpen( int Type );
```

Opens a temporary object for data block input. Once opened, the temporary object can be filled with data blocks and subsequently be terminated and appended to a memory list or to a block.

Contents Parameters

Type

[*int*] Type of object to be created. Allowed values are:

OBJ_LINE

Straight line. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_HATCH

Collection of curves, usually used for hatchings. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_CIRCLE

Circle. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_ARC

Circular arc. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_SECTOR

Circular sector. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_SEGMENT

Circular segment. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_ZIGZAG

Zigzag line. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_SPLINE

Spline curve. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_CURVE

Curve consisting of lines, Bézier curves and circular arcs. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_SURFACE

Surface consisting of lines, Bézier curves and circular arcs. May contain nested contours. For information about the internal data block structure of this object type, see the corresponding

paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_ELLIPSE
Ellipse. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_EARC
Elliptical arc. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_ESECTOR
Elliptical sector. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_ESEGMENT
Elliptical segment. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_DLINE
Straight dimension line. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_DARC
Curved dimension line. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_DDISTANCE
Dimension with straight dimension line, used for length and distance dimensioning. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_DRADIUS
Dimension with straight dimension line, used for radius dimensioning. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_DDIAMETER
Dimension with straight dimension line, used for diameter dimensioning. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_DANGLE
Dimension with curved dimension line, used for angle dimensioning. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_DARCLENGTH
Dimension with curved dimension line, used for arc-length dimensioning. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_DCOORDINATE
Dimension without dimension line, used for coordinate dimensioning. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_DAREA
Dimension without dimension line, used for area dimensioning. For information about the

internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_DPERIMETER

Dimension without dimension line, used for perimeter dimensioning. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_TEXTSTANDARD

Standard text defined by an insertion point. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_TEXTFRAME

Frame text defined by a surrounding frame. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_TEXTREFERENCE

Reference text defined by an insertion point, plus a frame and an arrow. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_COMMENT

Comment (non-printed object). For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_MARK

Marking (optionally printed object). For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_CLIPSURFACE

Clipping surface. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_BITMAPREF

Bitmap reference (optionally printed object). For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_GEOLINE

Geometry line (optionally printed object). For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_GEOCIRCLE

Geometry circle (optionally printed object). For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_GEOELLIPSE

Geometry ellipse (optionally printed object). For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

Contents **Return Value**

Address of the temporary object, if successful. This address can be used to modify the object's extended properties (located in *ReturnValue->XProperty*). Please do not modify any other value! If NULL is returned, this indicates that another temporary object is already open.

Contents **Comment**

Only one temporary object is available at a time. Anyway, other temporary entities like instance, user object, block and group can be open simultaneously. After opening the temoporary object, it must later be closed using TosoObjectClose or TosoObjectFastInsert.

Contents

Related Topics:

Contents TosoObjectGetAddress

Contents TosoObjectAddPoint

Contents TosoObjectAddConstant

Contents TosoObjectAddOrient

Contents TosoObjectAddCurve

Contents TosoObjectAddTextShort

Contents TosoObjectAddTextLong

Contents TosoObjectAddDimLine

Contents TosoObjectAddDimSmall

Contents TosoObjectAddDimLarge

Contents TosoObjectAddTextStandard

Contents TosoObjectAddTextFrame

TosoObjectGetAddress (Unit Creation)

Contents **Syntax**

```
UNIT_OBJECT_PTR TosoObjectGetAddress( void );
```

This procedure retrieves the address of the currently open temporary object. It can also be used to check whether an object is currently open.

Contents **Parameters**

None.

Contents **Return Value**

Returns the address of a previously opened temporary object. Returns NULL if the object is not open.

Contents **Comment**

None.

Contents

Related Topics:

Contents TosoObjectOpen

Contents TosoObjectAddPoint

Contents TosoObjectAddConstant

Contents TosoObjectAddOrient

Contents TosoObjectAddCurve

Contents TosoObjectAddTextShort

Contents TosoObjectAddTextLong

Contents TosoObjectAddDimLine

TosoObjectAddPoint (Unit Creation)

Contents **Syntax**

```
BOOL TosoObjectAddPoint( int Type,
                         double x,
                         double y );
```

Adds a point data block (DB_POINT_???) to a previously opened temporary object.

Contents **Parameters**

Type

[*int*] Data block type. Allowed values are:

DB_POINT_ANY
: Unspecified point type.

DB_POINT_START
: Start-point.

DB_POINT_END
: End-point.

DB_POINT_CENTER
: Center-point.

DB_POINT_RADIUS
: Radius definition point.

DB_POINT_ANGLE
: Angle definition point.

DB_POINT_VECTOR
: Vector definition point.

DB_POINT_PIVOT1
: First pivot point of a Bézier curve.

DB_POINT_PIVOT2
: Second pivot point of a Bézier curve.

DB_POINT_ARC
: End-point of a circular arc inside a curve or surface.

DB_POINT_MARK
: Marking.

x

y

[*double*] Coordinates of the point to be added in internal millimeters (relative to the page center, scale-independent). Valid range: COORD_MIN <= Value <= COORD_MAX.

Contents **Return Value**

Return TRUE if the data block was added successfully. If FALSE is returned, this indicates that the object

would be too large after adding this data block.

Comment

Even though the application does usually not check the point data block types of standard objects, whose data block sequence is fix, every point data block should have a correct type, as this type is used during snapping!

Related Topics:


TosoObjectOpen


TosoObjectGetAddress


TosoObjectAddConstant


TosoObjectAddOrient


TosoObjectAddCurve


TosoObjectAddTextShort


TosoObjectAddTextLong


TosoObjectAddDimLine


TosoObjectAddDimSmall


TosoObjectAddDimLarge


TosoObjectAddTextStandard


TosoObjectAddTextFrame


TosoObjectAddTextReference

TosoObjectAddConstant (Unit Creation)

Syntax

```
BOOL TosoObjectAddConstant( double Value );
```

Adds a constant data block (DB_ALL_CONST) to a previously opened temporary object.

Parameters

Value

[*double*] Numeric value to be stored in the constant data block. Valid range: -1e300 < Value < 1e300.

Return Value

Return TRUE if the data block was added successfully. If FALSE is returned, this indicates that the object would be too large after adding this data block.

Comment

This type of data block is usually used inside objects of type OBJ_ZIGZAG to store the indent distance, and inside objects of type OBJ_DAREA and OBJ_DPERIMETER to store the calculated dimension.

Related Topics:

TosoObjectOpen

TosoObjectGetAddress

TosoObjectAddPoint

TosoObjectAddOrient

TosoObjectAddCurve

TosoObjectAddTextShort

TosoObjectAddTextLong

TosoObjectAddDimLine

TosoObjectAddDimSmall

TosoObjectAddDimLarge

TosoObjectAddTextStandard

TosoObjectAddTextFrame

TosoObjectAddTextReference

TosoObjectAddClipSurface

TosoObjectAddBitmapRef

TosoObjectAddEnd

TosoObjectCopyDataBlocks

TosoObjectInsert

TosoObjectClose

TosoObjectFastInsert

TosoObjectAddOrient (Unit Creation)

Syntax

```
BOOL TosoObjectAddOrient( double Orient );
```

Adds an arc orientation data block (DB_ALL_ORIENT) to a previously opened temporary object.

Parameters

Orient

[*double*] Orientation of the arc. Negative values indicate clockwise orientation, non-negative values indicate counter-clockwise direction. Valid range: COORD_MIN <= Value <= COORD_MAX.

Return Value

Return TRUE if the data block was added successfully. If FALSE is returned, this indicates that the object would be too large after adding this data block.

Comment

This type of data block is usually used inside objects of type OBJ_ARC, OBJ_SECTOR, OBJ_SEGMENT etc. to determine the arc's orientation (clockwise or counter-clockwise).

Related Topics:

TosoObjectOpen

TosoObjectGetAddress

TosoObjectAddPoint

TosoObjectAddConstant

TosoObjectAddCurve

TosoObjectAddTextShort

TosoObjectAddTextLong

TosoObjectAddDimLine

TosoObjectAddDimSmall

TosoObjectAddDimLarge

TosoObjectAddTextStandard

TosoObjectAddTextFrame

TosoObjectAddTextReference

TosoObjectAddClipSurface

TosoObjectAddBitmapRef

TosoObjectAddEnd

TosoObjectCopyDataBlocks

TosoObjectInsert

TosoObjectClose

TosoObjectFastInsert

TosoObjectAddCurve (Unit Creation)

Contents **Syntax**

```
BOOL TosoObjectAddCurve( double Orient,
                         double Curve );
```

Adds an arc orientation and curvature data block (DB_ALL_CURVE) to a previously opened temporary object.

Contents **Parameters**

Orient

[*double*] Orientation of the arc. Negative values indicate clockwise orientation, non-negative values indicate counter-clockwise direction. Valid range: COORD_MIN <= Value <= COORD_MAX.

Curve

[*double*] Curvature of the arc. For a detailed description of the curvature, see the description of the object type "Curve" in the description of the TVG 4.0 file format in TVG 4.0 Documentation (TVG40.HLP). If the arc's center-point and end-points are known, the curvature can easily be calculated by means of TosoCalcCurvature. Valid range: COORD_MIN <= Value <= COORD_MAX.

Contents **Return Value**

Return TRUE if the data block was added successfully. If FALSE is returned, this indicates that the object would be too large after adding this data block.

Contents **Comment**

This type of data block is usually used inside objects of type OBJ_CURVE and OBJ_SURFACE to determine an arc's orientation (clockwise or counter-clockwise) and curvature (radius).

Contents

Related Topics:

Contents TosoObjectOpen

Contents TosoObjectGetAddress

Contents TosoObjectAddPoint

Contents TosoObjectAddConstant

Contents TosoObjectAddOrient

TosoObjectAddTextShort (Unit Creation)

Contents Syntax

```
BOOL TosoObjectAddTextShort( const LPSTR Text,
                             BOOL StaticLength );
```

Adds a short text data block (DB_ALL_TEXT) to a previously opened temporary object.

Contents Parameters

Text

[*const LPSTR*] Text to be stored in the text data block. The maximum length of *Text* is NAME_LENGTH_TEXTSHORT.

StaticLength

[*BOOL*] Indicates whether the text data block should have static length or not. If *StaticLength* is TRUE, the application will allocate memory for 250 characters, independent of the length of *Text*. This allows direct manipulation of that data block without the need to reorganise the object. If *StaticLength* is FALSE, the application will allocate only as much memory as required to store *Text*.

Contents Return Value

Return TRUE if the data block was added successfully. If FALSE is returned, this indicates that the object would be too large after adding this data block.

Contents Comment

This type of data block is usually used inside dimension objects (OBJ_D???) and inside comments (OBJ_COMMENT).

Contents

Related Topics:

Contents TosoObjectOpen

Contents TosoObjectGetAddress

Contents TosoObjectAddPoint

Contents TosoObjectAddConstant

Contents TosoObjectAddOrient

TosoObjectAddTextLong (Unit Creation)

Syntax

```
BOOL TosoObjectAddTextLong( const LPSTR Text,
                            BOOL StaticLength );
```

Adds a long text data block (DB_ALL_TEXT) to a previously opened temporary object.

Parameters

Text

[*const LPSTR*] Text to be stored in the text data block. The maximum length of *Text* is NAME_LENGTH_TEXTLONG.

StaticLength

[*BOOL*] Indicates whether the text data block should have static length or not. If *StaticLength* is TRUE, the application will allocate memory for 8000 characters, independent of the length of *Text*. This allows direct manipulation of that data block without the need to reorganise the object. If *StaticLength* is FALSE, the application will allocate only as much memory as required to store *Text*.

Return Value

Return TRUE if the data block was added successfully. If FALSE is returned, this indicates that the object would be too large after adding this data block.

Comment

This type of data block is usually used inside text objects (OBJ_TEXTSTANDARD, OBJ_TEXTFRAME and OBJ_TEXTREFERENCE).

Related Topics:

TosoObjectOpen

TosoObjectGetAddress

TosoObjectAddPoint

TosoObjectAddConstant

TosoObjectAddOrient

TosoObjectAddDimLine (Unit Creation)

Syntax

```
BOOL TosoObjectAddDimLine( const DIMLINE* Data );
```

Adds a dimension line parameter data block (DB_INFO_DIMLINE) to a previously opened temporary object.

Parameters

Data

[*const DIMLINE**] Data to be stored in the dimension line parameter data block.

Return Value

Return TRUE if the data block was added successfully. If FALSE is returned, this indicates that the object would be too large after adding this data block.

Comment

This type of data block is usually used inside dimension line objects (OBJ_DLINE and OBJ_DARC).

Related Topics:

TosoObjectOpen

TosoObjectGetAddress

TosoObjectAddPoint

TosoObjectAddConstant

TosoObjectAddOrient

TosoObjectAddCurve

TosoObjectAddTextShort

TosoObjectAddDimSmall (Unit Creation)

Syntax

```
XPROPERTY* TosoObjectAddDimSmall( const DIMSMALL* Data,
                                  BOOL UseGlobal );
```

Adds a small dimension parameter data block (DB_INFO_DIMSMALL) to a previously opened temporary object.

Parameters

Data

[*const DIMSMALL**] Data to be stored in the small dimension parameter data block. The *TextXProperty* component will be overwritten by the current default properties for that object. In case these properties are to be modified, use the address returned from this procedure to access them directly.

UseGlobal

[*BOOL*] If this value is TRUE, the elements *TextFont*, *TextSize1*, *TextSize2*, *NumAccuracy* and *System* of the structure pointed to by *Data* will not be used, instead, the current global dimension parameters will be used.

Return Value

Returns the address of the *TextXProperty* component of the created data block if the data block was added successfully. This address can be used to alter the default properties of the dimension text. If NULL is returned, this indicates that the object would be too large after adding this data block.

Comment

This type of data block is usually used inside dimension objects that do not include dimension lines (OBJ_DCOORDINATE, OBJ_DAREA and OBJ_DPERIMETER).

Related Topics:

TosoObjectOpen

TosoObjectGetAddress

TosoObjectAddPoint

TosoObjectAddConstant

TosoObjectAddDimLarge (Unit Creation)

Contents

Syntax

```
XPROPERTY* TosoObjectAddDimLarge( const DIMLARGE* Data,
                                  BOOL UseGlobal );
```

Adds a large dimension parameter data block (DB_INFO_DIMLARGE) to a previously opened temporary object.

Contents

Parameters

Data

[*const DIMLARGE**] Data to be stored in the large dimension parameter data block. The *TextXProperty* component will be overwritten by the current default properties for that object. In case these properties are to be modified, use the address returned from this procedure to access them directly.

UseGlobal

[*BOOL*] If this value is TRUE, the elements *TextFont*, *TextSize1*, *TextSize2*, *NumAccuracy* and *System* of the structure pointed to by *Data* will not be used, instead, the current global dimension parameters will be used.

Contents

Return Value

Returns the address of the *TextXProperty* component of the created data block if the data block was added successfully. This address can be used to alter the default properties of the dimension text. If NULL is returned, this indicates that the object would be too large after adding this data block.

Contents

Comment

This type of data block is usually used inside dimension objects that include dimension lines (OBJ_DDISTANCE, OBJ_DRADIUS, OBJ_DDIAMETER, OBJ_DANGLE and OBJ_DARCLENGTH).

Contents

Related Topics:

Contents TosoObjectOpen

Contents TosoObjectGetAddress

Contents TosoObjectAddPoint

Contents TosoObjectAddConstant

TosoObjectAddTextStandard (Unit Creation)

Syntax

```
XPROPERTY* TosoObjectAddTextStandard( const TEXTSTANDARD* Data );
```

Adds a standard text parameter data block (DB_INFO_TEXTSTANDARD) to a previously opened temporary object.

Parameters

Data

[*const TEXTSTANDARD**] Data to be stored in the standard text parameter data block. The *TextXProperty* component will be overwritten by the current default properties for that object. In case these properties are to be modified, use the address returned from this procedure to access them directly.

Return Value

Returns the address of the *TextXProperty* component of the created data block if the data block was added successfully. This address can be used to alter the default properties of the text. If NULL is returned, this indicates that the object would be too large after adding this data block.

Comment

This type of data block is usually used inside standard and reference texts (OBJ_TEXTSTANDARD and OBJ_TEXTREFERENCE).

Related Topics:

TosoObjectOpen

TosoObjectGetAddress

TosoObjectAddPoint

TosoObjectAddConstant

TosoObjectAddOrient

TosoObjectAddCurve

TosoObjectAddTextFrame (Unit Creation)

Syntax

```
XPROPERTY* TosoObjectAddTextFrame( const TEXTFRAME* Data );
```

Adds a frame text parameter data block (DB_INFO_TEXTFRAME) to a previously opened temporary object.

Parameters

Data

[*const TEXTFRAME**] Data to be stored in the frame text parameter data block. The *TextXProperty* component will be overwritten by the current default properties for that object. In case these properties are to be modified, use the address returned from this procedure to access them directly.

Return Value

Returns the address of the *TextXProperty* component of the created data block if the data block was added successfully. This address can be used to alter the default properties of the text. If NULL is returned, this indicates that the object would be too large after adding this data block.

Comment

This type of data block is usually used inside frame texts (OBJ_TEXTFRAME).

Related Topics:

TosoObjectOpen

TosoObjectGetAddress

TosoObjectAddPoint

TosoObjectAddConstant

TosoObjectAddOrient

TosoObjectAddCurve

TosoObjectAddTextReference (Unit Creation)

Syntax

```
BOOL TosoObjectAddTextReference( const TEXTREFERENCE* Data );
```

Parameters

Data

[*const TEXTREFERENCE**] Data to be stored in the reference text parameter data block.

Adds a reference text parameter data block (DB_INFO_TEXTREFERENCE) to a previously opened temporary object.

Return Value

Return TRUE if the data block was added successfully. If FALSE is returned, this indicates that the object would be too large after adding this data block.

Comment

This type of data block is usually used inside reference texts (OBJ_TEXTREFERENCE).

Related Topics:

- TosoObjectOpen
- TosoObjectGetAddress
- TosoObjectAddPoint
- TosoObjectAddConstant
- TosoObjectAddOrient
- TosoObjectAddCurve
- TosoObjectAddTextShort

TosoObjectAddClipSurface (Unit Creation)

Contents **Syntax**

```
XPROPERTY* TosoObjectAddClipSurface( const CLIPSURFACE* Data );
```

Adds a clipping surface parameter data block (DB_INFO_CLIPSURFACE) to a previously opened temporary object.

Contents **Parameters**

Data

[*const CLIPSURFACE**] Data to be stored in the clipping surface parameter data block. The *XProperty* component will be overwritten by the current default properties for that object. In case these properties are to be modified, use the address returned from this procedure to access them directly.

Contents **Return Value**

Returns the address of the *XProperty* component of the created data block if the data block was added successfully. This address can be used to alter the default properties of the referenced block. If NULL is returned, this indicates that the object would be too large after adding this data block.

Contents **Comment**

This type of data block is usually used inside clipping surfaces (OBJ_CLIPSURFACE).

Contents

Related Topics:

Contents TosoObjectOpen

Contents TosoObjectGetAddress

Contents TosoObjectAddPoint

Contents TosoObjectAddConstant

Contents TosoObjectAddOrient

Contents TosoObjectAddCurve

TosoObjectAddBitmapRef (Unit Creation)

Syntax

```
BOOL TosoObjectAddBitmapRef( const BITMAPREF* Data );
```

Adds a bitmap reference parameter data block (DB_INFO_BITMAPREF) to a previously opened temporary object.

Parameters

Data

[*const BITMAPREF**] Data to be stored in the bitmap reference parameter data block.

Return Value

Return TRUE if the data block was added successfully. If FALSE is returned, this indicates that the object would be too large after adding this data block.

Comment

This type of data block is usually used inside bitmap reference (OBJ_BITMAPREF).

Related Topics:

TosoObjectOpen

TosoObjectGetAddress

TosoObjectAddPoint

TosoObjectAddConstant

TosoObjectAddOrient

TosoObjectAddCurve

TosoObjectAddTextShort

TosoObjectAddEnd (Unit Creation)

Contents **Syntax**

```
BOOL TosoObjectAddEnd( void );
```

Adds an end-of-list data block (DB_END) to a previously opened temporary object and sets the internal length of the object to the correct value.

Contents **Parameters**

None.

Contents **Return Value**

Return TRUE if the data block was added successfully. If FALSE is returned, this indicates that the object would be too large after adding this data block.

Contents **Comment**

Once the temporary object has been terminated using this procedure, no further data block may be added! Nevertheless, its header can still be modified.

Contents

Related Topics:

Contents TosoObjectOpen

Contents TosoObjectGetAddress

Contents TosoObjectAddPoint

Contents TosoObjectAddConstant

Contents TosoObjectAddOrient

Contents TosoObjectAddCurve

Contents TosoObjectAddTextShort

TosoObjectCopyDataBlocks (Unit Creation)

Syntax

```
BOOL TosoObjectCopyDataBlocks( const UNIT_OBJECT_PTR* RefObj );
```

Copies all data blocks contained in a reference object (excluding the DB_END block) to a previously opened temporary object. They will appended to already existing data blocks.

Parameters

RefObj

[*const UNIT_OBJECT_PTR**] Address of a reference objects whose data blocks are to be copied.

Return Value

Return TRUE if all data block were copied and added successfully. If FALSE is returned, this indicates that the object would be too large after adding those data blocks.

Comment

None.

Related Topics:

TosoObjectOpen

TosoObjectGetAddress

TosoObjectAddPoint

TosoObjectAddConstant

TosoObjectAddOrient

TosoObjectAddCurve

TosoObjectAddTextShort

TosoObjectInsert (Unit Creation)

Syntax

```
UNIT_OBJECT_PTR TosoObjectInsert( int DrawingNum,
                                  const UNIT_PTR RefObj );
```

Inserts the temporary object to a memory list. If no block is currently open, the object is inserted to the main memory list of the drawing behind the object pointed to by *RefObj*. If *RefObj* is NULL, the object is appended at the end.

If a temporary block is currently open, the object is appended to that block. *RefObj* is ignored in this case.

Parameters

DrawingNum

[*int*] Zero-based index of the drawing into which the object shall be inserted. Valid range: 0 <= Value < TVG_DRAWING_MAX.

RefObj

[*const UNIT_PTR*] Address of a reference unit behind which the object shall be inserted. If *RefObj* is NULL, the object is appended at the end.

Return Value

Returns the address of the inserted object. Returns NULL if the insertion was not successful, normally due to insufficient memory.

Comment

In most cases, you can use TosoObjectFastInsert instead of this procedure to speed up execution and to reduce code size. See the description of that procedure for details.

Related Topics:

TosoObjectOpen

TosoObjectGetAddress

TosoObjectAddPoint

TosoObjectAddConstant

TosoObjectClose (Unit Creation)

Contents **Syntax**

```
void TosoObjectClose( void );
```

Closes a previously opened temporary object. The object may not be used afterwards until TosoObjectOpen is called again.

Contents **Parameters**

None.

Contents **Return Value**

None.

Contents **Comment**

None.

Contents

Related Topics:

Contents TosoObjectOpen

Contents TosoObjectGetAddress

Contents TosoObjectAddPoint

Contents TosoObjectAddConstant

Contents TosoObjectAddOrient

Contents TosoObjectAddCurve

Contents TosoObjectAddTextShort

Contents TosoObjectAddTextLong

TosoObjectFastInsert (Unit Creation)

Syntax

```
BOOL TosoObjectFastInsert( void );
```

This procedure terminates a currently opened temporary object and inserts it to the current drawing or to a currently open temporary block.

Parameters

None.

Return Value

Returns TRUE if the object was successfully inserted. Returns FALSE if the insertion was not successful, normally due to insufficient memory.

Comment

This procedure is a kind of makro that executes steps similar to the following code segment:

```
BOOL Result = TRUE;

TosoObjectAddEnd();
if( !TosoObjectInsert( TosoDrawingGetActive(), NULL ) )
  Result = FALSE;
TosoObjectClose();

return( Result );
```

This causes the temporary object to be terminated, initialised and then inserted to the current drawing. Afterwards, it is closed. For a detailed description, see the referenced procedure's descriptions.

If possible, you should use this command instead of calling the separate procedures to increase speed and to reduce code size.

Related Topics:

TosoObjectOpen

TosoObjectGetAddress

TosoObjectAddPoint

TosoUserOpen (Unit Creation)

Contents Syntax

```
UNIT_USER_PTR TosoUserOpen( int OwnerID,
                            int Type );
```

Opens a temporary user object for data block input. Once opened, the temporary user object can be filled with data blocks and subsequently be terminated and appended to a memory list or to a block.

Contents Parameters

Type

[*int*] Type of user object to be created. This type can freely be defined by the module.

Contents Return Value

Address of the temporary user object, if successful. This address can be used to modify the user object's extended properties (located in *ReturnValue->XProperty*). Please do not modify any other value! If NULL is returned, this indicates that another temporary user object is already open.

Contents Comment

Only one temporary user object is available at a time. Anyway, other temporary entities like object, instance, block and group can be open simultaneously. After opening the temporary user object, it must later be closed using TosoUserClose or TosoUserFastInsert.

Contents

Related Topics:

Contents TosoUserGetAddress

Contents TosoUserAddDataBlock

Contents TosoUserAddEnd

Contents TosoUserInsert

Contents TosoUserClose

Contents TosoUserFastInsert

TosoUserGetAddress (Unit Creation)

Syntax

```
UNIT_USER_PTR TosoUserGetAddress( void );
```

This procedure retrieves the address of the currently open temporary user object. It can also be used to check whether a user object is currently open.

Parameters

None.

Return Value

Returns the address of a previously opened temporary user object. Returns NULL if the user object is not open.

Comment

None.

Related Topics:

TosoUserOpen

TosoUserAddDataBlock

TosoUserAddEnd

TosoUserInsert

TosoUserClose

TosoUserFastInsert

TosoUserAddDataBlock (Unit Creation)

Contents

Syntax

```
BOOL TosoUserAddDataBlock( int Type,
                           int ElemType,
                           int ElemCount,
                           const LPVOID Data,
                           int Bytes );
```

Adds a data block to a previously opened temporary user object.

Contents

Parameters

Type

[*int*] Data block type. If *ElemType* is DB_TYPE_NATIVE, this value must be one of the following predefined values:

DB_INFO_DIMLINE
: The data block contains one DIMLINE structure.

DB_INFO_DIMLARGE
: The data block contains one DIMLARGE structure.

DB_INFO_DIMSMALL
: The data block contains one DIMSMALL structure.

DB_INFO_TEXTSTANDARD
: The data block contains one TEXTSTANDARD structure.

DB_INFO_TEXTFRAME
: The data block contains one TEXTFRAME structure.

DB_INFO_TEXTREFERENCE
: The data block contains one TEXTREFERENCE structure.

DB_INFO_CLIPSURFACE
: The data block contains one CLIPSURFACE structure.

DB_INFO_BITMAPREF
: The data block contains one BITMAPREF structure.

In all other cases, *Type* may have any value between 1000 and 29999 inclusive.

ElemType

[*int*] Type of elements that are stored in the data block. Available types are:

DB_TYPE_NATIVE
: The data block contains a predefined TVG 4.0 structure, i.e. *Type* must be one of the predefined data block types listed above.

DB_TYPE_LONG
: The data block contains a list of values of type *long*.

DB_TYPE_DOUBLE
: The data block contains a list of values of type *double*.

DB_TYPE_DPOINT

The data block contains a list of values of type *DPOINT*.

DB_TYPE_COLORREF
The data block contains a list of values of type *COLORREF*.

DB_TYPE_PROPERTY
The data block contains a list of values of type *PROPERTY*.

DB_TYPE_XPROPERTY
The data block contains a list of values of type *XPROPERTY*.

DB_TYPE_FONTDEF
The data block contains a list of values of type *FONTDEF*.

DB_TYPE_TEXT
The data block contains a list of values of type *TEXT*.

DB_TYPE_BINARY
The data block contains a list of values of type *BINARY*.

ElemCount
[*int*] Number of elements that are contained in the data block. If *ElemType* is DB_TYPE_NATIVE, this value should always be zero.

Data
[*const LPVOID*] Address of the data to be copied to the data block.

Bytes
[*int*] Total size of the data in bytes.

Return Value

Return TRUE if the data block was added successfully. If FALSE is returned, this indicates that the user object would be too large after adding this data block.

Comment

None.

Related Topics:

TosoUserOpen

TosoUserGetAddress

TosoUserAddEnd

TosoUserInsert

TosoUserAddEnd (Unit Creation)

Contents **Syntax**

```
BOOL TosoUserAddEnd( void );
```

Adds an end-of-list data block (DB_END) to a previously opened temporary user object and sets the internal length of the user object to the correct value.

Contents **Parameters**

None.

Contents **Return Value**

Return TRUE if the data block was added successfully. If FALSE is returned, this indicates that the user object would be too large after adding this data block.

Contents **Comment**

Once the temporary user object has been terminated using this procedure, no further data block may be added! Nevertheless, its header can still be modified.

Contents

Related Topics:

Contents TosoUserOpen

Contents TosoUserGetAddress

Contents TosoUserAddDataBlock

Contents TosoUserInsert

Contents TosoUserClose

Contents TosoUserFastInsert

TosoUserInsert (Unit Creation)

Syntax

```
UNIT_USER_PTR TosoUserInsert( int DrawingNum,
                              const UNIT_PTR RefObj );
```

Inserts the temporary user object to a memory list. If no block is currently open, the user object is inserted to the main memory list of the drawing behind the object pointed to by *RefObj*. If *RefObj* is NULL, the user object is appended at the end.

If a temporary block is currently open, the user object is appended to that block. *RefObj* is ignored in this case.

Parameters

DrawingNum

[*int*] Zero-based index of the drawing into which the user object shall be inserted. Valid range: 0 <= Value < TVG_DRAWING_MAX.

RefObj

[*const UNIT_PTR*] Address of a reference unit behind which the user object shall be inserted. If *RefObj* is NULL, the user object is appended at the end.

Return Value

Returns the address of the inserted user object. Returns NULL if the insertion was not successful, normally due to insufficient memory.

Comment

In most cases, you can use TosoUserFastInsert instead of this procedure to speed up execution and to reduce code size. See the description of that procedure for details.

Related Topics:

TosoUserOpen

TosoUserGetAddress

TosoUserAddDataBlock

TosoUserAddEnd

TosoUserClose (Unit Creation)

Contents **Syntax**

```
void TosoUserClose( void );
```

Closes a previously opened temporary user object. The user object may not be used afterwards until TosoUserOpen is called again.

Contents **Parameters**

None.

Contents **Return Value**

None.

Contents **Comment**

None.

Contents

Related Topics:

Contents TosoUserOpen

Contents TosoUserGetAddress

Contents TosoUserAddDataBlock

Contents TosoUserAddEnd

Contents TosoUserInsert

Contents TosoUserFastInsert

TosoUserFastInsert (Unit Creation)

Syntax

```
BOOL TosoUserFastInsert( void );
```

This procedure terminates a currently opened temporary user object and inserts it to the current drawing or to a currently open temporary block.

Parameters

None.

Return Value

Returns TRUE if the user object was successfully inserted. Returns FALSE if the insertion was not successful, normally due to insufficient memory.

Comment

This procedure is a kind of makro that executes steps similar to the following code segment:

```
BOOL Result = TRUE;

TosoUserAddEnd();
if( !TosoUserInsert( TosoDrawingGetActive(), NULL ) )
  Result = FALSE;
TosoUserClose();

return( Result );
```

This causes the temporary user object to be terminated, initialised and then inserted to the current drawing. Afterwards, it is closed. For a detailed description, see the referenced procedure's descriptions.

If possible, you should use this command instead of calling the separate procedures to increase speed and to reduce code size.

Contents

Related Topics:

TosoUserOpen

TosoUserGetAddress

TosoUserAddDataBlock

TosoInstanceOpen (Unit Creation)

Contents **Syntax**

```
UNIT_INSTANCE_PTR TosoInstanceOpen( void );
```

Opens a temporary instance for data block input. Once opened, the instance can be filled with data blocks and subsequently be terminated and appended to a memory list or to a block.

Contents **Parameters**

None.

Contents **Return Value**

Address of the temporary instance, if successful. This address can be used to modify the instance's extended properties (located in *ReturnValue->XProperty*). Please do not modify any other value! If NULL is returned, this indicates that another temporary instance is already open.

Contents **Comment**

Only one temporary instance is available at a time. Anyway, other temporary entities like object, user object, block and group can be open simultaneously. After opening the temporary instance, it must later be closed using TosoInstanceClose or TosoInstanceFastInsert.

Contents

Related Topics:

Contents TosoInstanceGetAddress

Contents TosoInstanceAddAttribute

Contents TosoInstanceAddEnd

Contents TosoInstanceInsert

Contents TosoInstanceClose

Contents TosoInstanceFastInsert

TosoInstanceGetAddress (Unit Creation)

Contents **Syntax**

```
UNIT_INSTANCE_PTR TosoInstanceGetAddress( void );
```

This procedure retrieves the address of the currently open instance. It can also be used to check whether an instance is currently open.

Contents **Parameters**

None.

Contents **Return Value**

Returns the address of a previously opened temporary instance. Returns NULL if the instance is not open.

Contents **Comment**

None.

Contents

Related Topics:

Contents TosoInstanceOpen

Contents TosoInstanceAddAttribute

Contents TosoInstanceAddEnd

Contents TosoInstanceInsert

Contents TosoInstanceClose

Contents TosoInstanceFastInsert

TosoInstanceAddAttribute (Unit Creation)

Contents Syntax

```
BOOL TosoInstanceAddAttribute( int Type,
                               const LPSTR Name,
                               const LPSTR Text );
```

Adds an attribute data block (DB_ATTRIB_???) to a previously opened temporary instance.

Contents Parameters

Type

[*int*] Type of the attribute. Possible values are:

DB_ATTRIB_LOCAL_TEXT

The attribute is a local attribute, i.e. an attribute that may be different for each instance. It may contain any type of textual data.

DB_ATTRIB_LOCAL_NUM

The attribute is a local attribute, i.e. an attribute that may be different for each instance. It may contain only texts that represent a valid floating point number.

Name

[*const LPSTR*] This text contains the name of the attribute.

Text

[*const LPSTR*] This text contains the current value of the attribute.

Contents Return Value

Return TRUE if the data block was added successfully. If FALSE is returned, this indicates that the object would be too large after adding this data block.

Contents Comment

None.

Contents

Related Topics:

Contents TosoInstanceOpen

Contents TosoInstanceGetAddress

Contents TosoInstanceAddEnd

TosoInstanceAddEnd (Unit Creation)

Contents **Syntax**

```
BOOL TosoInstanceAddEnd( void );
```

Adds an end-of-list data block (DB_END) to a previously opened temporary instance and sets the internal length of the instance to the correct value.

Contents **Parameters**

None.

Contents **Return Value**

Return TRUE if the data block was added successfully. If FALSE is returned, this indicates that the object would be too large after adding this data block.

Contents **Comment**

Once the temporary instance has been terminated using this procedure, no further data block may be added! Nevertheless, its header can still be modified.

Contents

Related Topics:

Contents TosoInstanceOpen

Contents TosoInstanceGetAddress

Contents TosoInstanceAddAttribute

Contents TosoInstanceInsert

Contents TosoInstanceClose

Contents TosoInstanceFastInsert

TosoInstanceInsert (Unit Creation)

Syntax

```
UNIT_INSTANCE_PTR TosoInstanceInsert( int DrawingNum,
                                      const UNIT_PTR RefObj );
```

Inserts the temporary instance to a memory list. If no block is currently open, the instance is inserted to the main memory list of the drawing behind the object pointed to by *RefObj*. If *RefObj* is NULL, the instance is appended at the end.

If a temporary block is currently open, the instance is appended to that block. *RefObj* is ignored in this case.

Parameters

DrawingNum

[*int*] Zero-based index of the drawing into which the instance shall be inserted. Valid range: 0 <= Value < TVG_DRAWING_MAX.

RefObj

[*const UNIT_PTR*] Address of a reference unit behind which the instance shall be inserted. If *RefObj* is NULL, the instance is appended at the end.

Return Value

Returns the address of the inserted instance. Returns NULL if the insertion was not successful, normally due to insufficient memory.

Comment

In most cases, you can use TosoInstanceFastInsert instead of this procedure to speed up execution and to reduce code size. See the description of that procedure for details.

Related Topics:

TosoInstanceOpen

TosoInstanceGetAddress

TosoInstanceAddAttribute

TosoInstanceAddEnd

TosoInstanceClose (Unit Creation)

Syntax

```
void TosoInstanceClose( void );
```

Closes a previously opened temporary instance. The instance may not be used afterwards until TosoInstanceOpen is called again.

Parameters

None.

Return Value

None.

Comment

None.

Related Topics:

- TosoInstanceOpen
- TosoInstanceGetAddress
- TosoInstanceAddAttribute
- TosoInstanceAddEnd
- TosoInstanceInsert
- TosoInstanceFastInsert

TosoInstanceFastInsert (Unit Creation)

Syntax

```
BOOL TosoInstanceFastInsert( void );
```

This procedure terminates a currently opened temporary instance and inserts it to the current drawing or to a currently open temporary block.

Parameters

None.

Return Value

Returns TRUE if the instance was successfully inserted. Returns FALSE if the insertion was not successful, normally due to insufficient memory.

Comment

This procedure is a kind of makro that executes steps similar to the following code segment:

```
BOOL Result = TRUE;

TosoInstanceAddEnd();
if( !TosoInstanceInsert( TosoDrawingGetActive(), NULL ) )
  Result = FALSE;
TosoInstanceClose();

return( Result );
```

This causes the temporary instance to be terminated, initialised and then inserted to the current drawing. Afterwards, it is closed. For a detailed description, see the referenced procedure's descriptions.

If possible, you should use this command instead of calling the separate procedures to increase speed and to reduce code size.

Related Topics:

TosoInstanceOpen

TosoInstanceGetAddress

TosoInstanceAddAttribute

TosoBlockOpen (Unit Creation)

Syntax

```
UNIT_BLOCK_PTR TosoBlockOpen( void );
```

Opens a temporary block for data block input. Once opened, the block can be filled with data blocks and subsequently be terminated and appended to a memory list.

Parameters

None.

Return Value

Address of the temporary object, if successful. Use this address to set the block's name by copying a string into the *ReturnValue->BlockName* value of the block. This address can also be used to modify the object's extended properties (located in *ReturnValue->XProperty*). Please do not modify any other value! If NULL is returned, this indicates that another temporary object is already open.

Comment

Only one temporary block is available at a time. Anyway, other temporary entities like object, instance, user object and group can be open simultaneously. After opening the temporary block, it must later be closed using TosoBlockClose or TosoBlockFastInsert.

Related Topics:

TosoBlockGetAddress

TosoBlockAddAttribute

TosoBlockAddEnd

TosoBlockInsert

TosoBlockClose

TosoBlockFastInsert

TosoBlockGetAddress (Unit Creation)

Syntax

```
UNIT_BLOCK_PTR TosoBlockGetAddress( void );
```

This procedure retrieves the address of the currently open block. It can also be used to check whether a block is currently open.

Parameters

None.

Return Value

Returns the address of a previously opened temporary block. Returns NULL if the block is not open.

Comment

None.

Related Topics:

TosoBlockOpen

TosoBlockAddAttribute

TosoBlockAddEnd

TosoBlockInsert

TosoBlockClose

TosoBlockFastInsert

TosoBlockAddAttribute (Unit Creation)

Contents Syntax

```
BOOL TosoBlockAddAttribute( int Type,
                            const LPSTR Name,
                            const LPSTR Text );
```

Adds an attribute data block (DB_ATTRIB_???) to a previously opened temporary block.

Contents Parameters

Type

[*int*] Type of the attribute. Possible values are:

DB_ATTRIB_GLOBAL_TEXT
: The attribute is a global attribute, i.e. an attribute that is equal for all instances. It may contain any type of textual data.

DB_ATTRIB_GLOBAL_NUM
: The attribute is a global attribute, i.e. an attribute that is equal for all instances. It may contain only texts that represent a valid floating point number.

DB_ATTRIB_LOCAL_TEXT
: The attribute is a local attribute, i.e. an attribute that may be different for each instance. It may contain any type of textual data.

DB_ATTRIB_LOCAL_NUM
: The attribute is a local attribute, i.e. an attribute that may be different for each instance. It may contain only texts that represent a valid floating point number.

Name

[*const LPSTR*] This text contains the name of the attribute.

Text

[*const LPSTR*] This text contains the current value of the attribute.

Contents Return Value

Return TRUE if the data block was added successfully. If FALSE is returned, this indicates that the object would be too large after adding this data block.

Contents Comment

None.

Contents

Related Topics:

Contents TosoBlockOpen

TosoBlockAddEnd (Unit Creation)

Contents **Syntax**

```
BOOL TosoBlockAddEnd( void );
```

Adds an end-of-list data block (DB_END) to a previously opened temporary block and sets the internal length of the block to the correct value.

Contents **Parameters**

None.

Contents **Return Value**

Return TRUE if the data block was added successfully. If FALSE is returned, this indicates that the object would be too large after adding this data block.

Contents **Comment**

None.

Contents

Related Topics:

Contents TosoBlockOpen

Contents TosoBlockGetAddress

Contents TosoBlockAddAttribute

Contents TosoBlockInsert

Contents TosoBlockClose

Contents TosoBlockFastInsert

TosoBlockInsert (Unit Creation)

Contents **Syntax**

```
UNIT_BLOCK_PTR TosoBlockInsert( int DrawingNum,
                                const LPSTR LibraryName );
```

Appends the temporary block to the block memory list of the drawing or to a library.

Contents **Parameters**

DrawingNum

[*int*] Zero-based index of the drawing into which the block shall be inserted (ignored if *LibraryName* is not equal to TVG_BLOCK_ID). Valid range: 0 <= Value < TVG_DRAWING_MAX.

LibraryName

[*const LPSTR*] Name of the library the block shall be inserted to. If the library name is TVG_BLOCK_ID, the block will be added to the drawing's block memory list.

Contents **Return Value**

Returns the address of the inserted block. Returns NULL if the insertion was not successful, normally due to insufficient memory, or if another block with the same name already exists.

Contents **Comment**

None.

Contents

Related Topics:

Contents TosoBlockOpen

Contents TosoBlockGetAddress

Contents TosoBlockAddAttribute

Contents TosoBlockAddEnd

Contents TosoBlockClose

Contents TosoBlockFastInsert

TosoBlockClose (Unit Creation)

Contents **Syntax**

```
void TosoBlockClose( void );
```

Closes a previously opened temporary block. The block may not be used afterwards until TosoBlockOpen is called again.

Contents **Parameters**

None.

Contents **Return Value**

None.

Contents **Comment**

None.

Contents

Related Topics:

Contents TosoBlockOpen

Contents TosoBlockGetAddress

Contents TosoBlockAddAttribute

Contents TosoBlockAddEnd

Contents TosoBlockInsert

Contents TosoBlockFastInsert

TosoBlockFastInsert (Unit Creation)

Contents Syntax

```
XPROPERTY* TosoBlockFastInsert( BOOL CreateInstance,
                                MATRIX* Matrix );
```

This procedure terminates a currently opened temporary block and inserts it to the block memory list of the current drawing.

Contents Parameters

CreateInstance

[*BOOL*] If this value is TRUE, the serving application will automatically create an instance that references the block and inserts it into the drawing.

Matrix

[*MATRIX**] Address of a display matrix that is used to initialise the instance's display matrix if *CreateInstance* is TRUE. If *Matrix* is NULL, the instance will be initialised with a standard matrix, setting the instance to (0.0, 0.0) with a scaling of 1.0.

Contents Return Value

If *CreateInstance* is TRUE, the procedure returns the address of the block's instance's *XProperty* component if successful. This address can be used to alter the instance's properties. If *CreateInstance* is FALSE, it returns the address of the block's *XProperty* component if successful. If NULL is returned, the insertion was not successful, normally due to insufficient memory, or if another block with the same name already exists.

Contents Comment

This procedure is a kind of makro that executes steps similar to the following code segment:

```
BOOL Result = TRUE;

TosoBlockAddEnd();
if( !TosoBlockInsert( TosoDrawingGetActive(), TVG_BLOCK_ID ) )
  Result = FALSE;
TosoBlockClose();

if( CreateInstance )
  ...

return( Result );
```

This causes the temporary block to be terminated, initialised and then inserted to the current drawing. Afterwards, it is closed. For a detailed description, see the referenced procedure's descriptions.

If possible, you should use this command instead of calling the separate procedures to increase speed and to reduce code size.

Related Topics:

TosoBlockOpen

TosoBlockGetAddress

TosoBlockAddAttribute

TosoBlockAddEnd

TosoBlockInsert

TosoBlockClose

TosoGroupOpen (Unit Creation)

Syntax

```
UNIT_BLOCK_PTR TosoGroupOpen( void );
```

Opens a temporary group for data block input. Once opened, the group can be filled with data blocks and subsequently be terminated and appended to a memory list.

Parameters

None.

Return Value

Address of the temporary group, if successful. Use this address to retrieve the group's name by reading the string located at *ReturnValue->BlockName*. This address can also be used to modify the group's extended properties (located in *ReturnValue->XProperty*). Please do not modify any other value! If NULL is returned, this indicates that another temporary group is already open.

Comment

Only one temporary group is available at a time. Anyway, other temporary entities like object, instance, user object and block can be open simultaneously. After opening the temporary group, it must later be closed using TosoGroupClose or TosoGroupFastInsert.

Related Topics:

TosoGroupGetAddress

TosoGroupAddEnd

TosoGroupInsert

TosoGroupClose

TosoGroupFastInsert

TosoGroupGetAddress (Unit Creation)

Contents **Syntax**

```
UNIT_BLOCK_PTR TosoGroupGetAddress( void );
```

This procedure retrieves the address of the currently open group. It can also be used to check whether a group is currently open.

Contents **Parameters**

None.

Contents **Return Value**

Returns the address of a previously opened temporary group. Returns NULL if the group is not open.

Contents **Comment**

None.

Contents

Related Topics:

Contents TosoGroupOpen

Contents TosoGroupAddEnd

Contents TosoGroupInsert

Contents TosoGroupClose

Contents TosoGroupFastInsert

TosoGroupAddEnd (Unit Creation)

Contents **Syntax**

```
BOOL TosoGroupAddEnd( void );
```

Adds an end-of-list data block (DB_END) to a previously opened temporary group and sets the internal length of the group to the correct value.

Contents **Parameters**

None.

Contents **Return Value**

Return TRUE if the data group was added successfully. If FALSE is returned, this indicates that the object would be too large after adding this data block.

Contents **Comment**

None.

Contents

Related Topics:

Contents TosoGroupOpen

Contents TosoGroupGetAddress

Contents TosoGroupInsert

Contents TosoGroupClose

Contents TosoGroupFastInsert

TosoGroupInsert (Unit Creation)

Contents **Syntax**

```
UNIT_BLOCK_PTR TosoGroupInsert( int DrawingNum );
```

Appends the temporary group to the block memory list of the drawing.

Contents **Parameters**

DrawingNum

[*int*] Zero-based index of the drawing into which the group shall be inserted. Valid range: 0 <= Value < TVG_DRAWING_MAX.

Contents **Return Value**

Returns the address of the inserted group. Returns NULL if the insertion was not successful, normally due to insufficient memory, or if another group with the same name already exists.

Contents **Comment**

None.

Contents

Related Topics:

Contents TosoGroupOpen

Contents TosoGroupGetAddress

Contents TosoGroupAddEnd

Contents TosoGroupClose

Contents TosoGroupFastInsert

TosoGroupClose (Unit Creation)

Contents **Syntax**

```
void TosoGroupClose( void );
```

Closes a previously opened temporary group. The group may not be used afterwards until TosoGroupOpen is called again.

Contents **Parameters**

None.

Contents **Return Value**

None.

Contents **Comment**

None.

Contents

Related Topics:

Contents TosoGroupOpen

Contents TosoGroupGetAddress

Contents TosoGroupAddEnd

Contents TosoGroupInsert

Contents TosoGroupFastInsert

TosoGroupFastInsert (Unit Creation)

Contents **Syntax**

```
XPROPERTY* TosoGroupFastInsert( BOOL CreateInstance,
                                MATRIX* Matrix );
```

This procedure terminates a currently opened temporary group and inserts it to the current drawing.

Contents **Parameters**

CreateInstance

[*BOOL*] If this value is TRUE, the serving application will automatically create an instance that references the group and inserts it into the drawing.

Matrix

[*MATRIX**] Address of a display matrix that is used to initialise the instance's display matrix if *CreateInstance* is TRUE. If *Matrix* is NULL, the instance will be initialised with a standard matrix, setting the instance to (0.0, 0.0) with a scaling of 1.0.

Contents **Return Value**

If *CreateInstance* is TRUE, the procedure returns the address of the group's instance's *XProperty* component if successful. This address can be used to alter the instance's properties. If *CreateInstance* is FALSE, it returns the address of the group's *XProperty* component if successful. If NULL is returned, the insertion was not successful, normally due to insufficient memory, or if another group with the same name already exists.

Contents **Comment**

This procedure is a kind of makro that executes steps similar to the following code segment:

```
BOOL Result = TRUE;

TosoGroupAddEnd();
if( !TosoGroupInsert( TosoDrawingGetActive() ) )
  Result = FALSE;
TosoGroupClose();

if( CreateInstance )
  ...

return( Result );
```

This causes the temporary group to be terminated, initialised and then inserted to the current drawing. Afterwards, it is closed. For a detailed description, see the referenced procedure's descriptions.

If possible, you should use this command instead of calling the separate procedures to increase speed and to reduce code size.

Related Topics:

TosoGroupOpen

TosoGroupGetAddress

TosoGroupAddEnd

TosoGroupInsert

TosoGroupClose

TosoEditDelete (Unit Editing)

Contents Syntax

```
void TosoEditDelete( int DrawingNum,
                     UNIT_PTR UnitPtr );
```

Deletes the given unit maintaing its undo structure, i.e. it can be undeleted by the user.

Contents Parameters

DrawingNum

[*int*] Zero-based index of the drawing whose unit shall be deleted. Valid range: 0 <= Value < TVG_DRAWING_MAX.

UnitPtr

[*UNIT_PTR*] Address of the unit to be deleted. Make sure that this address points to a unit located in the drawing stated by *DrawingNum*! If not, the system might reveal unexpected behaviour.

Contents Return Value

None.

Contents Comment

This procedure may only be called inside a TosoUndoInitProcess and TosoUndoFinishProcess bracket.

Contents

Related Topics:

Contents TosoEditIdentCount

Contents TosoEditIdentMatrix

Contents TosoEditIdentEnumModify

Contents TosoEditPointsCount

Contents TosoEditPointsMatrix

TosoEditIdentCount (Unit Editing)

Syntax

```
long TosoEditIdentCount( int DrawingNum,
                         DRECT* Frame );
```

Retrieves the number of identified entities and their surrounding frame. This will only work if the current command forced a multi-unit identification.

Parameters

DrawingNum

[*int*] Zero-based index of the drawing in which the identified units shall be count. Valid range: 0 <= Value < TVG_DRAWING_MAX.

Frame

[*DRECT**] Address of a rectangle buffer to receive the surrounding frame of all identified units.

Return Value

Number of identified units, 0 is no unit is identified.

Comment

Multi-unit identification is forced by using point types whose names begin with POINT_ID_MULTI (but not with POINT_ID_MULTIPOINT). For a complete list of point types, see Point Types.

Related Topics:

TosoEditDelete

TosoEditIdentMatrix

TosoEditIdentEnumModify

TosoEditPointsCount

TosoEditPointsMatrix

TosoEditIdentMatrix (Unit Editing)

Syntax

```
BOOL TosoEditIdentMatrix( int DrawingNum,
                          const MATRIX* Matrix,
                          BOOL Duplicate )
```

Multiplies all identified entities with the given matrix. This will only work if the current command forced a multi-unit identification.

Parameters

DrawingNum

[*int*] Zero-based index of the drawing whose identified units shall be modified. Valid range: 0 <= Value < TVG_DRAWING_MAX.

Matrix

[*const MATRIX**] Matrix with which all identified units shall be multiplied. If *Matrix* is NULL, the units will not be modified.

Duplicate

[*BOOL*] Determines whether the units themselves shall be modified or a copy of them. This is equivalent to the "Duplicate" function in the serving application.

Return Value

Returns TRUE if successful. If FALSE is returned, this is usually due to insufficient memory, especially if *Duplicate* is TRUE.

Comment

This procedure may only be called inside a TosoUndoInitProcess and TosoUndoFinishProcess bracket.

Contents

Related Topics:

TosoEditDelete

TosoEditIdentCount

TosoEditIdentEnumModify

TosoEditPointsCount

Contents

TosoEditIdentEnumModify (Unit Editing)

Contents **Syntax**

```
BOOL TosoEditIdentEnumModify( int DrawingNum,
                              TOSOENUMOBJECT_PROC CallBack,
                              BOOL Duplicate )
```

Enumerates all identified entities allowing the module to modify them. This will only work if the current command forced a multi-unit identification.

Contents **Parameters**

DrawingNum

[*int*] Zero-based index of the drawing whose identified units shall be modified. Valid range: 0 <= Value < TVG_DRAWING_MAX.

CallBack

[*TOSOENUMOBJECT_PROC*] Callback procedure for an object enumeration. This procedure will be called for every enumeration step. For a detailed description of object enumeration, see also the description of the TosoEnumObjectProc procedure. The enumeration will always be done using the ENUMMODE_NATIVE mode, i.e. only the addresses of the identified entities are passed to the callback procedure.

Duplicate

[*BOOL*] Determines whether the units themselves shall be modified or a copy of them. This is equivalent to the "Duplicate" function in the serving application.

Contents **Return Value**

Returns TRUE if successful. If FALSE is returned, this is usually due to insufficient memory, especially if *Duplicate* is TRUE.

Contents **Comment**

During this type of modification, the memory structure of the entities may not be altered, only the entity's header and the data areas of its data blocks may be modified!

This procedure may only be called inside a TosoUndoInitProcess and TosoUndoFinishProcess bracket.

Contents

Related Topics:

Contents TosoEditDelete

Contents TosoEditIdentCount

TosoEditPointsCount (Unit Editing)

Contents Syntax

```
long TosoEditPointsCount( int DrawingNum,
                          DRECT* Frame );
```

Retrieves the number of identified points and their surrounding frame. This will only work if the current command forced a multi-point identification.

Contents Parameters

DrawingNum

[*int*] Zero-based index of the drawing in which the identified points shall be count. Valid range: 0 <= Value < TVG_DRAWING_MAX.

Frame

[*DRECT**] Address of a rectangle buffer to receive the surrounding frame of all identified points.

Contents Return Value

Number of identified points, 0 is no point is identified.

Contents Comment

Multi-point identification is forced by using the point type POINT_ID_MULTIPOINT. For a complete list of point types, see Point Types.

Contents

Related Topics:

Contents TosoEditDelete

Contents TosoEditIdentCount

Contents TosoEditIdentMatrix

Contents TosoEditIdentEnumModify

Contents TosoEditPointsMatrix

TosoEditPointsMatrix (Unit Editing)

Contents Syntax

```
BOOL TosoEditPointsMatrix( int DrawingNum,
                           const MATRIX* Matrix,
                           BOOL Duplicate )
```

Multiplies all identified points with the given matrix. This will only work if the current command forced a multi-unit identification.

Contents Parameters

DrawingNum

[*int*] Zero-based index of the drawing whose identified points shall be modified. Valid range: 0 <= Value < TVG_DRAWING_MAX.

Matrix

[*const MATRIX**] Matrix with which all identified points shall be multiplied. If *Matrix* is NULL, the points will not be modified.

Duplicate

[*BOOL*] Determines whether the units containing the identified points shall be modified themselves or a copy of them. This is equivalent to the "Duplicate" function in the serving application.

Contents Return Value

Returns TRUE if successful. If FALSE is returned, this is usually due to insufficient memory, especially if *Duplicate* is TRUE.

Contents Comment

This procedure may only be called inside a TosoUndoInitProcess and TosoUndoFinishProcess bracket.

Contents

Related Topics:

Contents TosoEditDelete

Contents TosoEditIdentCount

Contents TosoEditIdentMatrix

Contents TosoEditIdentEnumModify

Contents

TosoEnumerateAll (Unit Enumeration)

Contents **Syntax**

```
BOOL TosoEnumerateAll( int DrawingNum,
                       int EnumFlag1,
                       int EnumFlag2,
                       int EnumMode,
                       TOSOENUMOBJECT_PROC CallBack );
```

Starts an enumeration stream. This stream enumerates all objects and instances of the given drawing depending on *EnumFlag1*, *EnumFlag2* and *EnumMode*.

Contents **Parameters**

DrawingNum

[*int*] Zero-based index of the drawing whose units shall be enumerated. Valid range: 0 <= Value < TVG_DRAWING_MAX.

EnumFlag1,
EnumFlag2

[*int*] Two flag combinations used to filter special units for the enumeration. The enumeration will only include units where the following boolean expression is true:

((*UnitPtr->Header.Flag* & *EnumFlag1*) = = *EnumFlag2*)

Both values can be a bitwise OR combination of the following flags:

FLAG_SELECT
: This flag is set if the unit is permanently selected.

FLAG_IDENT
: This flag is set if the unit is currently highlighted for final selection due to an ambigous identification.

FLAG_USE
: This flag is set if the unit has been identified in a multi-object identification.

FLAG_POINT
: This flag is set if at least one definition point inside the unit is selected.

FLAG_INTERNAL
: This flag is set if the unit is a temporary unit.

FLAG_NODISPLAY
: This flag is set if the unit lies in layer that will not be displayed on screen.

FLAG_NOOUTPUT
: This flag is set if the unit lies in layer that will not be output to printer, clipboard etc.

FLAG_FROZEN
: This flag is set if the unit lies in a frozen layer.

FLAG_IDLE
: This flag is set if the unit lies in an ignored layer.

Typical combinations of *EnumFlag1* and *EnumFlag2* are:

EnumFlag1 = FLAG_SELECT, *EnumFlag2* = FLAG_SELECT

Enumerate only permanently selected units.

EnumFlag1 = FLAG_USE, *EnumFlag2* = FLAG_USE

Enumerate only identified units.

EnumFlag1 = FLAG_USE | FLAG_FROZEN | FROZEN_NODISPLAY, *EnumFlag2* = FLAG_USE

Enumerate only identified units that are neither invisible nor frozen.

EnumMode

[*int*] This values determines the way that units are enumerated. It indicates what types of geometrical data the module can handle, and what type of units shall be enumerated. It can be a bitwise OR combination of the following flags:

Flags for Geometrical Data Type

These flags tell the application what kind of geometrical data the module is able to handle. All other geometrical data will be converted to those types.

If, e.g., the module cannot handle circular arcs but Bézier curves, all circles and circular arcs will be converted to a sequence of Bézier curves.

ENUMMODE_LINES

The module can only handle solid polylines with line color and line width, i.e. lines using line patterns will be resolved to single line segments. This is the basic mode which can be combined with one or more of the following modes to allow more complex geometric data.

ENUMMODE_FILLS

The module can handle filled, nested polygons.

ENUMMODE_ARCS

The module can handle circular arcs in polylines / polygons.

ENUMMODE_BEZIERS

The module can handle Bézier curves in polylines / polygons.

ENUMMODE_LINETYPES

The module can handle line patterns.

ENUMMODE_SOLID_CHARS

The module can handle solid characters, i.e. characters that are filled and non-distorted and can thus be displayed using standard system fonts. Appropriate texts will not be resolved into polylines / polygons, but passed to the module as single characters.
Can be combined with ENUMMODE_OTHER_CHARS to allow the passing of all types of characters.

ENUMMODE_OTHER_CHARS

The module can handle non-solid characters, i.e. characters that are framed or distorted and can thus not be displayed using standard system fonts. Appropriate texts will not be resolved into polylines / polygons, but passed to the module as single characters.
Can be combined with ENUMMODE_SOLID_CHARS to allow the passing of all types of characters.

ENUMMODE_SOLID_TEXTS

The module can handle null-terminated strings with format information being filled and non-distorted and can thus be displayed using standard system fonts. Appropriate texts will not be resolved into polylines / polygons, but passed to the module as complete text lines.
Can be combined with ENUMMODE_OTHER_TEXTS to allow the passing of all types of texts.

ENUMMODE_OTHER_TEXTS

The module can handle null-terminated strings with format information being framed or distorted and can thus not be displayed using standard system fonts. Appropriate texts will not be resolved

into polylines / polygons, but passed to the module as complete text lines.
Can be combined with ENUMMODE_SOLID_TEXTS to allow the passing of all types of texts.

ENUMMODE_MATRIX
The module can handle 3×2 matrices, i.e. it is able to apply them to *all* types of data. In this case, the serving application will pass the display matrix directly to the module instead of applying it to each unit.

ENUMMODE_NATIVE
The module can handle native object data, i.e. only the addresses of the units will be passed.

Flags for Handling of Blocks and Instances
These flags tell the application how to handle instances and blocks. If required, the application recurses through the calling tree of instances and blocks and resolves it.

ENUMMODE_PLAIN
If this flag is set, all blocks and instances will be resolved before passing data to the module, i.e. no instances will be passed.
If neither ENUMMODE_PLAIN nor ENUMMODE_STRUCTURED is set, no object enumeration will be performed. In this case, only ENUMMODE_USED_BLOCKS and/or ENUMMODE_USED_FONTS will be recognized.

ENUMMODE_STRUCTURED
If this flag is set, instances will directly be passed to the module. It's the module's duty to retrieve the correpsonding block definition if required.
Usually, a module being able to handle instances and blocks will first make an enumeration by means of TosoEnumerateAll with ENUMMODE_USED_BLOCKS set in order to get information about the blocks used. Then it will enumerate all blocks reported manually by means of TosoEnumerateBlock. Finally, it enumerates the objects and instances by calling TosoEnumerateAll again with ENUMMODE_STRUCTURED set.

If neither ENUMMODE_PLAIN nor ENUMMODE_STRUCTURED is set, no object enumeration will be performed. In this case, only ENUMMODE_USED_BLOCKS and/or ENUMMODE_USED_FONTS will be recognized.

Flags for Retrieving Information
These flags tell the application to return information about blocks and fonts used in the units. This can be used to determines what blocks have to be exported or whether all fonts used are available.

ENUMMODE_USED_BLOCKS
If this flag is set, the procedure will list all block references found. This can be used by the module to find out what blocks are used in the units to be enumerated. If a block is used several times, it will be enumerated more than once!

ENUMMODE_USED_FONTS
If this flag is set, the procedure will list all font references found. This can be used by the module to find out what fonts are used in the units to be enumerated. If a font is used several times, it will be enumerated more than once!

If either ENUMMODE_USED_BLOCKS and/or ENUMMODE_USED_FONTS is set, the flags ENUMMODE_PLAIN and ENUMMODE_STRUCTURED will be ignored, i.e. no objects or instances will be passed.

Flags for Unit Type Limitation
These flags tell the application to ignore special types of units. In many cases, it is not useful to have fonts resolved and passed during enumeration, especially if only basic geometrical information is required like during hatching or surface generation.

ENUMMODE_NO_OBJECTS
: If this flag is set, the procedure will *not* enumerate objects, i.e. only instances and referenced blocks / instances will be enumerated.

ENUMMODE_NO_BLOCKS
: If this flag is set, the procedure will *not* enumerate instances and referenced blocks, i.e. only objects will be enumerated.

ENUMMODE_NO_FONTS
: If this flag is set, the procedure will *not* enumerate characters of any type. This applies to characters in text objects as well as in dimensions.

Flags for Property Handling

These flags tell the application which property set to use. As pens define two sets of properties (one for screen display and one for printer output), the resulting image depends on which property set is used. In addition, the serving application features several settings that influence the properties (e.g. layers transmitting properties) and the kind of units used (e.g. whether to display geometry or not).

ENUMMODE_SCREEN_PROP
: If this flag is set, the procedure will use the application's current settings for screen display. This will determine whether geometry objects will be visible, and whether layers will transmit properties or not.

ENUMMODE_OUTPUT_PROP
: If this flag is set, the procedure will use the application's current settings for printer output. This will determine whether geometry objects will be visible, and whether layers will transmit properties or not.

If neither ENUMMODE_SCREEN_PROP nor ENUMMODE_OUTPUT_PROP is set, the printer output settings will be used.

CallBack
: [*TOSOENUMOBJECT_PROC*] Callback procedure for an object enumeration. This procedure will be called for every enumeration step. For a detailed description of object enumeration, see also the description of the TosoEnumObjectProc procedure.

Return Value

Returns TRUE if the enumeration was finished successfully, or FALSE if the TosoEnumObjectProc procedure returned FALSE at its last call.

Comment

For a detailed description of entity enumeration, see the description of the corresponding callback procedure TosoEnumObjectProc.

Related Topics:

TosoEnumerateLibrary

TosoEnumerateUnit

TosoEnumerateBlock

TosoEnumerateChar

TosoEnumerateInstanceAttrib

TosoEnumerateBlockAttrib

TosoEnumerateIdent

TosoEnumeratePoints

TosoEnumerateLibrary (Unit Enumeration)

Contents

Syntax

```
BOOL TosoEnumerateLibrary( int DrawingNum,
                           const LPSTR LibraryName,
                           int EnumMode,
                           TOSOENUMOBJECT_PROC CallBack );
```

Enumerates all blocks defined in a library.

Contents

Parameters

DrawingNum

[*int*] Zero-based index of the drawing whose blocks shall be enumerated (ignored if *LibraryName* is not equal to TVG_BLOCK_ID). Valid range: 0 <= Value < TVG_DRAWING_MAX.

LibraryName

[*const LPSTR*] Title of the library to be enumerated. This library title can be retrieved by means of the TosoLibraryGetInfo procedure. If you want to enumerate the blocks defined in a drawing, set this value to TVG_BLOCK_ID.

EnumMode

[*int*] This values determines the way that units are enumerates. When enumerating a library, only two enumeration modes are available:

ENUMMODE_NATIVE

If this flag is set, all blocks will be enumerated in native form, ie. only the address of the block will be passed.

ENUMMODE_USED_BLOCKS

If this flag is set, the procedure will list all blocks found. This can be used by the module to find out what blocks are used in the units to be enumerated.

Either ENUMMODE_NATIVE or ENUMMODE_USED_BLOCKS have to be set! If both are set, only the native data will be passed.

CallBack

[*TOSOENUMOBJECT_PROC*] Callback procedure for an object enumeration. This procedure will be called for every enumeration step. For a detailed description of object enumeration, see also the description of the TosoEnumObjectProc procedure.

Contents

Return Value

Returns TRUE if the enumeration was finished successfully, or FALSE if the TosoEnumObjectProc procedure returned FALSE at its last call.

Contents

Comment

For a detailed description of entity enumeration, see the description of the corresponding callback procedure TosoEnumObjectProc.

Related Topics:


TosoEnumerateAll


TosoEnumerateUnit


TosoEnumerateBlock


TosoEnumerateChar


TosoEnumerateInstanceAttrib


TosoEnumerateBlockAttrib


TosoEnumerateIdent


TosoEnumeratePoints

TosoEnumerateUnit (Unit Enumeration)

Contents

Syntax

```
BOOL TosoEnumerateUnit( int DrawingNum,
                        const UNIT_PTR UnitPtr,
                        int EnumMode,
                        TOSOENUMOBJECT_PROC CallBack );
```

Enumerates a single unit.

Contents

Parameters

DrawingNum

[*int*] Zero-based index of the drawing whose units shall be enumerated. Valid range: 0 <= Value < TVG_DRAWING_MAX.

UnitPtr

[*const UNIT_PTR*] Address of the unit to be enumerated.

EnumMode

[*int*] This values determines the way that units are enumerates. It indicates what types of geometrical data the module can handle, and what type of units shall be enumerated. It can be a bitwise OR combination of the following flags:

Flags for Geometrical Data Type

These flags tell the application what kind of geometrical data the module is able to handle. All other geometrical data will be converted to those types.

If, e.g., the module cannot handle circular arcs but Bézier curves, all circles and circular arcs will be converted to a sequence of Bézier curves.

ENUMMODE_LINES

The module can only handle solid polylines with line color and line width, i.e. line patterns will be resolved to single line segments. This is the basic mode which can be combined with one or more of the following modes to allow more complex geometric data.

ENUMMODE_FILLS

The module can handle filled, nested polygons.

ENUMMODE_ARCS

The module can handle circular arcs in polylines / polygons.

ENUMMODE_BEZIERS

The module can handle Bézier curves in polylines / polygons.

ENUMMODE_LINETYPES

The module can handle line patterns.

ENUMMODE_SOLID_CHARS

The module can handle solid characters, i.e. characters that are filled and non-distorted and can thus be displayed using standard system fonts. Appropriate texts will not be resolved into polylines / polygons, but passed to the module as single characters.

Can be combined with ENUMMODE_OTHER_CHARS to allow the passing of all types of characters.

ENUMMODE_OTHER_CHARS

The module can handle non-solid characters, i.e. characters that are framed or distorted and can

thus not be displayed using standard system fonts. Appropriate texts will not be resolved into polylines / polygons, but passed to the module as single characters.
Can be combined with ENUMMODE_SOLID_CHARS to allow the passing of all types of characters.

ENUMMODE_SOLID_TEXTS
The module can handle null-terminated strings with format information being filled and non-distorted and can thus be displayed using standard system fonts. Appropriate texts will not be resolved into polylines / polygons, but passed to the module as complete text lines.
Can be combined with ENUMMODE_OTHER_TEXTS to allow the passing of all types of texts.

ENUMMODE_OTHER_TEXTS
The module can handle null-terminated strings with format information being framed or distorted and can thus not be displayed using standard system fonts. Appropriate texts will not be resolved into polylines / polygons, but passed to the module as complete text lines.
Can be combined with ENUMMODE_SOLID_TEXTS to allow the passing of all types of texts.

ENUMMODE_MATRIX
The module can handle 3×2 matrices, i.e. it is able to apply them to *all* types of data. In this case, the serving application will pass the display matrix directly to the module instead of applying it to each unit.

ENUMMODE_NATIVE
The module can handle native object data, i.e. only the addresses of the units will be passed.

Flags for Handling of Blocks and Instances
These flags tell the application how to handle instances and blocks. If required, the application recurses through the calling tree of instances and blocks and resolves it.

ENUMMODE_PLAIN
If this flag is set, all blocks and instances will be resolved before passing data to the module, i.e. no instances will be passed.
If neither ENUMMODE_PLAIN nor ENUMMODE_STRUCTURED is set, no object enumeration will be performed. In this case, only ENUMMODE_USED_BLOCKS and/or ENUMMODE_USED_FONTS will be recognized.

ENUMMODE_STRUCTURED
If this flag is set, instances will directly be passed to the module. It's the module's duty to retrieve the correpsonding block definition if required.
Usually, a module being able to handle instances and blocks will first make an enumeration by means of TosoEnumerateAll with ENUMMODE_USED_BLOCKS set in order to get information about the blocks used. Then it will enumerate all blocks reported manually by means of TosoEnumerateBlock. Finally, it enumerates the objects and instances by calling TosoEnumerateAll again with ENUMMODE_STRUCTURED set.

If neither ENUMMODE_PLAIN nor ENUMMODE_STRUCTURED is set, no object enumeration will be performed. In this case, only ENUMMODE_USED_BLOCKS and/or ENUMMODE_USED_FONTS will be recognized.

Flags for Retrieving Information
These flags tell the application to return information about blocks and fonts used in the units. This can be used to determines what blocks have to be exported or whether all fonts used are available.

ENUMMODE_USED_BLOCKS
If this flag is set, the procedure will list all block references found. This can be used by the module to find out what blocks are used in the units to be enumerated. If a block is used several

times, it will be enumerated more than once!

ENUMMODE_USED_FONTS
: If this flag is set, the procedure will list all font references found. This can be used by the module to find out what fonts are used in the units to be enumerated. If a font is used several times, it will be enumerated more than once!

If either ENUMMODE_USED_BLOCKS and/or ENUMMODE_USED_FONTS is set, the flags ENUMMODE_PLAIN and ENUMMODE_STRUCTURED will be ignored, i.e. no objects or instances will be passed.

Flags for Unit Type Limitation
These flags tell the application to ignore special types of units. In many cases, it is not useful to have fonts resolved and passed during enumeration, especially if only basic geometrical information is required like during hatching or surface generation.

ENUMMODE_NO_OBJECTS
: If this flag is set, the procedure will *not* enumerate objects, i.e. only instances and referenced blocks / instances will be enumerated.

ENUMMODE_NO_BLOCKS
: If this flag is set, the procedure will *not* enumerate instances and referenced blocks, i.e. only objects will be enumerated.

ENUMMODE_NO_FONTS
: If this flag is set, the procedure will *not* enumerate characters of any type. This applies to characters in text objects as well as in dimensions.

Flags for Property Handling
These flags tell the application which property set to use. As pens define two sets of properties (one for screen display and one for printer output), the resulting image depends on which property set is used. In addition, the serving application features several settings that influence the properties (e.g. layers transmitting properties) and the kind of units used (e.g. whether to display geometry or not).

ENUMMODE_SCREEN_PROP
: If this flag is set, the procedure will use the application's current settings for screen display. This will determine whether geometry objects will be visible, and whether layers will transmit properties or not.

ENUMMODE_OUTPUT_PROP
: If this flag is set, the procedure will use the application's current settings for printer output. This will determine whether geometry objects will be visible, and whether layers will transmit properties or not.

If neither ENUMMODE_SCREEN_PROP nor ENUMMODE_OUTPUT_PROP is set, the printer output settings will be used.

CallBack
: [*TOSOENUMOBJECT_PROC*] Callback procedure for an object enumeration. This procedure will be called for every enumeration step. For a detailed description of object enumeration, see also the description of the TosoEnumObjectProc procedure.

Contents

Return Value
Returns TRUE if the enumeration was finished successfully, or FALSE if the TosoEnumObjectProc procedure returned FALSE at its last call.

Contents **Comment**

For a detailed description of unit enumeration, see the description of the corresponding callback procedure TosoEnumObjectProc.

Contents

Related Topics:

Contents TosoEnumerateAll

Contents TosoEnumerateLibrary

Contents TosoEnumerateBlock

Contents TosoEnumerateChar

Contents TosoEnumerateInstanceAttrib

Contents TosoEnumerateBlockAttrib

Contents TosoEnumerateIdent

Contents TosoEnumeratePoints

TosoEnumerateBlock (Unit Enumeration)

Contents Syntax

```
BOOL TosoEnumerateBlock( int DrawingNum,
                         const LPSTR BlockName,
                         const LPSTR LibraryName,
                         int EnumMode,
                         TOSOENUMOBJECT_PROC CallBack );
```

Enumerates a single block definition.

Contents Parameters

DrawingNum

[*int*] Zero-based index of the drawing whose blocks shall be enumerated (ignored if *LibraryName* is not equal to TVG_BLOCK_ID). Valid range: 0 <= Value < TVG_DRAWING_MAX.

BlockName

[*const LPSTR*] Name of the block to be enumerated.

LibraryName

[*const LPSTR*] Name of the library the block is located in.

EnumMode

[*int*] This values determines the way that units are enumerates. It indicates what types of geometrical data the module can handle, and what type of units shall be enumerated. It can be a bitwise OR combination of the following flags:

Flags for Geometrical Data Type

These flags tell the application what kind of geometrical data the module is able to handle. All other geometrical data will be converted to those types.

If, e.g., the module cannot handle circular arcs but Bézier curves, all circles and circular arcs will be converted to a sequence of Bézier curves.

ENUMMODE_LINES

The module can only handle solid polylines with line color and line width, i.e. line patterns will be resolved to single line segments. This is the basic mode which can be combined with one or more of the following modes to allow more complex geometric data.

ENUMMODE_FILLS

The module can handle filled, nested polygons.

ENUMMODE_ARCS

The module can handle circular arcs in polylines / polygons.

ENUMMODE_BEZIERS

The module can handle Bézier curves in polylines / polygons.

ENUMMODE_LINETYPES

The module can handle line patterns.

ENUMMODE_SOLID_CHARS

The module can handle solid characters, i.e. characters that are filled and non-distorted and can thus be displayed using standard system fonts. Appropriate texts will not be resolved into polylines / polygons, but passed to the module as single characters.

Can be combined with ENUMMODE_OTHER_CHARS to allow the passing of all types of characters.

ENUMMODE_OTHER_CHARS

The module can handle non-solid characters, i.e. characters that are framed or distorted and can thus not be displayed using standard system fonts. Appropriate texts will not be resolved into polylines / polygons, but passed to the module as single characters.
Can be combined with ENUMMODE_SOLID_CHARS to allow the passing of all types of characters.

ENUMMODE_SOLID_TEXTS

The module can handle null-terminated strings with format information being filled and non-distorted and can thus be displayed using standard system fonts. Appropriate texts will not be resolved into polylines / polygons, but passed to the module as complete text lines.
Can be combined with ENUMMODE_OTHER_TEXTS to allow the passing of all types of texts.

ENUMMODE_OTHER_TEXTS

The module can handle null-terminated strings with format information being framed or distorted and can thus not be displayed using standard system fonts. Appropriate texts will not be resolved into polylines / polygons, but passed to the module as complete text lines.
Can be combined with ENUMMODE_SOLID_TEXTS to allow the passing of all types of texts.

ENUMMODE_MATRIX

The module can handle 3×2 matrices, i.e. it is able to apply them to *all* types of data. In this case, the serving application will pass the display matrix directly to the module instead of applying it to each unit.

ENUMMODE_NATIVE

The module can handle native object data, i.e. only the addresses of the units will be passed.

Flags for Handling of Blocks and Instances

These flags tell the application how to handle instances and blocks. If required, the application recurses through the calling tree of instances and blocks and resolves it.

ENUMMODE_PLAIN

If this flag is set, all blocks and instances will be resolved before passing data to the module, i.e. no instances will be passed.
If neither ENUMMODE_PLAIN nor ENUMMODE_STRUCTURED is set, no object enumeration will be performed. In this case, only ENUMMODE_USED_BLOCKS and/or ENUMMODE_USED_FONTS will be recognized.

ENUMMODE_STRUCTURED

If this flag is set, instances will directly be passed to the module. It's the module's duty to retrieve the correpsonding block definition if required.
Usually, a module being able to handle instances and blocks will first make an enumeration by means of TosoEnumerateAll with ENUMMODE_USED_BLOCKS set in order to get information about the blocks used. Then it will enumerate all blocks reported manually by means of TosoEnumerateBlock. Finally, it enumerates the objects and instances by calling TosoEnumerateAll again with ENUMMODE_STRUCTURED set.

If neither ENUMMODE_PLAIN nor ENUMMODE_STRUCTURED is set, no object enumeration will be performed. In this case, only ENUMMODE_USED_BLOCKS and/or ENUMMODE_USED_FONTS will be recognized.

Flags for Retrieving Information

These flags tell the application to return information about blocks and fonts used in the units. This can be used to determines what blocks have to be exported or whether all fonts used are available.

ENUMMODE_USED_BLOCKS

If this flag is set, the procedure will list all block references found. This can be used by the module to find out what blocks are used in the units to be enumerated. If a block is used several times, it will be enumerated more than once!

ENUMMODE_USED_FONTS

If this flag is set, the procedure will list all font references found. This can be used by the module to find out what fonts are used in the units to be enumerated. If a font is used several times, it will be enumerated more than once!

If either ENUMMODE_USED_BLOCKS and/or ENUMMODE_USED_FONTS is set, the flags ENUMMODE_PLAIN and ENUMMODE_STRUCTURED will be ignored, i.e. no objects or instances will be passed.

Flags for Unit Type Limitation

These flags tell the application to ignore special types of units. In many cases, it is not useful to have fonts resolved and passed during enumeration, especially if only basic geometrical information is required like during hatching or surface generation.

ENUMMODE_NO_OBJECTS

If this flag is set, the procedure will *not* enumerate objects, i.e. only instances and referenced blocks / instances will be enumerated.

ENUMMODE_NO_BLOCKS

If this flag is set, the procedure will *not* enumerate instances and referenced blocks, i.e. only objects will be enumerated.

ENUMMODE_NO_FONTS

If this flag is set, the procedure will *not* enumerate characters of any type. This applies to characters in text objects as well as in dimensions.

Flags for Property Handling

These flags tell the application which property set to use. As pens define two sets of properties (one for screen display and one for printer output), the resulting image depends on which property set is used. In addition, the serving application features several settings that influence the properties (e.g. layers transmitting properties) and the kind of units used (e.g. whether to display geometry or not).

ENUMMODE_SCREEN_PROP

If this flag is set, the procedure will use the application's current settings for screen display. This will determine whether geometry objects will be visible, and whether layers will transmit properties or not.

ENUMMODE_OUTPUT_PROP

If this flag is set, the procedure will use the application's current settings for printer output. This will determine whether geometry objects will be visible, and whether layers will transmit properties or not.

If neither ENUMMODE_SCREEN_PROP nor ENUMMODE_OUTPUT_PROP is set, the printer output settings will be used.

CallBack

[*TOSOENUMOBJECT_PROC*] Callback procedure for an object enumeration. This procedure will be called for every enumeration step. For a detailed description of object enumeration, see also the description of the TosoEnumObjectProc procedure.

Contents

Return Value

Returns TRUE if the enumeration was finished successfully, or FALSE if the TosoEnumObjectProc procedure returned FALSE at its last call.

Contents **Comment**

For a detailed description of block enumeration, see the description of the corresponding callback procedure TosoEnumObjectProc.

Contents

Related Topics:

Contents TosoEnumerateAll

Contents TosoEnumerateLibrary

Contents TosoEnumerateUnit

Contents TosoEnumerateChar

Contents TosoEnumerateInstanceAttrib

Contents TosoEnumerateBlockAttrib

Contents TosoEnumerateIdent

Contents TosoEnumeratePoints

TosoEnumerateChar (Unit Enumeration)

Contents Syntax

```
BOOL TosoEnumerateChar( int DrawingNum,
                        const FONTDEF* Font,
                        int CharIndex,
                        const MATRIX* Matrix,
                        int EnumMode,
                        TOSOENUMOBJECT_PROC CallBack );
```

Enumerates a single characters.

Contents Parameters

DrawingNum

[*int*] Zero-based index of the drawing whose units shall be enumerated. Valid range: 0 <= Value < TVG_DRAWING_MAX.

Font

[*const FONTDEF**] Font definition of the font that shall be used to enumerate the character.

CharIndex

[*int*] Number of the character to be enumerated. Valid range: 32 <= Value <= 255.

Matrix

[*const MATRIX**] Matrix with which the character data shall be multiplied before enumeration.

EnumMode

[*int*] This values determines the way that units are enumerates. It indicates what types of geometrical data the module can handle, and what type of units shall be enumerated. It can be a bitwise OR combination of the following flags:

Flags for Geometrical Data Type

These flags tell the application what kind of geometrical data the module is able to handle. All other geometrical data will be converted to those types.

If, e.g., the module cannot handle circular arcs but Bézier curves, all circles and circular arcs will be converted to a sequence of Bézier curves.

ENUMMODE_LINES

The module can only handle solid polylines with line color and line width, i.e. line patterns will be resolved to single line segments. This is the basic mode which can be combined with one or more of the following modes to allow more complex geometric data.

ENUMMODE_FILLS

The module can handle filled, nested polygons.

ENUMMODE_ARCS

The module can handle circular arcs in polylines / polygons.

ENUMMODE_BEZIERS

The module can handle Bézier curves in polylines / polygons.

ENUMMODE_LINETYPES

The module can handle line patterns.

ENUMMODE_SOLID_CHARS

The module can handle solid characters, i.e. characters that are filled and non-distorted and can thus be displayed using standard system fonts. Appropriate texts will not be resolved into

polylines / polygons, but passed to the module as single characters.
Can be combined with ENUMMODE_OTHER_CHARS to allow the passing of all types of characters.

ENUMMODE_OTHER_CHARS
The module can handle non-solid characters, i.e. characters that are framed or distorted and can thus not be displayed using standard system fonts. Appropriate texts will not be resolved into polylines / polygons, but passed to the module as single characters.
Can be combined with ENUMMODE_SOLID_CHARS to allow the passing of all types of characters.

ENUMMODE_SOLID_TEXTS
The module can handle null-terminated strings with format information being filled and non-distorted and can thus be displayed using standard system fonts. Appropriate texts will not be resolved into polylines / polygons, but passed to the module as complete text lines.
Can be combined with ENUMMODE_OTHER_TEXTS to allow the passing of all types of texts.

ENUMMODE_OTHER_TEXTS
The module can handle null-terminated strings with format information being framed or distorted and can thus not be displayed using standard system fonts. Appropriate texts will not be resolved into polylines / polygons, but passed to the module as complete text lines.
Can be combined with ENUMMODE_SOLID_TEXTS to allow the passing of all types of texts.

ENUMMODE_MATRIX
The module can handle 3×2 matrices, i.e. it is able to apply them to *all* types of data. In this case, the serving application will pass the display matrix directly to the module instead of applying it to each unit.

ENUMMODE_NATIVE
The module can handle native object data, i.e. only the addresses of the units will be passed.

Flags for Handling of Blocks and Instances
These flags tell the application how to handle instances and blocks. If required, the application recurses through the calling tree of instances and blocks and resolves it.

ENUMMODE_PLAIN
If this flag is set, all blocks and instances will be resolved before passing data to the module, i.e. no instances will be passed.
If neither ENUMMODE_PLAIN nor ENUMMODE_STRUCTURED is set, no object enumeration will be performed. In this case, only ENUMMODE_USED_BLOCKS and/or ENUMMODE_USED_FONTS will be recognized.

ENUMMODE_STRUCTURED
If this flag is set, instances will directly be passed to the module. It's the module's duty to retrieve the correpsonding block definition if required.
Usually, a module being able to handle instances and blocks will first make an enumeration by means of TosoEnumerateAll with ENUMMODE_USED_BLOCKS set in order to get information about the blocks used. Then it will enumerate all blocks reported manually by means of TosoEnumerateBlock. Finally, it enumerates the objects and instances by calling TosoEnumerateAll again with ENUMMODE_STRUCTURED set.

If neither ENUMMODE_PLAIN nor ENUMMODE_STRUCTURED is set, no object enumeration will be performed. In this case, only ENUMMODE_USED_BLOCKS and/or ENUMMODE_USED_FONTS will be recognized.

Flags for Retrieving Information

These flags tell the application to return information about blocks and fonts used in the units. This can be used to determines what blocks have to be exported or whether all fonts used are available.

ENUMMODE_USED_BLOCKS
: If this flag is set, the procedure will list all block references found. This can be used by the module to find out what blocks are used in the units to be enumerated. If a block is used several times, it will be enumerated more than once!

ENUMMODE_USED_FONTS
: If this flag is set, the procedure will list all font references found. This can be used by the module to find out what fonts are used in the units to be enumerated. If a font is used several times, it will be enumerated more than once!

If either ENUMMODE_USED_BLOCKS and/or ENUMMODE_USED_FONTS is set, the flags ENUMMODE_PLAIN and ENUMMODE_STRUCTURED will be ignored, i.e. no objects or instances will be passed.

Flags for Unit Type Limitation

These flags tell the application to ignore special types of units. In many cases, it is not useful to have fonts resolved and passed during enumeration, especially if only basic geometrical information is required like during hatching or surface generation.

ENUMMODE_NO_OBJECTS
: If this flag is set, the procedure will *not* enumerate objects, i.e. only instances and referenced blocks / instances will be enumerated.

ENUMMODE_NO_BLOCKS
: If this flag is set, the procedure will *not* enumerate instances and referenced blocks, i.e. only objects will be enumerated.

ENUMMODE_NO_FONTS
: If this flag is set, the procedure will *not* enumerate characters of any type. This applies to characters in text objects as well as in dimensions.

Flags for Property Handling

These flags tell the application which property set to use. As pens define two sets of properties (one for screen display and one for printer output), the resulting image depends on which property set is used. In addition, the serving application features several settings that influence the properties (e.g. layers transmitting properties) and the kind of units used (e.g. whether to display geometry or not).

ENUMMODE_SCREEN_PROP
: If this flag is set, the procedure will use the application's current settings for screen display. This will determine whether geometry objects will be visible, and whether layers will transmit properties or not.

ENUMMODE_OUTPUT_PROP
: If this flag is set, the procedure will use the application's current settings for printer output. This will determine whether geometry objects will be visible, and whether layers will transmit properties or not.

If neither ENUMMODE_SCREEN_PROP nor ENUMMODE_OUTPUT_PROP is set, the printer output settings will be used.

CallBack

[*TOSOENUMOBJECT_PROC*] Callback procedure for an object enumeration. This procedure will be called for every enumeration step. For a detailed description of object enumeration, see also the description of the TosoEnumObjectProc procedure.

Return Value

Returns TRUE if the enumeration was finished successfully, or FALSE if the TosoEnumObjectProc procedure returned FALSE at its last call.

Comment

For a detailed description of character enumeration, see the description of the corresponding callback procedure TosoEnumObjectProc.

Currently, character enumeration will only work with internal (FONTTYPE_INTERN) and TrueType (FONTTYPE_TRUETYPE) fonts. Device fonts (FONTTYPE_DEVICE) cannot be resolved into curves and thus cannot be enumerated.

Related Topics:

TosoEnumerateAll

TosoEnumerateLibrary

TosoEnumerateUnit

TosoEnumerateBlock

TosoEnumerateInstanceAttrib

TosoEnumerateBlockAttrib

TosoEnumerateIdent

TosoEnumeratePoints

TosoEnumerateInstanceAttrib (Unit Enumeration)

Syntax

```
int TosoEnumerateInstanceAttrib( int DrawingNum,
                                 const UNIT_INSTANCE_PTR InstObj,
                                 TOSOENUMATTRIB_PROC CallBack );
```

Enumerates all attributes of the given instance.

Parameters

DrawingNum

[*int*] Zero-based index of the drawing whose units shall be enumerated. Valid range: 0 <= Value < TVG_DRAWING_MAX.

InstObj

[*const UNIT_INSTANCE_PTR*] Address of the instance whose attributes shall be enumerated.

CallBack

[*TOSOENUMATTRIB_PROC*] Callback procedure for an attribute enumeration. This procedure will be called for every enumeration step. For a detailed description of attribute enumeration, see also the description of the TosoEnumAttribProc procedure.

If *CallBack* is zero, the attributes will not be enumerated, but the attribute number will still be returned. This can be used to determine whether an instance has attributes (and how many).

Return Value

Returns the number of attributes enumerated, or -1 if the TosoEnumAttribProc procedure returned FALSE at its last call.

Comment

For a detailed description of attribute enumeration, see the description of the corresponding callback procedure TosoEnumAttribProc.

Contents

Related Topics:

TosoEnumerateAll

TosoEnumerateLibrary

TosoEnumerateUnit

TosoEnumerateBlock

TosoEnumerateChar

TosoEnumerateBlockAttrib

TosoEnumerateIdent

TosoEnumeratePoints

TosoEnumerateBlockAttrib (Unit Enumeration)

Contents **Syntax**

```
int TosoEnumerateBlockAttrib( int DrawingNum,
                              const LPSTR BlockName,
                              const LPSTR LibraryName,
                              TOSOENUMATTRIB_PROC CallBack );
```

Enumerates all attributes of the given block.

Contents **Parameters**

DrawingNum

[*int*] Zero-based index of the drawing whose units shall be enumerated (ignored if *LibraryName* is not equal to TVG_BLOCK_ID). Valid range: 0 <= Value < TVG_DRAWING_MAX.

BlockName

[*const LPSTR*] Name of the block whose attributes shall be enumerated.

LibraryName

[*const LPSTR*] Name of the library the block is located in.

CallBack

[*TOSOENUMATTRIB_PROC*] Callback procedure for an attribute enumeration. This procedure will be called for every enumeration step. For a detailed description of attribute enumeration, see also the description of the TosoEnumAttribProc procedure.

If *CallBack* is zero, the attributes will not be enumerated, but the attribute number will still be returned. This can be used to determine whether a block has attributes (and how many).

Contents **Return Value**

Returns the number of attributes enumerated, or -1 if the TosoEnumAttribProc procedure returned FALSE at its last call.

Contents **Comment**

For a detailed description of attribute enumeration, see the description of the corresponding callback procedure TosoEnumAttribProc.

Contents

Related Topics:

Contents TosoEnumerateAll

Contents TosoEnumerateLibrary

Contents TosoEnumerateUnit

TosoEnumerateBlock

TosoEnumerateChar

TosoEnumerateInstanceAttrib

TosoEnumerateIdent

TosoEnumeratePoints

TosoEnumerateIdent (Unit Enumeration)

Syntax

```
BOOL TosoEnumerateIdent( int DrawingNum,
                         TOSOENUMIDENT_PROC CallBack );
```

Enumerates the addresses of all identified objects (only valid for multi-object-identification).

Parameters

DrawingNum

[*int*] Zero-based index of the drawing whose units shall be enumerated. Valid range: 0 <= Value < TVG_DRAWING_MAX.

CallBack

[*TOSOENUMIDENT_PROC*] Callback procedure for an identification enumeration. This procedure will be called for every enumeration step. For a detailed description of identification enumeration, see also the description of the TosoEnumIdentProc procedure.

Return Value

Returns TRUE if the enumeration was finished successfully, or FALSE if the TosoEnumIdent procedure returned FALSE at its last call.

Comment

For a detailed description of identification enumeration, see the description of the corresponding callback procedure TosoEnumIdentProc.

Related Topics:

TosoEnumerateAll

TosoEnumerateLibrary

TosoEnumerateUnit

TosoEnumerateBlock

TosoEnumerateChar

TosoEnumeratePoints (Unit Enumeration)

Syntax

```
BOOL TosoEnumerateIdent( int DrawingNum,
                         TOSOENUMPOINTS_PROC CallBack );
```

Enumerates the addresses of all identified points and their parent objects (only valid for multi-point-identification).

Parameters

DrawingNum

[*int*] Zero-based index of the drawing whose points shall be enumerated. Valid range: 0 <= Value < TVG_DRAWING_MAX.

CallBack

[*TOSOENUMPOINTS_PROC*] Callback procedure for an identification enumeration. This procedure will be called for every enumeration step. For a detailed description of identification enumeration, see also the description of the TosoEnumPointsProc procedure.

Return Value

Returns TRUE if the enumeration was finished successfully, or FALSE if the TosoEnumPoints procedure returned FALSE at its last call.

Comment

For a detailed description of identification enumeration, see the description of the corresponding callback procedure TosoEnumPointsProc.

Related Topics:

TosoEnumerateAll

TosoEnumerateLibrary

TosoEnumerateUnit

TosoEnumerateBlock

TosoEnumerateChar

TosoEnumerateInstanceAttrib

TosoEnumerateBlockAttrib

TosoEnumerateIdent

TosoDrawingNewFile (File Handling)

Syntax

```
BOOL TosoDrawingNewFile( int DrawingNum,
                         BOOL Prompt );
```

Deletes a drawing and creates an empty, untitled one.

Parameters

DrawingNum

[*int*] Zero-based index of the drawing that is to be deleted. This can either be a currently used drawing (which will then be deleted) or the return value of TosoDrawingGetNumber if a totally new drawing is to be added. Valid range: 0 <= Value < TVG_DRAWING_MAX.

Prompt

[*BOOL*] If this value is TRUE, the function checks whether the drawing to be deleted has changed since last saving. If so, a dialog appears allowing the user to save the "old" drawing first.

Return Value

Returns TRUE if successful, else FALSE. If FALSE is returned, the "old" drawing was changed and the user did not allow it to be removed (with or without saving).

Comment

None.

Related Topics:

TosoDrawingOpenFile

TosoDrawingSaveFile

TosoDrawingOpenFile (File Handling)

Contents **Syntax**

```
BOOL TosoDrawingOpenFile( int DrawingNum,
                          const LPSTR FileName,
                          LONG OpenFlags,
                          BOOL Prompt );
```

Removes the current drawing from memory and loads another drawing file from disk.

Contents **Parameters**

DrawingNum

[*int*] Zero-based index of the drawing that is to be opened. This can either be a currently used drawing (which will then replaced) or the return value of TosoDrawingGetNumber if a totally new drawing is to be loaded. Valid range: 0 <= Value < TVG_DRAWING_MAX.

FileName

[*const LPSTR*] Filename of the drawing file to be loaded.

OpenFlags

[*long*] Bitwise OR combination of flags that determine which sections of the drawing are to be loaded. Possible values are:

LOADSAVE_HEADER
: If this flag is set, the file header is loaded (should normally be set unless LOADSAVE_USERSECTIONS is set!).

LOADSAVE_TOOLBOX
: If this flag is set, the toolbox settings are loaded.

LOADSAVE_BLOCKLIST
: If this flag is set, the block list settings are loaded.

LOADSAVE_KEYBOARD
: If this flag is set, the keyboard settings are loaded.

LOADSAVE_WINDOW
: If this flag is set, the window settings are loaded.

LOADSAVE_DEFAULT
: If this flag is set, the default assignment of pens and layers are loaded.

LOADSAVE_USER
: If this flag is set, the user settings are loaded.

LOADSAVE_MODULE
: If this flag is set, the module-dependent are loaded.

LOADSAVE_PAGE
: If this flag is set, the page settings are loaded.

LOADSAVE_COLOR
: If this flag is set, the custom-color definitions are loaded.

LOADSAVE_HATCH
: If this flag is set, the hatching definitions are loaded.

LOADSAVE_MULTILINE
If this flag is set, the line sequence definitions are loaded.

LOADSAVE_SYSTEM
If this flag is set, the coordinate system definitions are loaded.

LOADSAVE_PEN
If this flag is set, the pen definitions are loaded.

LOADSAVE_LINE
If this flag is set, the line pattern definitions are loaded.

LOADSAVE_LAYER
If this flag is set, the layer definitions are loaded.

LOADSAVE_ATTRIB
If this flag is set, the standard attributes are loaded (only valid for library files).

LOADSAVE_BLOCK
If this flag is set, the blocks are loaded.

LOADSAVE_OBJECT
If this flag is set, the objects are loaded.

LOADSAVE_USERSECTIONS
If this flag is set, the current default flags are added to all other flags set. This allows the user to determine which sections to load.

LOADSAVE_IDENTIFIED
This flag is not used when opening files.

LOADSAVE_REPLACE
If this flag is set, all elements loaded (pen, layers, linepattern, coordinate system, etc.) replace the currently available ones. This flag should always be set if not only specific elements are to be loaded. When loading a "normal" drawing with this flag cleared can lead to fascinating results...

LOADSAVE_NODIALOG
If this flag is set, no progress indicator is display during the file operation.

Normally, this flag should be set to LOADSAVE_USERSECTIONS only, thus using the current default settings. If not altered by the user, these default settings include *all* sections.

Prompt
[*BOOL*] If this value is TRUE, the function checks whether the drawing to be replaced has changed since last saving. If so, a dialog appears allowing the user to save the "old" drawing first.

Return Value

Returns TRUE if successful, else FALSE. If FALSE is returned, the drawing file could not be loaded successfully, either because it did not exist, or it was faulty, or the user did not allow the "old" drawing to be replaced (with or without saving).

Comment

None.

Related Topics:

TosoDrawingNewFile

TosoDrawingSaveFile

TosoDrawingSaveFile (File Handling)

Contents Syntax

```
BOOL TosoDrawingSaveFile( int DrawingNum,
                          const LPSTR FileName,
                          LONG SaveFlags,
                          BOOL Prompt );
```

Saves the current drawing to disk.

Contents Parameters

DrawingNum

[*int*] Zero-based index of the drawing that is to be saved. Valid range: 0 <= Value < TVG_DRAWING_MAX.

FileName

[*const LPSTR*] Filename of the drawing file to be created. If *FileName* is NULL, the current drawing's filename is used. This will fail if the drawing is still untitled.

OpenFlags

[*long*] Bitwise OR combination of flags that determine which sections of the drawing are to be saved. Possible values are:

LOADSAVE_HEADER
: If this flag is set, the file header is saved (should normally be set unless LOADSAVE_USERSECTIONS is set!).

LOADSAVE_TOOLBOX
: If this flag is set, the toolbox settings are saved.

LOADSAVE_BLOCKLIST
: If this flag is set, the block list settings are saved.

LOADSAVE_KEYBOARD
: If this flag is set, the keyboard settings are saved.

LOADSAVE_WINDOW
: If this flag is set, the window settings are saved.

LOADSAVE_DEFAULT
: If this flag is set, the default assignment of pens and layers are saved.

LOADSAVE_USER
: If this flag is set, the user settings are saved.

LOADSAVE_MODULE
: If this flag is set, the module-dependent are saved.

LOADSAVE_PAGE
: If this flag is set, the page settings are saved.

LOADSAVE_COLOR
: If this flag is set, the custom-color definitions are saved.

LOADSAVE_HATCH
: If this flag is set, the hatching definitions are saved.

LOADSAVE_MULTILINE
If this flag is set, the line sequence definitions are saved.

LOADSAVE_SYSTEM
If this flag is set, the coordinate system definitions are saved.

LOADSAVE_PEN
If this flag is set, the pen definitions are saved.

LOADSAVE_LINE
If this flag is set, the line pattern definitions are saved.

LOADSAVE_LAYER
If this flag is set, the layer definitions are saved.

LOADSAVE_ATTRIB
If this flag is set, the standard attributes are saved (only valid for library files).

LOADSAVE_BLOCK
If this flag is set, the blocks are saved.

LOADSAVE_OBJECT
If this flag is set, the objects are saved.

LOADSAVE_USERSECTIONS
If this flag is set, the current default flags are added to all other flags set. This allows the user to determine which sections to save. In addition, it assures that *all* sections that have previously been loaded from disk will be saved.

LOADSAVE_IDENTIFIED
If this flag is set, only identified objects are saved. The saving of other sections is not influenced.

LOADSAVE_REPLACE
This flag is not used when saving files.

LOADSAVE_NODIALOG
If this flag is set, no progress indicator is display during the file operation.

Normally, this flag should be set to LOADSAVE_USERSECTIONS only, thus using the current default settings. These default settings include *all* sections that have previously been loaded.

Prompt
[*BOOL*] If this value is TRUE, the function checks whether the drawing file passed in *FileName* already exists. If so, a dialog appears allowing the user to cancel.

Return Value

Returns TRUE if successful, else FALSE. If FALSE is returned, the file has not been saved successfully, either because the disk was full, or the filename was invalid, or the user did not allow to overwrite an already existing file.

Comment

If the drawing was untitled before, the filename stated in *FileName* will be assigned to the drawing.

Related Topics:

TosoDrawingNewFile

TosoDrawingOpenFile

TosoFileSetExtension (File Handling)

Syntax

```
BOOL TosoFileSetExtension( const LPSTR OldName,
                           LPSTR NewName,
                           const LPSTR Extension );
```

Sets a file name's extension to a given extension.

Parameters

OldName

[*const LPSTR*] Address of file name to be modified. This file name may have an extension.

NewName

[*LPSTR*] Address of text buffer to receive the modified file name. Allow at least MAX_PATH characters.

Extension

[*const LPSTR*] Extension to be added to the given file name, with or without a leading point, e.g. "EXT" or ".EXT" . If *Extension* is either NULL or points to an empty string, the current file name's extension will be removed without adding a new extension.

Return Value

Returns TRUE if successful, else FALSE.

Comment

This procedure will work for all types of file systems (FAT, Enhanced FAT, NTFS, HPFS).

Related Topics:

TosoFileGetExtension

TosoFileShortName

TosoFileSplitName

TosoFileFullPath

TosoFileGetExtension (File Handling)

Syntax

```
BOOL TosoFileGetExtension( const LPSTR FullName,
                           LPSTR Extension );
```

Retrieves a file name's extension.

Parameters

FullName

[*const LPSTR*] Address of file name whose extension shall be returned.

Extension

[*LPSTR*] Address of text buffer to receive the modified file name. Allow at least MAX_PATH characters.

Return Value

Returns TRUE if successful, else FALSE.

Comment

This procedure will work for all types of file systems (FAT, Enhanced FAT, NTFS, HPFS).

Related Topics:

TosoFileSetExtension

TosoFileShortName

TosoFileSplitName

TosoFileFullPath

TosoFileShortName (File Handling)

Syntax

```
BOOL TosoFileShortName( const LPSTR FullName,
                        LPSTR ShortName );
```

Shortens a file's name, usually to display it in a dialog window of limited size. Usually, the resulting file name will not be longer than 60 characters (unless the file's base name itself is longer than 60 characters).

Parameters

FullName

[*const LPSTR*] Address of file name to be shortened.

ShortName

[*LPSTR*] Address of text buffer to receive the shortened file name. Allow at least MAX_PATH characters.

Return Value

Returns TRUE if successful, else FALSE.

Comment

If, e.g., the current filename is:

```
d:\dirname1\dirname2\dirname3\dirname4\filename.ext
```

the resulting short file name might be:

```
d:\...\dirname4\filename.ext
```

This procedure will work for all types of file systems (FAT, Enhanced FAT, NTFS, HPFS).

Related Topics:

TosoFileSetExtension

TosoFileGetExtension

TosoFileSplitName

TosoFileFullPath

TosoFileSplitName (File Handling)

Syntax

```
BOOL TosoFileSplitName( const LPSTR FullName,
                        LPSTR Path,
                        LPSTR Name );
```

Separates a file's name into its path and file components.

Parameters

FullName

[*const LPSTR*] Address of file name to be split.

Path

[*LPSTR*] Address of text buffer to receive the path component of the file name. Allow at least MAX_PATH characters. If *Path* is NULL, this component will be ignored.

Name

[*LPSTR*] Address of text buffer to receive the file component of the file name. Allow at least MAX_PATH characters. If *Name* is NULL, this component will be ignored.

Return Value

Returns TRUE if successful, else FALSE.

Comment

This procedure will work for all types of file systems (FAT, Enhanced FAT, NTFS, HPFS).

Related Topics:

TosoFileSetExtension

TosoFileGetExtension

TosoFileShortName

TosoFileFullPath

TosoFileFullPath (File Handling)

Syntax

```
BOOL TosoFileFullPath( const LPSTR OldName,
                       LPSTR NewName );
```

Supplies a "simple" file name with a complete path and a drive letter. For this purpose, all pathes stated in the system's settings will be searched, plus the path of the serving application.

Parameters

OldName

[*const LPSTR*] Address of incomplete file name.

NewName

[*LPSTR*] Address of text buffer to receive the complete file name. Allow at least MAX_PATH characters.

Return Value

Returns TRUE if successful, else FALSE.

Comment

This procedure will work for all types of file systems (FAT, Enhanced FAT, NTFS, HPFS).

Related Topics:

TosoFileSetExtension

TosoFileGetExtension

TosoFileShortName

TosoFileSplitName

TosoFileExist (File Handling)

Contents Syntax

```
BOOL TosoFileExist( const LPSTR FileName );
```

Checks whether the given file exists or not.

Contents Parameters

FileName

[*const LPSTR*] Address of file name.

Contents Return Value

Returns TRUE if the file exists, else FALSE (indicating that either the file does not exist or the current user has no reading access to that file).

Contents Comment

None.

Contents

Related Topics:

Contents TosoFileDelete

Contents TosoFileOpen

Contents TosoFileCreate

Contents TosoFileSize

Contents TosoFileRead

Contents TosoFileWrite

Contents TosoFileSetPointer

Contents TosoFileClose

Contents

TosoFileDelete (File Handling)

Syntax

```
BOOL TosoFileDelete( const LPSTR FileName );
```

Deletes the given file.

Parameters

FileName

[*const LPSTR*] Address of file name.

Return Value

Returns TRUE if the file has been deleted successfully, else FALSE (indicating that either the file does not exist or the current user has no deleting access to that file).

Comment

None.

Related Topics:

TosoFileExist

TosoFileOpen

TosoFileCreate

TosoFileSize

TosoFileRead

TosoFileWrite

TosoFileSetPointer

TosoFileClose

Contents

TosoFileOpen (File Handling)

Syntax

```
BOOL TosoFileOpen( HANDLE* FileHandle,
                   const LPSTR FileName );
```

Opens an existing file for read-only access.

Parameters

FileHandle
[*HANDLE**] Address of file handle to receive the opened file's handle.
FileName
[*const LPSTR*] Address of file name.

Return Value

Returns TRUE if the file has been opened successfully, else FALSE (indicating that either the file does not exist or the current user has no reading access to that file).

Comment

None.

Related Topics:

- TosoFileExist
- TosoFileDelete
- TosoFileCreate
- TosoFileSize
- TosoFileRead
- TosoFileWrite
- TosoFileSetPointer

TosoFileCreate (File Handling)

Contents Syntax

```
BOOL TosoFileCreate( HANDLE* FileHandle,
                     const LPSTR FileName );
```

Creates a file for write-only access. If a file of the same name does already exist, it will be deleted.

Contents Parameters

FileHandle
[*HANDLE**] Address of file handle to receive the created file's handle.

FileName
[*const LPSTR*] Address of file name.

Contents Return Value

Returns TRUE if the file has been created successfully, else FALSE (indicating that either the drive is full, write-protected, or the current user has no writing access to that drive).

Contents Comment

None.

Contents

Related Topics:

Contents TosoFileExist

Contents TosoFileDelete

Contents TosoFileOpen

Contents TosoFileSize

Contents TosoFileRead

Contents TosoFileWrite

Contents TosoFileSetPointer

TosoFileSize (File Handling)

Contents

Syntax

```
long TosoFileSize( HANDLE FileHandle );
```

Retrieves the size of a previously opened file.

Contents

Parameters

FileHandle

[*HANDLE*] File handle of the file whose size is to be retrieved.

Contents

Return Value

Returns the size of the given file in bytes.

Contents

Comment

None.

Contents

Related Topics:

Contents TosoFileExist
Contents TosoFileDelete
Contents TosoFileOpen
Contents TosoFileCreate
Contents TosoFileRead
Contents TosoFileWrite
Contents TosoFileSetPointer
Contents TosoFileClose

Contents

TosoFileRead (File Handling)

Syntax

```
long TosoFileRead( HANDLE FileHandle,
                   LPVOID Data,
                   long Size );
```

Parameters

FileHandle

[*HANDLE*] File handle of a file open for read access.

Data

[*LPVOID*] Address of a buffer to receive the data read from the file.

Size

[*long*] Number of bytes to be read from the file.

Return Value

Returns the number of bytes read from the file. If this value is less than *Size*, the file did contain less data than requested.

Comment

This procedure does *not* support the additional disk write cache, i.e. it may not be used after calling either TosoFileReadInitDisk or TosoFileReadInitMemory!!!

Related Topics:

- TosoFileExist
- TosoFileDelete
- TosoFileOpen
- TosoFileCreate
- TosoFileSize
- TosoFileWrite

TosoFileWrite (File Handling)

Contents **Syntax**

```
long TosoFileWrite( HANDLE FileHandle,
                    const LPVOID Data,
                    long Size );
```

Contents **Parameters**

FileHandle

[*HANDLE*] File handle of a file open for write access.

Data

[*const LPVOID*] Address of a buffer containing the data to be written to the file.

Size

[*long*] Number of bytes to be written to the file.

Contents **Return Value**

Returns the number of bytes written to the file. If this value is less than *Size*, an error has occured (usually indicating that the drive is full).

Contents **Comment**

This procedure does *not* support the additional disk write cache, i.e. it may not be used after calling either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory!!!

Contents

Related Topics:

Contents TosoFileExist

Contents TosoFileDelete

Contents TosoFileOpen

Contents TosoFileCreate

Contents TosoFileSize

Contents TosoFileRead

TosoFileSetPointer (File Handling)

Contents

Syntax

```
BOOL TosoFileSetPointer( HANDLE FileHandle,
                         long Offset,
                         DWORD Mode );
```

Moves the file pointer of the given file, i.e. the current position for reading or writing inside that file.

Contents

Parameters

FileHandle

[*HANDLE*] File handle of a previously opened file whose file pointer is to be moved.

Offset

[*long*] Relative movement of the file pointer in bytes.

Mode

[*DWORD*] Specifies the starting point for the file pointer move. This parameter can be one of the following values:

FILE_BEGIN

The starting point is zero or the beginning of the file. If FILE_BEGIN is specified, *Offset* is interpreted as an unsigned location for the new file pointer.

FILE_CURRENT

The current value of the file pointer is the starting point.

FILE_END

The current end-of-file position is the starting point.

Contents

Return Value

Returns TRUE if the file pointer has been moved successfully, else FALSE (indicating that the given file pointer position is invalid).

Contents

Comment

This procedure does *not* support the additional disk read and write cache, i.e. it may not be used after calling either TosoFileReadInitDisk or TosoFileReadInitMemory (for read-only files), or TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory (for write-only files) !!!

Contents

Related Topics:

Contents TosoFileExist

Contents TosoFileDelete

TosoFileClose (File Handling)

Contents **Syntax**

```
BOOL TosoFileClose( HANDLE FileHandle );
```

Contents **Parameters**

FileHandle

[*HANDLE*] File handle of the file to be closed.

Contents **Return Value**

Returns TRUE if the file has been closed successfully, else FALSE.

Contents **Comment**

None.

Contents

Related Topics:

Contents TosoFileExist

Contents TosoFileDelete

Contents TosoFileOpen

Contents TosoFileCreate

Contents TosoFileSize

Contents TosoFileRead

Contents TosoFileWrite

Contents TosoFileSetPointer

Contents TosoFileCopy

TosoFileCopy (File Handling)

Syntax

```
BOOL TosoFileCopy( const LPSTR SourceName,
                   const LPSTR DestinName );
```

Copies one file into another file.

Parameters

SourceName
[*const LPSTR*] Address of source file's name.

DestinName
[*const LPSTR*] Address of destination file's name.

Return Value

Returns TRUE if the file has been copied successfully, else FALSE (indicating that either the destination drive is full, write-protected, or the current user has no writing access to that drive).

Comment

None.

Related Topics:

TosoFileExist

TosoFileDelete

TosoFileOpen

TosoFileCreate

TosoFileSize

TosoFileRead

TosoFileWrite

Contents TosoFileSetPointer
Contents TosoFileClose

TosoFileReadInitDisk (File Handling)

Syntax

```
BOOL TosoFileReadInitDisk( HANDLE FileHandle );
```

Prepares reading from a disk file using an additional, high-performance read cache. This cache will automatically be used by all TosoFileRead* procedures.

Parameters

FileHandle

[*HANDLE*] File handle of the file previously opened for read access.

Return Value

Returns TRUE if the cache has been established successfully, else FALSE (indicating insufficient memory).

Comment

None.

Related Topics:

File Handling >

TosoFileReadInitMemory (File Handling)

Contents Syntax

```
BOOL TosoFileReadInitMemory( HGLOBAL hMemory );
```

Prepares reading from a memory image using an additional, high-performance read cache. This cache will automatically be used by all TosoFileRead* procedures.

Contents Parameters

hMemory

[*HGLOBAL*] Handle of allocated global memory. This memory must have been allocated using the GMEM_MOVEABLE attribute. This handle may also be a clipboard object handle returned by the GetClipboardData procedure of Win32.

Contents Return Value

Returns TRUE if the cache has been established successfully, else FALSE (indicating insufficient memory).

Contents Comment

None.

Contents

Related Topics:

Contents File Handling >

TosoFileReadData (File Handling)

Contents

Syntax

```
BOOL TosoFileReadData( LPVOID Data,
                       int Size );
```

Reads a given number of bytes from the current read cache.

Contents

Parameters

Data

[*LPVOID*] Address of a buffer to receive the data read from the file.

Size

[*int*] Number of bytes to be read from the file.

Contents

Return Value

Returns TRUE if the data has been read successfully, else FALSE (indicating end-of-file). Additionally, the return value of TosoFileReadError will be set to TRUE if an error occured.

Contents

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

Contents

Related Topics:

Contents File Handling >

TosoFileReadChar (File Handling)

Contents **Syntax**

```
int TosoFileReadChar( void );
```

Reads the next character from the current read cache.

Contents **Parameters**

None.

Contents **Return Value**

Returns the character read, or 0 if an error occured (indicating end-of-file). Additionally, the return value of TosoFileReadError will be set to TRUE if an error occured.

Contents **Comment**

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

Contents

Related Topics:

Contents File Handling >

TosoFileReadLine (File Handling)

Contents

Syntax

```
BOOL TosoFileReadLine( LPSTR Data,
                       int Size );
```

Reads from the current position of the read cache to the next end-of-line and copies the data read into a buffer.

Contents

Parameters

Data

[*LPSTR*] Address of a text buffer to receive the line read from the file. The line can either be ended by a CR (Ansi 13), a LF (Ansi 10) or a combination of both. This is *not* influenced by the *Newline* value of the current TosoFileReadDelimiters!

Size

[*int*] Size of the text buffer, i.e. maximum number of characters to read plus one.

Contents

Return Value

Returns TRUE if the text line has been read successfully, else FALSE (indicating end-of-file or line too long to fit into the buffer). Additionally, the return value of TosoFileReadError will be set to TRUE if an error occured.

Contents

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

Contents

Related Topics:

Contents File Handling >

TosoFileReadExit (File Handling)

Syntax

```
void TosoFileReadExit( void );
```

Frees the high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory.

Parameters

None.

Return Value

None.

Comment

None.

Related Topics:

File Handling >

TosoFileReadSemi (File Handling)

Contents

Syntax

```
void TosoFileReadSemi( void );
```

Tries to "read" a semicolon from the current read cache. In fact, the procedure does only check whether the last delimiter read was a semicolon.

Contents

Parameters

None.

Contents

Return Value

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Contents

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Contents

Related Topics:

Contents File Handling >

TosoFileReadComma (File Handling)

Syntax

```
void TosoFileReadComma( void );
```

Tries to "read" a comma from the current read cache. In fact, the procedure does only check whether the last delimiter read was a comma.

Parameters

None.

Return Value

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Related Topics:

File Handling >

TosoFileReadContinue (File Handling)

Syntax

```
BOOL TosoFileReadContinue( void );
```

Checks whether the delimiter of the previously read value was a semicolon or not. This procedure is used to determine whether a value has completely been read or not.

Parameters

None.

Return Value

Returns TRUE is the separator following the least recently read value is a comma and *not* a semicolon.

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Related Topics:

File Handling >

TosoFileReadLastKeyword (File Handling)

Contents Syntax

```
int TosoFileReadLastKeyword( void );
```

Retrieves the ID of the least recently read keyword (read by either calling TosoFileReadHeader, TosoFileReadKeyword, TosoFileReadNextKeyword or TosoFileReadNextSection).

Contents Parameters

None.

Contents Return Value

Returns the index of the least recently read keyword. Possible values are:

TVG_KEY_UNKNOWN
: The least recently read keyword is not known (but may still be valid). Ignore such a section by immediately calling TosoFileReadNextSection.

TVG_KEY_EXIT
: The least recently read keyword is `=EXIT=`.

TVG_KEY_END
: The least recently read keyword is `=END=`.

TVG_KEY_DRAWING
: The least recently read keyword is `=DRAWING=`.

TVG_KEY_LIBRARY
: The least recently read keyword is `=LIBRARY=`.

TVG_KEY_ATTRIB
: The least recently read keyword is `=ATTRIB=`.

TVG_KEY_PAGE
: The least recently read keyword is `=PAGE=`.

TVG_KEY_DEFAULT
: The least recently read keyword is `=DEFAULT=`.

TVG_KEY_ZOOM
: The least recently read keyword is `=ZOOM=`.

TVG_KEY_SYSTEM
: The least recently read keyword is `=SYSTEM=`.

TVG_KEY_PEN
: The least recently read keyword is `=PEN=`.

TVG_KEY_LINE
: The least recently read keyword is `=LINE=`.

TVG_KEY_LAYER

The least recently read keyword is =LAYER=.

TVG_KEY_HATCH
The least recently read keyword is =HATCH=.

TVG_KEY_BLOCK
The least recently read keyword is =BLOCK=.

TVG_KEY_OBJECT
The least recently read keyword is =OBJECT=.

TVG_KEY_COLOR
The least recently read keyword is =COLOR=.

TVG_KEY_KEYBOARD
The least recently read keyword is =KEYBOARD=.

TVG_KEY_TOOLBOX
The least recently read keyword is =TOOLBOX=.

TVG_KEY_SYMBOL
The least recently read keyword is =SYMBOL=.

TVG_KEY_USER
The least recently read keyword is =USER=.

TVG_KEY_MODULE
The least recently read keyword is =MODULE=.

For a detailed description of the keyword's usage, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

Related Topics:

File Handling >

TosoFileReadKeyword (File Handling)

Syntax

```
void TosoFileReadKeyword( void );
```

Tries to read a keyword from the read cache. To retrieve the index of the keyword read, use the TosoFileReadLastKeyword procedure.

Parameters

None.

Return Value

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Related Topics:

File Handling >

TosoFileReadNextKeyword (File Handling)

Contents

Syntax

```
void TosoFileReadNextKeyword( void );
```

Reads data from the read cache (ignoring all data found) until it finds the next keyword. To retrieve the index of the keyword read, use the TosoFileReadLastKeyword procedure.

Contents

Parameters

None.

Contents

Return Value

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Contents

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Contents

Related Topics:

Contents File Handling >

TosoFileReadNextSection (File Handling)

Syntax

```
void TosoFileReadNextSection( void );
```

Reads data from the read cache (ignoring all data found) until it finds the next section header keyword. To retrieve the index of the keyword read, use the TosoFileReadLastKeyword procedure.

Parameters

None.

Return Value

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Related Topics:

File Handling >

TosoFileReadString (File Handling)

Contents

Syntax

```
void TosoFileReadString( LPSTR Value,
                         int Size );
```

Tries to read a quoted string from the read cache.

Contents

Parameters

Value

[*LPSTR*] Address of a text buffer to receive the string read from the file. The string will be decoded before being copied to this buffer, i.e. it will no longer contain escape sequences.

Size

[*int*] Size of the text buffer, i.e. maximum number of characters to read plus one.

Contents

Return Value

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Contents

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Contents

Related Topics:

Contents File Handling >

TosoFileReadCommaString (File Handling)

Contents

Syntax

```
void TosoFileReadCommaString( LPSTR Value,
                              int Size );
```

Tries to read a comma followed by a quoted string from the read cache.

Contents

Parameters

Value

[*LPSTR*] Address of a text buffer to receive the string read from the file. The string will be decoded before being copied to this buffer, i.e. it will no longer contain escape sequences.

Size

[*int*] Size of the text buffer, i.e. maximum number of characters to read plus one.

Contents

Return Value

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Contents

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Contents

Related Topics:

Contents File Handling >

TosoFileReadBinary (File Handling)

Contents

Syntax

```
void TosoFileReadBinary( LPVOID Value,
                         int Size );
```

Tries to read quoted, encoded binary data from the read cache.

Contents

Parameters

Value

[*LPVOID*] Address of a buffer to receive the data read from the file. The binary data will be decoded before being copied to this buffer. If the binary string does contain less bytes than requested, the missing bytes will be set to zero.

Size

[*int*] Size of the buffer, i.e. maximum number of bytes to read.

Contents

Return Value

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Contents

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Contents

Related Topics:

Contents File Handling >

TosoFileReadCommaBinary (File Handling)

Contents

Syntax

```
void TosoFileReadCommaBinary( LPVOID Value,
                              int Size );
```

Tries to read a comma followed by encoded binary data from the read cache.

Contents

Parameters

Value

[*LPVOID*] Address of a buffer to receive the data read from the file. The binary data will be decoded before being copied to this buffer. If the binary string does contain less bytes than requested, the missing bytes will be set to zero.

Size

[*int*] Size of the buffer, i.e. maximum number of bytes to read.

Contents

Return Value

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Contents

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Contents

Related Topics:

Contents File Handling >

TosoFileReadBool (File Handling)

Syntax

```
void TosoFileReadBool( BOOL* Value );
```

Tries to read a boolean value from the read cache.

Parameters

Value

[*BOOL**] Address of boolean value to be read. The only valid values for an boolean are 0 and 1.

Return Value

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Related Topics:

File Handling >

TosoFileReadCommaBool (File Handling)

Syntax

```
void TosoFileReadCommaBool( BOOL* Value );
```

Tries to read a comma followed by a boolean value from the read cache.

Parameters

Value

[*BOOL**] Address of boolean value to be read. The only valid values for an boolean are 0 and 1.

Return Value

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Related Topics:

File Handling >

TosoFileReadShort (File Handling)

Syntax

```
void TosoFileReadShort( short* Value );
```

Tries to read a short integer value from the read cache.

Parameters

Value

[*short**] Address of short integer value to be read. The valid range is -32,766 to 32,767.

Return Value

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Related Topics:

File Handling >

TosoFileReadCommaShort (File Handling)

Syntax

```
void TosoFileReadCommaShort( short* Value );
```

Tries to read a comma followed by a short integer value from the read cache.

Parameters

Value

[*short**] Address of short integer value to be read. The valid range is -32,766 to 32,767.

Return Value

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Related Topics:

File Handling >

TosoFileReadInt (File Handling)

Contents

Syntax

```
void TosoFileReadInt( int* Value );
```

Tries to read an integer value from the read cache.

Contents

Parameters

Value

[*int**] Address of integer value to be read. The valid range is -2,147,483,646 to 2,147,483,647.

Contents

Return Value

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Contents

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Contents

Related Topics:

Contents File Handling >

TosoFileReadCommaInt (File Handling)

Syntax

```
void TosoFileReadCommaInt( int* Value );
```

Tries to read a comma followed by an integer value from the read cache.

Parameters

Value

[*int**] Address of integer value to be read. The valid range is -2,147,483,646 to 2,147,483,647.

Return Value

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Related Topics:

File Handling >

TosoFileReadDouble (File Handling)

Syntax

```
void TosoFileReadDouble( double* Value );
```

Tries to read a double precision floating point value from the read cache.

Parameters

Value

[*double**] Address of double precision floating point value to be read. The valid range is -1e100 to 1e100.

Return Value

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Related Topics:

File Handling >

TosoFileReadCommaDouble (File Handling)

Contents

Syntax

```
void TosoFileReadCommaDouble( double* Value );
```

Tries to read a comma followed by a double precision floating point value from the read cache.

Contents

Parameters

Value

[*double**] Address of double precision floating point value to be read. The valid range is -1e100 to 1e100.

Contents

Return Value

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Contents

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Contents

Related Topics:

Contents File Handling >

TosoFileReadColorref (File Handling)

Contents Syntax

```
void TosoFileReadColorref( COLORREF* Color );
```

Tries to read a color definition from the read cache.

Contents Parameters

Color

[*COLORREF**] Address of color definition to be read. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Contents Return Value

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Contents Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Contents

Related Topics:

Contents File Handling >

TosoFileReadCommaColorref (File Handling)

Syntax

```
void TosoFileReadCommaColorref( COLORREF* Color );
```

Tries to read a comma followed by a color definition from the read cache.

Parameters

Color

[*COLORREF**] Address of color definition to be read. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Return Value

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Related Topics:

File Handling >

TosoFileReadFontdef (File Handling)

Syntax

```
void TosoFileReadFontdef( FONTDEF* Font );
```

Tries to read a font definition from the read cache.

Parameters

Font

[*FONTDEF**] Address of font definition to be read. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Return Value

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Related Topics:

File Handling >

TosoFileReadCommaFontdef (File Handling)

Syntax

```
void TosoFileReadCommaFontdef( FONTDEF* Font );
```

Tries to read a comma followed by a font definition from the read cache.

Parameters

Font

[*FONTDEF**] Address of font definition to be read. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Return Value

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Related Topics:

File Handling >

TosoFileReadXProperty (File Handling)

Contents **Syntax**

```
void TosoFileReadXProperty( XPROPERTY* XProperty );
```

Tries to read an extended property set from the read cache.

Contents **Parameters**

XProperty

[*XPROPERTY**] Address of extended property set to be read. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Contents **Return Value**

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Contents **Comment**

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Contents

Related Topics:

Contents File Handling >

TosoFileReadCommaXProperty (File Handling)

Syntax

```
void TosoFileReadCommaXProperty( XPROPERTY* XProperty );
```

Tries to read a comma followed by an extended property set from the read cache.

Parameters

XProperty

[*XPROPERTY**] Address of extended property set to be read. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Return Value

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Related Topics:

File Handling >

TosoFileReadProperty (File Handling)

Syntax

```
void TosoFileReadProperty( PROPERTY* Property );
```

Tries to read a standard property set from the read cache.

Parameters

Property

[*PROPERTY**] Address of standard property set to be read. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Return Value

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Related Topics:

File Handling >

TosoFileReadCommaProperty (File Handling)

Contents **Syntax**

```
void TosoFileReadCommaProperty( PROPERTY* Property );
```

Tries to read a comma followed by a standard property set from the read cache.

Contents **Parameters**

Property

[*PROPERTY**] Address of standard property set to be read. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Contents **Return Value**

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Contents **Comment**

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Contents

Related Topics:

Contents File Handling >

TosoFileReadDimLine (File Handling)

Syntax

```
void TosoFileReadDimLine( DIMLINE* DimLine );
```

Tries to read a dimension line parameter set from the read cache.

Parameters

DimLine

[*DIMLINE**] Address of dimension line parameter set to be read. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Return Value

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Related Topics:

File Handling >

TosoFileReadCommaDimLine (File Handling)

Syntax

```
void TosoFileReadCommaDimLine( DIMLINE* DimLine );
```

Tries to read a comma followed by a dimension line parameter set from the read cache.

Parameters

DimLine

[*DIMLINE**] Address of dimension line parameter set to be read. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Return Value

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Related Topics:

File Handling >

TosoFileReadDimSmall (File Handling)

Contents Syntax

```
void TosoFileReadDimSmall( DIMSMALL* DimSmall );
```

Tries to read a small dimension parameter set from the read cache.

Contents Parameters

DimSmall

[*DIMSMALL**] Address of small dimension parameter set to be read. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Contents Return Value

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Contents Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Contents

Related Topics:

Contents File Handling >

TosoFileReadCommaDimSmall (File Handling)

Contents

Syntax

```
void TosoFileReadCommaDimSmall( DIMSMALL* DimSmall );
```

Tries to read a comma followed by a small dimension parameter set from the read cache.

Contents

Parameters

DimSmall

[*DIMSMALL**] Address of small dimension parameter set to be read. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Contents

Return Value

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Contents

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Contents

Related Topics:

Contents File Handling >

TosoFileReadDimLarge (File Handling)

Syntax

```
void TosoFileReadDimLarge( DIMLARGE* DimLarge );
```

Tries to read a large dimension parameter set from the read cache.

Parameters

DimLarge

[*DIMLARGE**] Address of large dimension parameter set to be read. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Return Value

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Related Topics:

File Handling >

TosoFileReadCommaDimLarge (File Handling)

Contents **Syntax**

```
void TosoFileReadCommaDimLarge( DIMLARGE* DimLarge );
```

Tries to read a comma followed by a large dimension parameter set from the read cache.

Contents **Parameters**

DimLarge

[*DIMLARGE**] Address of large dimension parameter set to be read. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Contents **Return Value**

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Contents **Comment**

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Contents

Related Topics:

Contents File Handling >

TosoFileReadTextStandard (File Handling)

Contents

Syntax

```
void TosoFileReadTextStandard( TEXTSTANDARD* TextStandard );
```

Tries to read a standard text parameter set from the read cache.

Contents

Parameters

TextStandard

[*TEXTSTANDARD**] Address of standard text parameter set to be read. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Contents

Return Value

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Contents

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Contents

Related Topics:

Contents File Handling >

TosoFileReadCommaTextStandard (File Handling)

Contents Syntax

```
void TosoFileReadCommaTextStandard( TEXTSTANDARD* TextStandard );
```

Tries to read a comma followed by a standard text parameter set from the read cache.

Contents Parameters

TextStandard

[*TEXTSTANDARD**] Address of standard text parameter set to be read. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Contents Return Value

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Contents Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Contents

Related Topics:

Contents File Handling >

TosoFileReadTextFrame (File Handling)

Syntax

```
void TosoFileReadTextFrame( TEXTFRAME* TextFrame );
```

Tries to read a frame text parameter set from the read cache.

Parameters

TextFrame

[*TEXTFRAME**] Address of frame text parameter set to be read. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Return Value

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Related Topics:

File Handling >

TosoFileReadCommaTextFrame (File Handling)

Syntax

```
void TosoFileReadCommaTextFrame( TEXTFRAME* TextFrame );
```

Tries to read a comma followed by a frame text parameter set from the read cache.

Parameters

TextFrame

[*TEXTFRAME**] Address of frame text parameter set to be read. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Return Value

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Related Topics:

File Handling >

TosoFileReadTextReference (File Handling)

Syntax

```
void TosoFileReadTextReference( TEXTREFERENCE* TextReference );
```

Tries to read a reference text parameter set from the read cache.

Parameters

TextReference

[*TEXTREFERENCE**] Address of reference text parameter set to be read. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Return Value

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Related Topics:

File Handling >

TosoFileReadCommaTextReference (File Handling)

Syntax

```
void TosoFileReadCommaTextReference( TEXTREFERENCE* TextReference );
```

Tries to read a comma followed by a reference text parameter set from the read cache.

Parameters

TextReference

[*TEXTREFERENCE**] Address of reference text parameter set to be read. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Return Value

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Related Topics:

File Handling >

TosoFileReadClipSurface (File Handling)

Contents Syntax

```
void TosoFileReadClipSurface( CLIPSURFACE* ClipSurface );
```

Tries to read a clipping surface parameter set from the read cache.

Contents Parameters

ClipSurface

[*CLIPSURFACE**] Address of clipping surface parameter set to be read. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Contents Return Value

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Contents Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Contents

Related Topics:

Contents File Handling >

TosoFileReadCommaClipSurface (File Handling)

Syntax

```
void TosoFileReadCommaClipSurface( CLIPSURFACE* ClipSurface );
```

Tries to read a comma followed by a clipping surface parameter set from the read cache.

Parameters

ClipSurface

[*CLIPSURFACE**] Address of clipping surface parameter set to be read. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Return Value

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Related Topics:

File Handling >

TosoFileReadBitmapRef (File Handling)

Contents Syntax

```
void TosoFileReadBitmapRef( BITMAPREF* BitmapRef );
```

Tries to read a bitmap reference parameter set from the read cache.

Contents Parameters

BitmapRef

[*BITMAPREF**] Address of bitmap reference parameter set to be read. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Contents Return Value

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Contents Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Contents

Related Topics:

Contents File Handling >

TosoFileReadCommaBitmapRef (File Handling)

Syntax

```
void TosoFileReadCommaBitmapRef( BITMAPREF* BitmapRef );
```

Tries to read a comma followed by a bitmap reference parameter set from the read cache.

Parameters

BitmapRef

[*BITMAPREF**] Address of bitmap reference parameter set to be read. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Return Value

None, but if a read error occurs, the return value of TosoFileReadError will be set to TRUE. Once a read error occured, all further calls to *any* read procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Related Topics:

File Handling >

TosoFileReadHeader (File Handling)

Contents

Syntax

```
BOOL TosoFileReadHeader( const LPSTR Header );
```

Starts reading from a file by first reading its file header (a non-quoted string of up to 21 characters in length). After correctly reading the header, the first keyword is read. This keyword can then be retrieved by means of TosoFileReadLastKeyword.

Contents

Parameters

Header

[*const LPSTR*] Address of header text that is expected. A standard file header should not have more than 21 characters (plus the terminating zero-character). The file header of TVG 4.0 files is TVG_40_HEADER.

Contents

Return Value

Returns TRUE if the file does have the correct header, else FALSE. Additionally, the return value of TosoFileReadError will be set to TRUE if an error occured.

Contents

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Contents

Related Topics:

Contents File Handling >

TosoFileReadEndOfFile (File Handling)

Syntax

```
BOOL TosoFileReadEndOfFile( void );
```

Checks whether the current file is terminated correctly by testing whether the last keyword read was `=EXIT=`. Even though the module could test this manually, do always use this procedure to be compatible with future releases of the interface!

Parameters

None.

Return Value

Returns TRUE if the file was terminated correctly by the keyword `=EXIT=`, else FALSE. Additionally, the return value of TosoFileReadError will be set to TRUE if an error occured.

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileReadDelimiters!

Related Topics:

File Handling >

TosoFileReadDelimiters (File Handling)

Syntax

```
void TosoFileReadDelimiters( const TOKEN_DATA* Data );
```

Sets the delimiters used by the parser to separate special file elements (like commas, semicolon, line feeds, escape characters etc.).

Parameters

Data

[*const TOKEN_DATA**] Address of delimiter description data. Each value that is non-zero will be activated. If a value is zero, the corresponding settings remains unchanged. In order to reset to the default delimiters, set *Data* to NULL.

Return Value

None.

Comment

The settings of TosoFileReadDelimiters is reset to its default values each time one the procedures TosoFileReadInitDisk or TosoFileReadInitMemory is called.

Related Topics:

File Handling >

TosoFileReadError (File Handling)

Syntax

```
BOOL TosoFileReadError( void );
```

This procedure is used to determine whether a read error has occured during file read operations. Once a read error occured, all further calls to read procedures will be ignored and will, if applicable, return FALSE.

Parameters

None.

Return Value

Returns TRUE if any error has occured during file read operations. This usually indicates that either an invalid data value or the end-of-file has been found.

Comment

None.

Related Topics:

File Handling >

TosoFileReadTotalSize (File Handling)

Syntax

```
long TosoFileReadTotalSize( void );
```

This procedure is used to determine the total size of the currently open file.

Parameters

None.

Return Value

Size of the opened file in bytes. If the file is a memory image, the value returned might be slightly above the real size of the image (in fact, it is the value returned by the GlobalSize procedure of Win32).

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

Related Topics:

File Handling >

TosoFileReadCurrentSize (File Handling)

Syntax

```
long TosoFileReadCurrentSize( void );
```

This procedure is used to determine the number of bytes that have been read from the currently open file until now.

Parameters

None.

Return Value

Number of bytes read from the file since it was opened.

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

Related Topics:

File Handling >

TosoFileReadCurrentLine (File Handling)

Syntax

```
long TosoFileReadCurrentLine( void );
```

This procedure is used to determine the number of lines that have been read from the currently open file until now.

Parameters

None.

Return Value

Number of lines read from the file since it was opened.

Comment

This procedure relies on a high-performance read cache previously established by either TosoFileReadInitDisk or TosoFileReadInitMemory. As a result, it will fail if this cache is currently not activated!

Related Topics:

File Handling >

TosoFileWriteInitNull (File Handling)

Contents

Syntax

```
BOOL TosoFileWriteInitNull( void );
```

Prepares writing to Null (i.e. to nowhere) using an additional, high-performance read cache. This cache will automatically be used by all TosoFileWrite* procedures.

Contents

Parameters

None.

Contents

Return Value

Returns TRUE if the cache has been established successfully, else FALSE (indicating insufficient memory).

Contents

Comment

Data written to Null will not be saved anywhere, only the bytes and lines "written" will be counted. This can be used to determine a file's size without explicitly writing it to disk. Usually, this procedure is used prior to TosoFileWriteInitMemory to determine the required size of the memory image.

Contents

Related Topics:

Contents File Handling >

TosoFileWriteInitDisk (File Handling)

Syntax

```
BOOL TosoFileWriteInitDisk( HANDLE FileHandle );
```

Prepares writing to a disk file using an additional, high-performance write cache. This cache will automatically be used by all TosoFileWrite* procedures.

Parameters

FileHandle

[*HANDLE*] File handle of the file previously opened for write access.

Return Value

Returns TRUE if the cache has been established successfully, else FALSE (indicating insufficient memory).

Comment

None.

Related Topics:

File Handling >

TosoFileWriteInitMemory (File Handling)

Syntax

```
BOOL TosoFileWriteInitMemory( HGLOBAL hMemory, long MemorySize );
```

Prepares writing to a memory image using an additional, high-performance write cache. This cache will automatically be used by all TosoFileWrite* procedures.

Parameters

hMemory

[*HGLOBAL*] Handle of allocated global memory. This memory must have been allocated using the GMEM_MOVEABLE attribute.

MemorySize

[*long*] Maximum number of bytes that may be written to the global memory area. This value may be less than the memory area's size.

Return Value

Returns TRUE if the cache has been established successfully, else FALSE (indicating insufficient memory).

Comment

None.

Related Topics:

File Handling >

TosoFileWriteData (File Handling)

Contents Syntax

```
BOOL TosoFileWriteData( const LPVOID Data, int Size );
```

Writes a given number of bytes to the current write cache.

Contents Parameters

Data

[*const LPVOID*] Address of a buffer containing the data to be written to the file.

Size

[*int*] Number of bytes to be written to the file.

Contents Return Value

Returns TRUE if the data has been written successfully, else FALSE (indicating disk full). Additionally, the return value of TosoFileWriteError will be set to TRUE if an error occured.

Contents Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitMull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

Contents

Related Topics:

Contents

File Handling >

TosoFileWriteTextData (File Handling)

Syntax

```
BOOL TosoFileWriteTextData( const LPSTR Data );
```

Writes a null-terminated string (not including the termination character) to the current write cache.

Parameters

Data

[*const LPSTR*] Address of a buffer containing the null-terminated string to be written to the file.

Return Value

Returns TRUE if the data has been written successfully, else FALSE (indicating disk full). Additionally, the return value of TosoFileWriteError will be set to TRUE if an error occured.

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitMull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

Related Topics:

File Handling >

TosoFileWriteFlush (File Handling)

Syntax

```
BOOL TosoFileWriteFlush( void );
```

Flushes the current buffer's content to the file.

Parameters

None.

Return Value

Returns TRUE if the data has been written successfully, else FALSE (indicating disk full). Additionally, the return value of TosoFileWriteError will be set to TRUE if an error occured.

Comment

This procedure should be called before TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory to avoid data loss!

Related Topics:

File Handling >

TosoFileWriteExit (File Handling)

Syntax

```
void TosoFileWriteExit( void );
```

Frees the high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory.

Parameters

None.

Return Value

None.

Comment

This procedure will *not* write the current buffer's content to the file! In order to do so, call TosoFileWriteFlush before freeing the write cache.

Related Topics:

File Handling >

TosoFileWriteNewline (File Handling)

Syntax

```
void TosoFileWriteNewline( void );
```

Writes a newline-string to the current write cache.

Parameters

None.

Return Value

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Related Topics:

File Handling >

TosoFileWriteComma (File Handling)

Syntax

```
void TosoFileWriteComma( void );
```

Writes a comma to the current write cache.

Parameters

None.

Return Value

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Related Topics:

File Handling >

TosoFileWriteSemi (File Handling)

Syntax

```
void TosoFileWriteSemi( void );
```

Writes a semicolon to the current write cache.

Parameters

None.

Return Value

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Related Topics:

File Handling >

TosoFileWriteComment (File Handling)

Syntax

```
void TosoFileWriteComment( const LPSTR Value );
```

Writes a comment to the current write cache.

Parameters

Value

[*const LPSTR*] Address of a buffer containing the null-terminated comment string to be written to the file.

Return Value

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Related Topics:

File Handling >

TosoFileWriteKeyword (File Handling)

Contents **Syntax**

```
void TosoFileWriteKeyword( int KeyNum );
```

Writes a keyword (including its terminating semicolon) to the current write cache.

Contents **Parameters**

KeyNum

[*int*] ID of the keyword to be written. Possible values are:

TVG_KEY_EXIT
: Writes the keyword =EXIT=.

TVG_KEY_END
: Writes the keyword =END=.

TVG_KEY_DRAWING
: Writes the keyword =DRAWING=.

TVG_KEY_LIBRARY
: Writes the keyword =LIBRARY=.

TVG_KEY_ATTRIB
: Writes the keyword =ATTRIB=.

TVG_KEY_PAGE
: Writes the keyword =PAGE=.

TVG_KEY_DEFAULT
: Writes the keyword =DEFAULT=.

TVG_KEY_ZOOM
: Writes the keyword =ZOOM=.

TVG_KEY_SYSTEM
: Writes the keyword =SYSTEM=.

TVG_KEY_PEN
: Writes the keyword =PEN=.

TVG_KEY_LINE
: Writes the keyword =LINE=.

TVG_KEY_LAYER
: Writes the keyword =LAYER=.

TVG_KEY_HATCH
: Writes the keyword =HATCH=.

TVG_KEY_BLOCK
: Writes the keyword =BLOCK=.

TVG_KEY_OBJECT
: Writes the keyword =OBJECT=.

TVG_KEY_COLOR
Writes the keyword =COLOR=.

TVG_KEY_KEYBOARD
Writes the keyword =KEYBOARD=.

TVG_KEY_TOOLBOX
Writes the keyword =TOOLBOX=.

TVG_KEY_SYMBOL
Writes the keyword =SYMBOL=.

TVG_KEY_USER
Writes the keyword =USER=.

TVG_KEY_MODULE
Writes the keyword =MODULE=.

For a detailed description of the keyword's usage, see the TVG 4.0 file format description in TVG40.HLP.

Contents

Return Value

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return FALSE).

Contents

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Contents

Related Topics:

Contents File Handling >

TosoFileWriteString (File Handling)

Syntax

```
void TosoFileWriteString( const LPSTR Value );
```

Writes a quoted string to the current write cache. If the string does contain escape characters, these will be encoded correctly.

Parameters

Value

[*const LPSTR*] Address of a buffer containing the null-terminated string to be written to the file.

Return Value

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Related Topics:

File Handling >

TosoFileWriteCommaString (File Handling)

Syntax

```
void TosoFileWriteCommaString( const LPSTR Value );
```

Writes a comma followed by a quoted string to the current write cache. If the string does contain escape characters, these will be encoded correctly.

Parameters

Value

[*const LPSTR*] Address of a buffer containing the null-terminated string to be written to the file.

Return Value

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Related Topics:

File Handling >

TosoFileWriteBinary (File Handling)

Syntax

```
void TosoFileWriteBinary( const LPVOID Value,
                          int Size );
```

Writes an encoded binary string to the current write cache.

Parameters

Value

[*const LPVOID*] Address of a buffer containing the binary data to be written to the file.

Size

[*int*] Number of bytes to be written to the file.

Return Value

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Related Topics:

File Handling >

TosoFileWriteCommaBinary (File Handling)

Syntax

```
void TosoFileWriteCommaBinary( const LPVOID Value,
                               int Size );
```

Writes a comma followed by an encoded binary string to the current write cache.

Parameters

Value

[*const LPVOID*] Address of a buffer containing the binary data to be written to the file.

Size

[*int*] Number of bytes to be written to the file.

Return Value

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Related Topics:

File Handling >

TosoFileWriteBool (File Handling)

Contents

Syntax

```
void TosoFileWriteBool( BOOL Value );
```

Writes a boolean value to the current write cache.

Contents

Parameters

Value

[*BOOL*] Boolean value to be written.

Contents

Return Value

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return FALSE).

Contents

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Contents

Related Topics:

Contents File Handling >

TosoFileWriteCommaBool (File Handling)

Syntax

```
void TosoFileWriteCommaBool( BOOL Value );
```

Writes a comma followed by a boolean value to the current write cache.

Parameters

Value

[*BOOL*] Boolean value to be written.

Return Value

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Related Topics:

File Handling >

TosoFileWriteShort (File Handling)

Syntax

```
void TosoFileWriteShort( short Value );
```

Writes a short integer value to the current write cache.

Parameters

Value

[*short*] Short integer value to be written.

Return Value

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TOSO_TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return TOSO_FALSE).

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Related Topics:

File Handling >

TosoFileWriteCommaShort (File Handling)

Syntax

```
void TosoFileWriteCommaShort( short Value );
```

Writes a comma followed by a short integer value to the current write cache.

Parameters

Value

[*short*] Short integer value to be written.

Return Value

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Related Topics:

File Handling >

TosoFileWriteInt (File Handling)

Syntax

```
void TosoFileWriteInt( int Value );
```

Writes an integer value to the current write cache.

Parameters

Value

[*int*] Integer value to be written.

Return Value

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Related Topics:

File Handling >

TosoFileWriteCommaInt (File Handling)

Contents

Syntax

```
void TosoFileWriteCommaInt( int Value );
```

Writes a comma followed by an integer value to the current write cache.

Contents

Parameters

Value

[*int*] Integer value to be written.

Contents

Return Value

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return FALSE).

Contents

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Contents

Related Topics:

Contents File Handling >

TosoFileWriteDouble (File Handling)

Syntax

```
void TosoFileWriteDouble( double Value );
```

Writes a double precision floating point value to the current write cache.

Parameters

Value

[*double*] Double precision floating point value to be written.

Return Value

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Related Topics:

File Handling >

TosoFileWriteCommaDouble (File Handling)

Syntax

```
void TosoFileWriteCommaDouble( double Value );
```

Writes a comma followed by a double precision floating point value to the current write cache.

Parameters

Value

[*double*] Double precision floating point value to be written.

Return Value

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Related Topics:

File Handling >

TosoFileWriteColorref (File Handling)

Contents Syntax

```
void TosoFileWriteColorref( COLORREF Color );
```

Writes a color definition to the current write cache.

Contents Parameters

Color

[*COLORREF*] Color definition to be written. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Contents Return Value

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return FALSE).

Contents Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Contents

Related Topics:

Contents File Handling >

TosoFileWriteCommaColorref (File Handling)

Syntax

```
void TosoFileWriteCommaColorref( COLORREF Color );
```

Writes a comma followed by a color definition to the current write cache.

Parameters

Color

[*COLORREF*] Color definition to be written. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Return Value

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Related Topics:

File Handling >

TosoFileWriteFontdef (File Handling)

Syntax

```
void TosoFileWriteFontdef( const FONTDEF* Font );
```

Writes a font definition to the current write cache.

Parameters

Font

[*FONTDEF*] Font definition to be written. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Return Value

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Related Topics:

File Handling >

TosoFileWriteCommaFontdef (File Handling)

Contents **Syntax**

```
void TosoFileWriteCommaFontdef( const FONTDEF* Font );
```

Writes a comma followed by a font definition to the current write cache.

Contents **Parameters**

Font

[*FONTDEF*] Font definition to be written. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Contents **Return Value**

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return FALSE).

Contents **Comment**

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Contents

Related Topics:

Contents File Handling >

TosoFileWriteXProperty (File Handling)

Contents Syntax

```
void TosoFileWriteXProperty( const XPROPERTY* XProperty );
```

Writes an extended property set to the current write cache.

Contents Parameters

XProperty

[*const XPROPERTY**] Address of extended property set to be written. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Contents Return Value

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return FALSE).

Contents Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Contents

Related Topics:

Contents

File Handling >

TosoFileWriteCommaXProperty (File Handling)

Contents

Syntax

```
void TosoFileWriteCommaXProperty( const XPROPERTY* XProperty );
```

Writes a comma followed by an extended property set to the current write cache.

Contents

Parameters

XProperty

[*const XPROPERTY**] Address of extended property set to be written. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Contents

Return Value

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return FALSE).

Contents

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Contents

Related Topics:

Contents File Handling >

TosoFileWriteProperty (File Handling)

Syntax

```
void TosoFileWriteProperty( const PROPERTY* Property );
```

Writes a property set to the current write cache.

Parameters

Property

[*const PROPERTY**] Address of property set to be written. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Return Value

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Related Topics:

File Handling >

TosoFileWriteCommaProperty (File Handling)

Syntax

```
void TosoFileWriteCommaProperty( const PROPERTY* Property );
```

Writes a comma followed by a property set to the current write cache.

Parameters

Property

[*const PROPERTY**] Address of property set to be written. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Return Value

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Related Topics:

File Handling >

TosoFileWriteDimLine (File Handling)

Syntax

```
void TosoFileWriteDimLine( const DIMLINE* DimLine );
```

Writes a dimension line parameter set to the current write cache.

Parameters

DimLine

[*const DIMLINE**] Address of dimension line parameter set to be written. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Return Value

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Related Topics:

File Handling >

TosoFileWriteCommaDimLine (File Handling)

Syntax

```
void TosoFileWriteCommaDimLine( const DIMLINE* DimLine );
```

Writes a comma followed by a dimension line parameter set to the current write cache.

Parameters

DimLine

[*const DIMLINE**] Address of dimension line parameter set to be written. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Return Value

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Related Topics:

File Handling >

TosoFileWriteDimSmall (File Handling)

Syntax

```
void TosoFileWriteDimSmall( const DIMSMALL* DimSmall );
```

Writes a small dimension parameter set to the current write cache.

Parameters

DimSmall

[*const DIMSMALL**] Address of small dimension parameter set to be written. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Return Value

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Related Topics:

File Handling >

TosoFileWriteCommaDimSmall (File Handling)

Contents

Syntax

```
void TosoFileWriteCommaDimSmall( const DIMSMALL* DimSmall );
```

Writes a comma followed by a small dimension parameter set to the current write cache.

Contents

Parameters

DimSmall

[*const DIMSMALL**] Address of small dimension parameter set to be written. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Contents

Return Value

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return FALSE).

Contents

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Contents

Related Topics:

Contents File Handling >

TosoFileWriteDimLarge (File Handling)

Contents

Syntax

```
void TosoFileWriteDimLarge( const DIMLARGE* DimLarge );
```

Writes a large dimension parameter set to the current write cache.

Contents

Parameters

DimLarge

[*const DIMLARGE**] Address of large dimension parameter set to be written. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Contents

Return Value

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return FALSE).

Contents

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Contents

Related Topics:

Contents File Handling >

TosoFileWriteCommaDimLarge (File Handling)

Contents

Syntax

```
void TosoFileWriteCommaDimLarge( const DIMLARGE* DimLarge );
```

Writes a comma followed by a large dimension parameter set to the current write cache.

Contents

Parameters

DimLarge

[*const DIMLARGE**] Address of large dimension parameter set to be written. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Contents

Return Value

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return FALSE).

Contents

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Contents

Related Topics:

Contents File Handling >

TosoFileWriteTextStandard (File Handling)

Syntax

```
void TosoFileWriteTextStandard( const TEXTSTANDARD* TextStandard );
```

Writes a standard text parameter set to the current write cache.

Parameters

TextStandard

[*const TEXTSTANDARD**] Address of standard text parameter set to be written. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Return Value

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Related Topics:

File Handling >

TosoFileWriteCommaTextStandard (File Handling)

Syntax

```
void TosoFileWriteCommaTextStandard( const TEXTSTANDARD* TextStandard );
```

Writes a comma followed by a standard text parameter set to the current write cache.

Parameters

TextStandard

[*const TEXTSTANDARD**] Address of standard text parameter set to be written. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Return Value

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Related Topics:

File Handling >

TosoFileWriteTextFrame (File Handling)

Syntax

```
void TosoFileWriteTextFrame( const TEXTFRAME* TextFrame );
```

Writes a frame text parameter set to the current write cache.

Parameters

TextFrame

[*const TEXTFRAME**] Address of frame text parameter set to be written. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Return Value

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Related Topics:

File Handling >

TosoFileWriteCommaTextFrame (File Handling)

Syntax

```
void TosoFileWriteCommaTextFrame( const TEXTFRAME* TextFrame );
```

Writes a comma followed by a frame text parameter set to the current write cache.

Parameters

TextFrame

[*const TEXTFRAME**] Address of frame text parameter set to be written. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Return Value

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Related Topics:

File Handling >

TosoFileWriteTextReference (File Handling)

Syntax

```
void TosoFileWriteTextReference( const TEXTREFERENCE* TextReference );
```

Writes a reference text parameter set to the current write cache.

Parameters

TextReference

[*const TEXTREFERENCE**] Address of reference text parameter set to be written. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Return Value

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Related Topics:

File Handling >

TosoFileWriteCommaTextReference (File Handling)

Syntax

```
void TosoFileWriteCommaTextReference( const TEXTREFERENCE* TextReference );
```

Writes a comma followed by a reference text parameter set to the current write cache.

Parameters

TextReference

[*const TEXTREFERENCE**] Address of reference text parameter set to be written. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Return Value

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Related Topics:

File Handling >

TosoFileWriteClipSurface (File Handling)

Syntax

```
void TosoFileWriteClipSurface( const CLIPSURFACE* ClipSurface );
```

Writes a clipping surface parameter set to the current write cache.

Parameters

ClipSurface

[*const CLIPSURFACE**] Address of clipping surface parameter set to be written. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Return Value

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Related Topics:

File Handling >

TosoFileWriteCommaClipSurface (File Handling)

Syntax

```
void TosoFileWriteCommaClipSurface( const CLIPSURFACE* ClipSurface );
```

Writes a comma followed by a clipping surface parameter set to the current write cache.

Parameters

ClipSurface

[*const CLIPSURFACE**] Address of clipping surface parameter set to be written. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Return Value

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Related Topics:

File Handling >

TosoFileWriteBitmapRef (File Handling)

Syntax

```
void TosoFileWriteBitmapRef( const BITMAPREF* BitmapRef );
```

Writes a bitmap reference parameter set to the current write cache.

Parameters

BitmapRef

[*const BITMAPREF**] Address of bitmap reference parameter set to be written. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Return Value

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Related Topics:

File Handling >

TosoFileWriteCommaBitmapRef (File Handling)

Syntax

```
void TosoFileWriteCommaBitmapRef( const BITMAPREF* BitmapRef );
```

Writes a comma followed by a bitmap reference surface parameter set to the current write cache.

Parameters

BitmapRef

[*const BITMAPREF**] Address of bitmap reference parameter set to be written. For a syntax description of this type, see the TVG 4.0 file format description in TVG 4.0 Documentation (TVG40.HLP).

Return Value

None, but if a write error occurs, the return value of TosoFileWriteError will be set to TRUE. Once a write error occured, all further calls to *any* write procedure will be ignored (i.e. return FALSE).

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Related Topics:

File Handling >

TosoFileWriteHeader (File Handling)

Syntax

```
BOOL TosoFileWriteHeader( const LPSTR Header );
```

Starts writing to a file by first writing its file header (a non-quoted string of up to 21 characters in length) following by a semicolon.

Parameters

Header

[*const LPSTR*] Address of null-terminated header string to be written.

Return Value

Returns TRUE if the header has been written successfully, else FALSE (indicating disk full). Additionally, the return value of TosoFileWriteError will be set to TRUE if an error occured.

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Related Topics:

File Handling >

TosoFileWriteEndOfFile (File Handling)

Contents

Syntax

```
BOOL TosoFileWriteEndOfFile( void );
```

Terminate the current file by writing the `=EXIT=` keyword followed by a semicolon. Even though the module could do this manually, do always use this procedure to be compatible with future releases of the interface!

Contents

Parameters

None.

Contents

Return Value

Returns TRUE if the data has been written successfully, else FALSE (indicating disk full). Additionally, the return value of TosoFileWriteError will be set to TRUE if an error occured.

Contents

Comment

This procedure relies on a high-performance write cache previously established by either TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory. As a result, it will fail if this cache is currently not activated!

The behaviour of this procedure depends on the settings of TosoFileWriteDelimiters and TosoFileWriteZeros!

Contents

Related Topics:

Contents File Handling >

TosoFileWriteZeros (File Handling)

Contents

Syntax

```
void TosoFileWriteZeros( BOOL Value );
```

Contents

Parameters

Value

[*BOOL*] This value indicates whether file write operations shall write zero-values or not. Inside TVG 4.0 files, zeros are usually not stated to reduce file size, so FALSE is the default value.

Contents

Return Value

None.

Contents

Comment

The setting of TosoFileWriteZeros is reset to its default value each time one the procedures TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory is called.

Contents

Related Topics:

Contents File Handling >

TosoFileWriteDelimiters (File Handling)

Syntax

```
void TosoFileWriteDelimiters( const TOKEN_DATA* Data );
```

Parameters

Data

[*const TOKEN_DATA**] Integer value to be converted into a string.

Return Value

None.

Comment

The settings of TosoFileWriteDelimiters is reset to its default values each time one the procedures TosoFileWriteInitNull, TosoFileWriteInitDisk or TosoFileWriteInitMemory is called.

Related Topics:

File Handling >

TosoFileWriteError (File Handling)

Syntax

```
BOOL TosoFileWriteError( void );
```

This procedure is used to determine whether a write error has occured during file write operations. Once a write error occured, all further calls to write procedures will be ignored and will, if applicable, return FALSE.

Parameters

None.

Return Value

Returns TRUE if any error has occured during file write operations. Usually, this indicates that the destination drive is full.

Comment

None.

Related Topics:

File Handling >

TosoFileWriteCurrentSize (File Handling)

Syntax

```
long TosoFileWriteCurrentSize( void );
```

Parameters

None.

Return Value

Number of bytes written to the file since it was created.

Comment

None.

Related Topics:

File Handling >

TosoFileWriteCurrentLine (File Handling)

Syntax

```
long TosoFileWriteCurrentLine( void );
```

Parameters

None.

Return Value

Number of lines written to the file since it was created.

Comment

None.

Related Topics:

File Handling >

TosoProfileReadKeyOpen (Profile and Settings)

Syntax

```
BOOL TosoProfileReadKeyOpen( const LPSTR KeyName,
                             BOOL Common );
```

Opens a profile key for reading in the registry database.

Parameters

KeyName

[*const LPSTR*] Name of the module's key. This key is placed inside a superior module directory inside the serving application's registration key. Usually, this string should contain the module's file name without any extension (e.g. "EXPDXF" or "TSTEXT1"), since this name is guaranteed to be unique. If you want to access a subkey to a module's key, add the subkey's name separated by a backslash (e.g. "EXPDXF\FileList").

Common

[*BOOL*] This value determines whether the key is located in the user-dependent HKEY_CURRENT_USER key of the registry (if it is FALSE) or inside the common HKEY_LOCAL_MACHINE key (if it is TRUE). Each module should carefully split its settings into user-dependent and non-user-dependent settings to offer the best possible comfort in multi-user environments.

Return Value

Returns TRUE if successful, FALSE if not.

Comment

Only one key can be open for reading at a time. Before being able to open another key for reading, the module has to call TosoProfileReadKeyClose.

Related Topics:

TosoProfileReadInt

TosoProfileReadString

TosoProfileReadData

TosoProfileReadKeyClose

TosoProfileReadInt (Profile and Settings)

Syntax

```
BOOL TosoProfileReadInt( const LPSTR ValueName,
                         int* Value );
```

Reads an int value from a previously opened key in the registry database.

Parameters

ValueName

[*const LPSTR*] Name of the value inside the module's key. This value name must be unique throughout all values stored by the module.

Value

[*int**] Address of a buffer that is to receive the value if successful.

Return Value

Returns TRUE if successful, FALSE if not. The content of *Value* will only be altered if TRUE is returned.

Comment

None.

Related Topics:

TosoProfileReadKeyOpen

TosoProfileReadString

TosoProfileReadData

TosoProfileReadKeyClose

TosoProfileWriteKeyOpen

TosoProfileWriteInt

TosoProfileWriteString

TosoProfileReadString (Profile and Settings)

Syntax

```
BOOL TosoProfileReadString( const LPSTR ValueName,
                            LPSTR Value );
```

Reads a string value from a previously opened key in the registry database.

Parameters

ValueName

[*const LPSTR*] Name of the value inside the module's key. This value name must be unique throughout all values stored by the module.

Value

[*LPSTR*] Address of a buffer that is to receive the value if successful.

Return Value

Returns TRUE if successful, FALSE if not. The content of *Value* will only be altered if TRUE is returned.

Comment

None.

Related Topics:

TosoProfileReadKeyOpen

TosoProfileReadInt

TosoProfileReadData

TosoProfileReadKeyClose

TosoProfileWriteKeyOpen

TosoProfileWriteInt

TosoProfileWriteString

TosoProfileReadData (Profile and Settings)

Syntax

```
BOOL TosoProfileReadData( const LPSTR ValueName,
                          LPBYTE Value,
                          int Size );
```

Reads binary data from a previously opened key in the registry database.

Parameters

ValueName

[*const LPSTR*] Name of the value inside the module's key. This value name must be unique throughout all values stored by the module.

Value

[*LPBYTE*] Address of a buffer that is to receive the data if successful.

Size

[*int*] Size of the buffer in bytes.

Return Value

Returns TRUE if successful, FALSE if not (e.g. if the buffer pointed to by *Value* is too small). The content of *Value* will only be altered if TRUE is returned.

Comment

None.

Related Topics:

TosoProfileReadKeyOpen

TosoProfileReadInt

TosoProfileReadString

TosoProfileReadKeyClose

TosoProfileWriteKeyOpen

TosoProfileReadKeyClose (Profile and Settings)

Syntax

```
void TosoProfileReadKeyClose( void );
```

Closes the profile key in the registry database.

Parameters

None.

Return Value

None.

Comment

None.

Related Topics:

- TosoProfileReadKeyOpen
- TosoProfileReadInt
- TosoProfileReadString
- TosoProfileReadData
- TosoProfileWriteKeyOpen
- TosoProfileWriteInt
- TosoProfileWriteString
- TosoProfileWriteData

TosoProfileWriteKeyOpen (Profile and Settings)

Syntax

```
BOOL TosoProfileWriteKeyOpen( const LPSTR KeyName,
                              BOOL Common );
```

Opens a profile key for writing in the registry database.

Parameters

KeyName

[*const LPSTR*] Name of the module's key. This key will be placed inside a superior module directory inside the serving application's registration key. Usually, this string should contain the module's file name without any extension (e.g. "EXPDXF" or "TSTEXT1"), since this name is guaranteed to be unique. If you want to access a subkey to a module's key, add the subkey's name separated by a backslash (e.g. "EXPDXF\FileList"). If the key does not exist, it will be created.

Common

[*BOOL*] This value determines whether the key is to be placed in the user-dependent HKEY_CURRENT_USER key of the registry (if it is FALSE) or inside the common HKEY_LOCAL_MACHINE key (if it is TRUE). Each module should carefully split its settings into user-dependent and non-user-dependent settings to offer the best possible comfort in multi-user environments.

Return Value

Returns TRUE if successful, FALSE if not.

Comment

Only one key can be open for writing at a time. Before being able to open another key for writing, the module has to call TosoProfileWriteKeyClose.

Contents

Related Topics:

TosoProfileReadKeyOpen

TosoProfileReadInt

TosoProfileReadString

TosoProfileReadData

TosoProfileWriteInt (Profile and Settings)

Syntax

```
BOOL TosoProfileWriteInt( const LPSTR ValueName,
                          int Value );
```

Writes an int value to a previously opened key in the registry database.

Parameters

ValueName

[*const LPSTR*] Name of the value inside the module's key. This value name must be unique throughout all values stored by the module.

Value

[*int*] Value to be stored in the registry.

Return Value

Returns TRUE if successful, FALSE if not.

Comment

None.

Related Topics:

TosoProfileReadKeyOpen

TosoProfileReadInt

TosoProfileReadString

TosoProfileReadData

TosoProfileReadKeyClose

TosoProfileWriteKeyOpen

TosoProfileWriteString

TosoProfileWriteString (Profile and Settings)

Contents Syntax

```
BOOL TosoProfileWriteString( const LPSTR ValueName,
                             const LPSTR Value );
```

Writes a string value to a previously opened key in the registry database.

Contents Parameters

ValueName

[*const LPSTR*] Name of the value inside the module's key. This value name must be unique throughout all values stored by the module.

Value

[*const LPSTR*] Value to be stored in the registry. If Value is NULL, the value identified by *KeyName* will be deleted!

Contents Return Value

Returns TRUE if successful, FALSE if not.

Contents Comment

None.

Contents

Related Topics:

Contents TosoProfileReadKeyOpen

Contents TosoProfileReadInt

Contents TosoProfileReadString

Contents TosoProfileReadData

Contents TosoProfileReadKeyClose

Contents TosoProfileWriteKeyOpen

TosoProfileWriteInt

TosoProfileWriteData

TosoProfileWriteKeyClose

TosoProfileDeleteKey

TosoProfileWriteData (Profile and Settings)

Contents Syntax

```
BOOL TosoProfileWriteData( const LPSTR ValueName,
                           const LPBYTE Value,
                           int Size );
```

Writes binary data to a previously opened key in the registry database.

Contents Parameters

ValueName

[*const LPSTR*] Name of the value inside the module's key. This value name must be unique throughout all values stored by the module.

Value

[*const LPSTR*] Data to be stored in the registry.

Size

[*int*] Size of the data in bytes.

Contents Return Value

Returns TRUE if successful, FALSE if not.

Contents Comment

None.

Contents

Related Topics:

Contents TosoProfileReadKeyOpen

Contents TosoProfileReadInt

Contents TosoProfileReadString

Contents TosoProfileReadData

Contents TosoProfileReadKeyClose

Contents TosoProfileWriteKeyOpen

TosoProfileWriteKeyClose (Profile and Settings)

Syntax

```
void TosoProfileWriteKeyClose( void );
```

Closes the profile key in the registry database.

Parameters

None.

Return Value

None.

Comment

None.

Related Topics:

- TosoProfileReadKeyOpen
- TosoProfileReadInt
- TosoProfileReadString
- TosoProfileReadData
- TosoProfileReadKeyClose
- TosoProfileWriteKeyOpen
- TosoProfileWriteInt
- TosoProfileWriteString

TosoProfileDeleteKey (Profile and Settings)

Syntax

```
BOOL TosoProfileDeleteKey( const LPSTR KeyName,
                           BOOL Common );
```

Removes a profile key from the registry database.

Parameters

KeyName

[*const LPSTR*] Name of the module's key. This key is placed inside a superior module directory inside the serving application's registration key. Usually, this string should contain the module's file name without any extension (e.g. "EXPDXF" or "TSTEXT1"), since this name is guaranteed to be unique. If you want to delete a subkey to a module's key, add the subkey's name separated by a backslash (e.g. "EXPDXF\FileList").

Common

[*BOOL*] This value determines whether the key is located in the user-dependent HKEY_CURRENT_USER key of the registry (if it is FALSE) or inside the common HKEY_LOCAL_MACHINE key (if it is TRUE). Each module should carefully split its settings into user-dependent and non-user-dependent settings to offer the best possible comfort in multi-user environments.

Return Value

Returns TRUE if successful, FALSE if not.

Comment

Be sure not to delete a key while it is open either for reading or writing (see TosoProfileReadKeyOpen and TosoProfileWriteKeyOpen).

Related Topics:

TosoProfileReadKeyOpen

TosoProfileReadInt

TosoProfileReadString

TosoProfileReadData

TosoProfileGetDrawing (Profile and Settings)

Contents

Syntax

```
int TosoProfileGetDrawing( int DrawingNum,
                           short OwnerID,
                           int ProfileID,
                           LPVOID ProfileData,
                           int Size );
```

Retrieves a module's profile that has been stored in a drawing.

Contents

Parameters

DrawingNum

[*int*] Zero-based index of the drawing in which the module's profile shall be seeked. Valid range: 0 <= Value < TVG_DRAWING_MAX.

OwnerID

[*short*] This value is a unique identification of the creator of the module. The value DB_OWNER_TOSO is reserved for use by TommySoftware®.

Third-party vendors that plan to implement an external module should contact TommySoftware® to receive their unique identification (see Getting Your Private Owner ID).

ProfileID

[*int*] This value is used to identify the profile for further access. Each module creator can use the full range of the value, as it is used in combination with his *OwnerID*. Usually, this value will be equal to the module's ID.

As this value is a 32bit value (rather than being a 16bit value like the module's ID), each module can store multiple profiles. In order to do so, store the module's ID in the low-word of *ProfileID*, and a additional number in its high-word.

ProfileData

[*LPVOID*] Address of a buffer to receive the profile data. This profile data is a list of valid data blocks (see Data Block Types), ended by a data block of type DB_END.

If this value is NULL, only the size of memory required to store the profile data is determined and returned.

Size

[*int*] Size of the buffer pointed to by *ProfileData*.

Contents

Return Value

Returns the size of the profile data in bytes. If the desired profile does not exist, the return value is 0. If the buffer is not large enough to receive the profile data, the return value negative and no data is copied.

Contents

Comment

None.

Contents

Related Topics:

TosoProfileSetDrawing

TosoProfileSetDrawing (Profile and Settings)

Contents

Syntax

```
BOOL TosoProfileSetDrawing( int DrawingNum,
                            short OwnerID,
                            int ProfileID,
                            const LPVOID ProfileData,
                            int Size );
```

Stores a module's profile in a drawing.

Contents

Parameters

DrawingNum

[*int*] Zero-based index of the drawing in which the module's profile shall be stored. Valid range: 0 <= Value < TVG_DRAWING_MAX.

OwnerID

[*short*] This value is a unique identification of the creator of the module. The value DB_OWNER_TOSO is reserved for use by TommySoftware®.
Third-party vendors that plan to implement an external module should contact TommySoftware® to receive their unique identification (see Getting Your Private Owner ID).

ProfileID

[*int*] This value is used to identify the profile for further access. Each module creator can use the full range of the value, as it is used in combination with his *OwnerID*. Usually, this value will be equal to the module's ID.
As this value is a 32bit value (rather than being a 16bit value like the module's ID), each module can store multiple profiles. In order to do so, store the module's ID in the low-word of *ProfileID*, and an additional number in its high-word.

ProfileData

[*const LPVOID*] Address of a buffer that contains the profile data. This profile data must be a list of valid data blocks (see Data Block Types), ended by a data block of type DB_END.
This profile data will replace a possibly existing profile entry with the same *OwnerID* and *ProfileID*. If the value of *ProfileData* is NULL, only a possibly existing profile entry in the drawing will be deleted without adding a new entry.

Size

[*int*] Size of the profile data pointed to by *ProfileData*. Make sure that this size does include the ending data block of type DB_END! If *ProfileData* is non-NULL and the value of *Size* is less or equal to zero, the procedure will fail. If *ProfileData* is NULL, the value of *Size* will be ignored.

Contents

Return Value

Returns TRUE if the profile was stored successfully, else FALSE (indicating out-of-memory).

Contents

Comment

None.

Contents

Related Topics:

Contents TosoProfileSetDrawing

TosoSettingGet (Profile and Settings)

Contents **Syntax**

```
int TosoSettingGet( int Mode );
```

Retrieves a setting of the serving application.

Contents **Parameters**

Mode

[*int*] This value determines what setting is to be retrieved. Possible values are:

SETTING_DUPLICATE
: Duplicate status (return value is *BOOL*, TRUE = Active, FALSE = Inactive).

SETTING_SNAPACTIVE
: Snap status (return value is *BOOL*, TRUE = Active, FALSE = Inactive).

SETTING_SNAPMODE
: Snap modes (return value is *int*, bitwise OR combination of SNAPMODE_???).

SETTING_SNAPRADIUS
: Snap radius (return value is *int*, 5..50).

SETTING_SNAPTOLERANCE
: Snap radius (return value is *int*, 0..10).

SETTING_ORTHO
: Orthogonal mode (return value is *BOOL*, TRUE = Active, FALSE = Disabled).

SETTING_ARCDIRECTION
: Arc direction (return value is *BOOL*, TRUE = Positive, FALSE = Negative).

SETTING_ARCMODE
: Arc mode (return value is *int*, 0 = Arc, 1 = Sector, 2 = Segment).

SETTING_AREAMODE
: Area mode (return value is *BOOL*, TRUE = Inside, FALSE = Overlapping).

SETTING_GEODISPLAY
: Geometry display (return value is *BOOL*, TRUE = Visible, FALSE = Invisible).

SETTING_GEOFROZEN
: Geometry status (return value is *BOOL*, TRUE = Frozen, FALSE = Editable).

SETTING_COLOR_GEOMETRY
: Color of default geometry objects (return value is *COLORREF*).

SETTING_COLOR_BLOCKS
: Color of block text display (return value is *COLORREF*).

SETTING_COLOR_ERASER
: Color of erasers (return value is *COLORREF*).

SETTING_COLOR_POINT
: Color of standard points (return value is *COLORREF*).

SETTING_COLOR_MARK

Color of markings (return value is *COLORREF*).

SETTING_COLOR_INPUT
Color of input rubber lines (return value is *COLORREF*).

SETTING_COLOR_OBJSEL
Color of permanently selected objects (return value is *COLORREF*).

SETTING_COLOR_OBJUSE
Color of identified objects (return value is *COLORREF*).

SETTING_COLOR_OBJIDENT
Color of object highlight when identification is inaccurate (return value is *COLORREF*).

SETTING_COLOR_POINTUSE
Color of selected points (return value is *COLORREF*).

SETTING_COLOR_CURSOR1
Color of "large" cursor cross in active window (return value is *COLORREF*).

SETTING_COLOR_CURSOR2
Color of "small" cursor cross in inactive windows (return value is *COLORREF*).

SETTING_COLOR_PAGEFRAME
Color of page frame and zoom / layer / system display (return value is *COLORREF*).

SETTING_COLOR_DRAWAREA
Color of drawing area background (return value is *COLORREF*).

SETTING_COLOR_COLORTEXT
Color of colored texts (return value is *COLORREF*).

SETTING_COLOR_SUBMENU
Color of submenu icons (return value is *COLORREF*).

SETTING_COLOR_COMMAND
Color of command icons (return value is *COLORREF*).

Return Value

Returns the desired setting. Depending on the type of setting to be retrieved, this return value has to be interpreted as a boolean value (*BOOL*), a normal integer (*int*), a bitwise OR combination of several integer values (*int*) or a color description (*COLORREF*).

Comment

None.

Related Topics:

TosoSettingSet

TosoSettingSet (Profile and Settings)

Contents

Syntax

```
void TosoSettingSet( int Mode, int Value );
```

Alters a setting of the serving application.

Contents

Parameters

Mode

[*int*] This value determines what setting is to be retrieved. Possible values are:

SETTING_DUPLICATE
: Duplicate status (*Value* is handled as *BOOL*, TRUE = Active, FALSE = Inactive).

SETTING_SNAPACTIVE
: Snap status (*Value* is handled as *BOOL*, TRUE = Active, FALSE = Inactive).

SETTING_SNAPMODE
: Snap modes (*Value* is handled as *int*, bitwise OR combination of SNAPMODE_???).

SETTING_SNAPRADIUS
: Snap radius (*Value* is handled as *int*, 5..50).

SETTING_SNAPTOLERANCE
: Snap radius (*Value* is handled as *int*, 0..10).

SETTING_ORTHO
: Orthogonal mode (*Value* is handled as *BOOL*, TRUE = Active, FALSE = Disabled).

SETTING_ARCDIRECTION
: Arc direction (*Value* is handled as *BOOL*, TRUE = Positive, FALSE = Negative).

SETTING_ARCMODE
: Arc mode (*Value* is handled as *int*, 0 = Arc, 1 = Sector, 2 = Segment).

SETTING_AREAMODE
: Area mode (*Value* is handled as *BOOL*, TRUE = Inside, FALSE = Overlapping).

SETTING_GEODISPLAY
: Geometry display (*Value* is handled as *BOOL*, TRUE = Visible, FALSE = Invisible).

SETTING_GEOFROZEN
: Geometry status (*Value* is handled as *BOOL*, TRUE = Frozen, FALSE = Editable).

Value

[*int*] Value to which the desired setting shall be set. Depending on the type of setting to be altered, this value will either be interpreted as a boolean value (*BOOL*), a normal integer (*int*) or a bitwise OR combination of several integer values (*int*).

Contents

Return Value

None.

Contents

Comment

Please note that this procedure is not able to alter all settings that can be determined by the TosoSettingGet procedure! When calling this procedure with an invalid *Mode* parameter, it will return immediately without performing any action.

If the status of the setting to be altered is displayed by the serving application (e.g. in a button in the panel window), this display will automatically be updated. If the setting influences the drawing's display, this will *not* automatically be updated. Instead, the module should redraw the display by calling TosoDrawWindowAll once after all settings have been modified.

Contents

Related Topics:

Contents TosoSettingGet

TosoCalcCurvature (Geometry and Conversion)

Syntax

```
BOOL TosoCalcCurvature( double x1,
                        double y1,
                        double x2,
                        double y2,
                        double x3,
                        double y3,
                        double* Result );
```

Calculates the curvature for a circular arc determined by its center-point (x1,y1), its start-point (x2,y2) and its end-point (x3,y3). Both (x2,y2) and (x3,y3) should lie exactly on the circle the arc is based on.

Parameters

x1
y1
: [*double*] Coordinates of the arc's center-point in internal millimeters (relative to the page center, scale-independent).

x2
y2
: [*double*] Coordinates of the arc's start-point in internal millimeters (relative to the page center, scale-independent).

x3
y3
: [*double*] Coordinates of the arc's end-point in internal millimeters (relative to the page center, scale-independent).

Result
: [*double**] Address of double buffer to receive the calculated curvature.

Return Value

Returns TRUE if successful. If FALSE is returned, *Result* in invalid.

Comment

None.

Related Topics:

TosoCalcTextFrame

TosoCalcBlockFrame

TosoCalcTextFrame (Geometry and Conversion)

Syntax

```
BOOL TosoCalcTextFrame( const TEXTSTANDARD* Data,
                        const LPSTR Text,
                        DRECT* Frame );
```

Calculates the surrounding frame of a standard text.

Parameters

Data

[*const TEXTSTANDARD**] Standard text parameter set used to display the given text.

Text

[*const LPSTR*] Text whose surrounding frame is to be calculated.

Frame

[*DRECT**] Address of a rectangle buffer to receive the calculated surrounding frame.

Return Value

Returns TRUE if successful. If FALSE is returned, the rectangle pointed to by *Frame* is not modified.

Comment

The resulting frame does surround the character cells, i.e. it is usually larger than the text's image. The values returned in *Frame* will always base on the text's insertion point indicated by *Data->TextMatrix.m31* and *Data->TextMatrix.m32*, and will consider the text alignment.

Related Topics:

TosoCalcCurvature

TosoCalcBlockFrame

TosoCalcBlockFrame (Geometry and Conversion)

Contents Syntax

```
BOOL TosoCalcBlockFrame( int DrawingNum,
                         const LPSTR BlockName,
                         const LPSTR LibraryName,
                         const MATRIX* Matrix,
                         DRECT* Frame,
                         BOOL AutoLoad );
```

Calculates the surrounding frame of a block.

Contents Parameters

DrawingNum

[*int*] Zero-based index of the drawing in which the block is located (ignored if *LibraryName* is not equal to TVG_BLOCK_ID). Valid range: 0 <= Value < TVG_DRAWING_MAX.

BlockName

[*const LPSTR*] Name of the block whose surrounding frame is to be calculated.

LibraryName

[*const LPSTR*] Name of the library the block is located in.

Matrix

[*const MATRIX**] Matrix with which the block is to be temporarily multiplied before calculating the surrounding frame. If *Matrix* is NULL, no temporary modification is performed. The block itself will never be modified!

Frame

[*DRECT**] Address of a rectangle buffer to receive the calculated surrounding frame. If *Frame* is NULL, the frame will be calculated.

AutoLoad

[*BOOL*] This parameter determines whether or not the desired block shall be loaded to cache or not. If *AutoLoad* is set to FALSE, the return value of TosoCalcBlockFrame can be used to determine whether the desired block is already in cache or not.

Contents Return Value

Returns TRUE if successful. If FALSE is returned, the rectangle pointed to by *Frame* is not modified.

Contents Comment

This procedure can be used to check whether a specific block is available. In these cases, set both *Matrix* and *Frame* to NULL. If you want to check whether a block is already in the cache, set *AutoLoad* to FALSE. If you want to check whether a block is available anywhere in the currently opened libraries, set *AutoLoad* to TRUE. Please be aware of the fact that this will result in that block being loaded into the cache! So avoid to call this procedure within a real need to access that block.

Contents

Related Topics:

TosoCalcCurvature

TosoCalcTextFrame

TosoConvertDoubleString (Geometry and Conversion)

Contents Syntax

```
BOOL TosoConvertDoubleString( LPSTR Text,
                              double Value );
```

Converts a double value to a string using default settings for length and precision.

Contents Parameters

Text

[*LPSTR*] Text buffer to receive the converted double string. Allow at least NAME_LENGTH_SHORT characters!

Value

[*double*] Floating point value to be converted into a string.

Contents Return Value

Returns TRUE if successful, else FALSE.

Contents Comment

This conversion will use the decimal separator defined by the user (point or comma).

Contents

Related Topics:

Contents TosoConvertDoubleStringEx

Contents TosoConvertIntString

Contents TosoConvertLengthString

Contents TosoConvertWidthString

Contents TosoConvertAngleString

Contents TosoConvertStringDouble

Contents TosoConvertStringDoubleEx

TosoConvertDoubleStringEx (Geometry and Conversion)

Syntax

```
BOOL TosoConvertDoubleStringEx( LPSTR Text,
                                double Value,
                                int TotalLen,
                                int FractionLen );
```

Converts a double value to a string with a given maximum length and precision.

Parameters

Text
[*LPSTR*] Text buffer to receive the converted double string. Allow at least *TotalLen* characters!

Value
[*double*] Floating point value to be converted into a string.

TotalLen
[*int*] Maximum length of the converted string in characters.

FractionLen
[*int*] Maximum number of fractional digits. Allowed values are 0 to 9. Whether trailing zeros are displayed or not depends on the application's settings.

Return Value

Returns TRUE if successful, else FALSE.

Comment

This conversion will use the decimal separator defined by the user (point or comma).

Contents

Related Topics:

TosoConvertDoubleString

TosoConvertIntString

TosoConvertLengthString

TosoConvertWidthString

TosoConvertAngleString

TosoConvertIntString (Geometry and Conversion)

Contents Syntax

```
BOOL TosoConvertIntString( LPSTR Text,
                           int Value );
```

Converts an int value to a string.

Contents Parameters

Text

[*LPSTR*] Text buffer to receive the converted int string. Allow at least NAME_LENGTH_SHORT characters!

Value

[*int*] Integer value to be converted into a string.

Contents Return Value

Returns TRUE if successful, else FALSE.

Contents Comment

None.

Contents

Related Topics:

Contents TosoConvertDoubleString

Contents TosoConvertDoubleStringEx

Contents TosoConvertLengthString

Contents TosoConvertWidthString

Contents TosoConvertAngleString

Contents TosoConvertStringDouble

Contents TosoConvertStringDoubleEx

TosoConvertLengthString (Geometry and Conversion)

Contents

Syntax

```
BOOL TosoConvertLengthString( LPSTR Text,
                              double Value,
                              LPSTR Unit );
```

Converts a length value (based on the currently active length unit) to a string.

Contents

Parameters

Text

[*LPSTR*] Text buffer to receive the converted double string. Allow at least NAME_LENGTH_SHORT characters! If *Text* is NULL, the value will not be converted, only the current unit will be returned (see *Unit*).

Value

[*double*] Floating point value to be converted into a string.

Unit

[*LPSTR*] Text buffer to receive the currently active length unit in form of a short text (like `[mm]`). Allow at least 8 characters! If *Unit* is NULL, no unit text is returned.

Contents

Return Value

Returns TRUE if successful, else FALSE.

Contents

Comment

This conversion will use the decimal separator defined by the user (point or comma).

Contents

Related Topics:

Contents TosoConvertDoubleString
Contents TosoConvertDoubleStringEx
Contents TosoConvertIntString
Contents TosoConvertWidthString
Contents TosoConvertAngleString

TosoConvertWidthString (Geometry and Conversion)

Syntax

```
BOOL TosoConvertWidthString( LPSTR Text,
                             double Value,
                             LPSTR Unit );
```

Converts a width value (based on the currently active line unit) to a string.

Parameters

Text

[*LPSTR*] Text buffer to receive the converted double string. Allow at least NAME_LENGTH_SHORT characters! If *Text* is NULL, the value will not be converted, only the current unit will be returned (see *Unit*).

Value

[*double*] Floating point value to be converted into a string.

Unit

[*LPSTR*] Text buffer to receive the currently active line unit in form of a short text (like `[mm]`). Allow at least 8 characters! If *Unit* is NULL, no unit text is returned.

Return Value

Returns TRUE if successful, else FALSE.

Comment

This conversion will use the decimal separator defined by the user (point or comma).

Related Topics:

TosoConvertDoubleString

TosoConvertDoubleStringEx

TosoConvertIntString

TosoConvertLengthString

TosoConvertAngleString

TosoConvertAngleString (Geometry and Conversion)

Syntax

```
BOOL TosoConvertAngleString( LPSTR Text,
                             double Value,
                             LPSTR Unit );
```

Converts an angle value (based on the currently active angle unit) to a string.

Parameters

Text

[*LPSTR*] Text buffer to receive the converted double string. Allow at least NAME_LENGTH_SHORT characters! If *Text* is NULL, the value will not be converted, only the current unit will be returned (see *Unit*).

Value

[*double*] Floating point value to be converted into a string.

Unit

[*LPSTR*] Text buffer to receive the currently angle length unit in form of a short text (like `[mm]`). Allow at least 8 characters! If *Unit* is NULL, no unit text is returned.

Return Value

Returns TRUE if successful, else FALSE.

Comment

This conversion will use the decimal separator defined by the user (point or comma).

Contents

Related Topics:

TosoConvertDoubleString

TosoConvertDoubleStringEx

TosoConvertIntString

TosoConvertLengthString

TosoConvertWidthString

TosoConvertStringDouble (Geometry and Conversion)

Contents Syntax

```
BOOL TosoConvertStringDouble( double* Value,
                              const LPSTR Text,
                              HWND hParentWindow );
```

Converts a string's content into a double value, with terms like (0.5+2.0)/3 being evaluated.

Contents Parameters

Value

[*double**] Address of variable to receive the evaluated double value.

Text

[*const LPSTR*] Text containing the double value to be converted. This text may either include a single numeric value or a complete term like `(0.5+2.0)/3`. For a description of the term syntax see the description of the application's command Extra > Coordinate Entry (F8).

hParentWindow

[*HWND*] If this handle is not NULL, the corresponding window will be used as parent window of any error message occuring during term evaluation. If *hParentWindow* is NULL, no error message is displayed.

Contents Return Value

Returns TRUE if successful, else FALSE.

Contents Comment

This procedure will allow the full possible *double* value range. The assumed default units are `[mm]` for lengths and `[deg]` for angles.

Contents

Related Topics:

Contents TosoConvertDoubleString

Contents TosoConvertDoubleStringEx

Contents TosoConvertIntString

Contents TosoConvertLengthString

TosoConvertStringDoubleEx (Geometry and Conversion)

Contents Syntax

```
BOOL TosoConvertStringDoubleEx( double* Value,
                                const LPSTR Text,
                                HWND hParentWindow,
                                double LenMMPerUnit,
                                double AngleRadPerUnit );
```

Converts a string's content into a double value, with terms like (0.5+2.0)/3 being evaluated.

Contents Parameters

Value

[*double**] Address of variable to receive the evaluated double value.

Text

[*const LPSTR*] Text containing the double value to be converted. This text may either include a single numeric value or a complete term like `(0.5+2.0)/3`. For a description of the term syntax see the description of the application's command Extra > Coordinate Entry (F8).

hParentWindow

[*HWND*] If this handle is not NULL, the corresponding window will be used as parent window of any error message occuring during term evaluation. If *hParentWindow* is NULL, no error message is displayed.

LenMMPerUnit

[*double*] Definition of the default length unit in millimeters. For a default unit `[inch]`, set this value to 25.4 or REAL_INCH_MM respectively.

AngleRadPerUnit

[*double*] Definition of the default angle unit relative to radiant. For a default unit of `[rad]`, set this value to 1.0, for a default unit of `[deg]`, set this value to REAL_DEG_RAD.

Contents Return Value

Returns TRUE if successful, else FALSE.

Contents Comment

This procedure will allow the full possible *double* value range.

Contents

Related Topics:

Contents TosoConvertDoubleString

Contents TosoConvertDoubleStringEx

TosoConvertStringInt (Geometry and Conversion)

Contents Syntax

```
BOOL TosoConvertStringInt( int* Value,
                           const LPSTR Text );
```

Converts a string's content into an int value.

Contents Parameters

Value

[*int**] Address of variable to receive the evaluated int value.

Text

[*const LPSTR*] Text containing the int value to be converted. This text may include leading and trailing spaces.

Contents Return Value

Returns TRUE if successful, else FALSE.

Contents Comment

None.

Contents

Related Topics:

Contents TosoConvertDoubleString

Contents TosoConvertDoubleStringEx

Contents TosoConvertIntString

Contents TosoConvertLengthString

Contents TosoConvertWidthString

Contents TosoConvertAngleString

Contents TosoConvertStringDouble

TosoConvertStringLength (Geometry and Conversion)

Syntax

```
BOOL TosoConvertStringLength( double* Value,
                              const LPSTR Text,
                              HWND hParentWindow );
```

Converts a string's content into a length value, with terms like (0.5+2.0)/3 being evaluated.

Parameters

Value

[*double**] Address of variable to receive the evaluated length value.

Text

[*const LPSTR*] Text containing the length value to be converted. This text may either include a single numeric value or a complete term like `(0.5+2.0)/3`. For a description of the term syntax see the description of the application's command Extra > Coordinate Entry (F8).

hParentWindow

[*HWND*] If this handle is not NULL, the corresponding window will be used as parent window of any error message occuring during term evaluation. If *hParentWindow* is NULL, no error message is displayed.

Return Value

Returns TRUE if successful, else FALSE.

Comment

This procedure will allow a value range of COORD_MIN (-1e100) to COORD_MAX (1e100). The default units are the currently active units for lengths and angles.

Related Topics:

TosoConvertDoubleString

TosoConvertDoubleStringEx

TosoConvertIntString

TosoConvertLengthString

TosoConvertStringWidth (Geometry and Conversion)

Contents Syntax

```
BOOL TosoConvertStringWidth( double* Value,
                             const LPSTR Text,
                             HWND hParentWindow );
```

Converts a string's content into a width value, with terms like (0.5+2.0)/3 being evaluated.

Contents Parameters

Value

[*double**] Address of variable to receive the evaluated width value.

Text

[*const LPSTR*] Text containing the length value to be converted. This text may either include a single numeric value or a complete term like `(0.5+2.0)/3`. For a description of the term syntax see the description of the application's command Extra > Coordinate Entry (F8).

hParentWindow

[*HWND*] If this handle is not NULL, the corresponding window will be used as parent window of any error message occuring during term evaluation. If *hParentWindow* is NULL, no error message is displayed.

Contents Return Value

Returns TRUE if successful, else FALSE.

Contents Comment

This procedure will allow a value range of COORD_MIN (-1e100) to COORD_MAX (1e100). The default units are the currently active units for lines and angles.

Contents

Related Topics:

Contents TosoConvertDoubleString

Contents TosoConvertDoubleStringEx

Contents TosoConvertIntString

Contents TosoConvertLengthString

TosoConvertStringAngle (Geometry and Conversion)

Syntax

```
BOOL TosoConvertStringAngle( double* Value,
                             const LPSTR Text,
                             HWND hParentWindow );
```

Converts a string's content into an angle value, with terms like (0.5+2.0)/3 being evaluated.

Parameters

Value

[*double**] Address of variable to receive the evaluated angle value.

Text

[*const LPSTR*] Text containing the length value to be converted. This text may either include a single numeric value or a complete term like `(0.5+2.0)/3`. For a description of the term syntax see the description of the application's command Extra > Coordinate Entry (F8).

hParentWindow

[*HWND*] If this handle is not NULL, the corresponding window will be used as parent window of any error message occuring during term evaluation. If *hParentWindow* is NULL, no error message is displayed.

Return Value

Returns TRUE if successful, else FALSE.

Comment

This procedure will allow a value range of -EDIT_MIN (-1e10) to EDIT_MAX (1e10), the resulting value will be limited to the range 0 to REAL_2PI. The default units are the currently active units for lengths and angles.

Related Topics:

TosoConvertDoubleString

TosoConvertDoubleStringEx

TosoConvertIntString

TosoConvertLengthString

TosoConvertWidthString

TosoConvertAngleString

TosoConvertStringDouble

TosoConvertStringDoubleEx

TosoConvertStringInt

TosoConvertStringLength

TosoConvertStringWidth

TosoGeoDistance (Geometry and Conversion)

Contents Syntax

```
BOOL TosoGeoDistance( const GEO_OBJECT* GeoObj,
                      double x,
                      double y,
                      double* Result );
```

Calculates the distance (in internal millimeters) between the standard object and the given point.

Contents Parameters

GeoObj

[*const GEO_OBJECT**] Standard object to which the distance is to be calculated. Following standard object types are supported:

OBJ_LINE
: Straight line.

OBJ_CIRCLE
: Circle.

OBJ_ARC
: Circular arc.

OBJ_SECTOR
: Circular sector.

OBJ_SEGMENT
: Circular segment.

OBJ_ELLIPSE
: Ellipse.

OBJ_EARC
: Elliptical arc.

OBJ_ESECTOR
: Elliptical sector.

OBJ_ESEGMENT
: Elliptical segment.

OBJ_MARK
: Marking (optionally printed object), sometimes used to define a point.

OBJ_GEOLINE
: Geometry line (optionally printed object).

OBJ_GEOCIRCLE
: Geometry circle (optionally printed object).

OBJ_GEOELLIPSE
: Geometry ellipse (optionally printed object).

x

y

[*double*] Coordinates of the reference point in internal millimeters (relative to the page center, scale-independent).

Result

[*double**] Address of double buffer to receive the calculated curvature.

Return Value

Returns TRUE if successful. If FALSE is returned, *Result* is invalid.

Comment

None.

Related Topics:

TosoGeoIntersection

TosoGeoPerpendicular

TosoGeoTangent

TosoGeoRadiusFit

TosoGeoIncircle

TosoGeoCircumcircle

TosoGeoIntersection (Geometry and Conversion)

Contents

Syntax

```
int TosoGeoIntersection( const GEO_OBJECT* GeoObj1,
                         const GEO_OBJECT* GeoObj2,
                         DPOINT* Result );
```

Calculates the distance (in internal millimeters) between the standard object and the given point.

Contents

Parameters

GeoObj1

[*const GEO_OBJECT**] First standard object whose intersection points are to be calculated. Following standard object types are supported:

OBJ_LINE
: Straight line.

OBJ_CIRCLE
: Circle.

OBJ_ELLIPSE
: Ellipse.

OBJ_GEOLINE
: Geometry line (optionally printed object).

OBJ_GEOCIRCLE
: Geometry circle (optionally printed object).

OBJ_GEOELLIPSE
: Geometry ellipse (optionally printed object).

GeoObj2

[*const GEO_OBJECT**] Second standard object whose intersection points are to be calculated. Following standard object types are supported:

OBJ_LINE
: Straight line.

OBJ_CIRCLE
: Circle.

OBJ_ELLIPSE
: Ellipse.

OBJ_GEOLINE
: Geometry line (optionally printed object).

OBJ_GEOCIRCLE
: Geometry circle (optionally printed object).

OBJ_GEOELLIPSE
: Geometry ellipse (optionally printed object).

Result

[*DPOINT**] Address of point buffer to receive the calculated intersection points. Allow at least 16 points.

Return Value

Number of intersection points found.

Comment

None.

Related Topics:


TosoGeoDistance


TosoGeoPerpendicular


TosoGeoTangent


TosoGeoRadiusFit


TosoGeoIncircle


TosoGeoCircumcircle

TosoGeoPerpendicular (Geometry and Conversion)

Contents

Syntax

```
int TosoGeoPerpendicular( const GEO_OBJECT* GeoObj,
                          double x,
                          double y,
                          DPOINT* Result );
```

Drops a perpendicular from a point onto a standard object and calculates the resulting perpendicular base points (i.e. end-points of the perpendicular).

Contents

Parameters

GeoObj

[*const GEO_OBJECT**] Standard object onto which the perpendicular is to be dropped. Following standard object types are supported:

OBJ_LINE
: Straight line.

OBJ_CIRCLE
: Circle.

OBJ_ELLIPSE
: Ellipse.

OBJ_GEOLINE
: Geometry line (optionally printed object).

OBJ_GEOCIRCLE
: Geometry circle (optionally printed object).

OBJ_GEOELLIPSE
: Geometry ellipse (optionally printed object).

x

y

[*double*] Coordinates of the reference point in internal millimeters (relative to the page center, scale-independent).

Result

[*DPOINT**] Address of point buffer to receive the calculated perpendicular base points. Allow at least 16 points.

Contents

Return Value

Number of perpendicular base points found.

Contents

Comment

None.

Related Topics:

TosoGeoDistance

TosoGeoIntersection

TosoGeoTangent

TosoGeoRadiusFit

TosoGeoIncircle

TosoGeoCircumcircle

TosoGeoTangent (Geometry and Conversion)

Contents Syntax

```
int TosoGeoTangent( const GEO_OBJECT* GeoObj1,
                    const GEO_OBJECT* GeoObj2,
                    DPOINT* Result );
```

Calculates all tangent lines between two standard objects.

Contents Parameters

GeoObj1

[*const GEO_OBJECT**] First standard object to which the tangents are to be calculated. Following standard object types are supported:

OBJ_LINE
: Straight line.

OBJ_CIRCLE
: Circle.

OBJ_ELLIPSE
: Ellipse.

OBJ_MARK
: Marking (optionally printed object), sometimes used to define a point.

OBJ_GEOLINE
: Geometry line (optionally printed object).

OBJ_GEOCIRCLE
: Geometry circle (optionally printed object).

OBJ_GEOELLIPSE
: Geometry ellipse (optionally printed object).

GeoObj2

[*const GEO_OBJECT**] Second standard object to which the tangents are to be calculated. Following standard object types are supported:

OBJ_LINE
: Straight line.

OBJ_CIRCLE
: Circle.

OBJ_ELLIPSE
: Ellipse.

OBJ_MARK
: Marking (optionally printed object), sometimes used to define a point.

OBJ_GEOLINE
: Geometry line (optionally printed object).

OBJ_GEOCIRCLE
: Geometry circle (optionally printed object).

OBJ_GEOELLIPSE
Geometry ellipse (optionally printed object).

Result
[*DPOINT**] Address of point buffer to receive the calculated tangent's end-points. Allow at least 16 points.

Return Value

Number of tangent points found. This value is always a multiple of 2, as for each found tangent a pair of two points is returned. The first point of each pair lies at *GeoObj2*, the second point at *GeoObj2*.

Comment

None.

Related Topics:


TosoGeoDistance


TosoGeoIntersection


TosoGeoPerpendicular


TosoGeoRadiusFit


TosoGeoIncircle


TosoGeoCircumcircle

TosoGeoRadiusFit (Geometry and Conversion)

Contents Syntax

```
int TosoGeoRadiusFit( const GEO_OBJECT* GeoObj1,
                      const GEO_OBJECT* GeoObj2,
                      double Radius,
                      int ObjType,
                      GEO_OBJECT* Result );
```

Calculates circles or circular arcs with a given radius so that they tangent the two given objects.

Contents Parameters

GeoObj1

[*const GEO_OBJECT**] First standard object which the calculated object shall be tangenting. Following standard object types are supported:

OBJ_LINE
: Straight line.

OBJ_CIRCLE
: Circle.

OBJ_MARK
: Marking (optionally printed object), sometimes used to define a point.

OBJ_GEOLINE
: Geometry line (optionally printed object).

OBJ_GEOCIRCLE
: Geometry circle (optionally printed object).

GeoObj2

[*const GEO_OBJECT**] Second standard object which the calculated object shall be tangenting. Following standard object types are supported:

OBJ_LINE
: Straight line.

OBJ_CIRCLE
: Circle.

OBJ_MARK
: Marking (optionally printed object), sometimes used to define a point.

OBJ_GEOLINE
: Geometry line (optionally printed object).

OBJ_GEOCIRCLE
: Geometry circle (optionally printed object).

Radius

[*double*] Desired radius of the calculated circles or circular arcs.

ObjType

[*int*] Desired object type of the calculated objects. Following standard object types are supported:

OBJ_CIRCLE

Circle.

OBJ_ARC
Circular arc.

OBJ_SECTOR
Circular sector.

OBJ_SEGMENT
Circular segment.

OBJ_GEOCIRCLE
Geometry circle (optionally printed object).

If the desired object type is either OBJ_ARC, OBJ_SECTOR or OBJ_SEGMENT, the serving application will first calculate circles that fit the given conditions and will then split each circle into two parts that each start and end at the tangent points. As a result, circle parts will always be returned in pairs.

Result
[*GEO_OBJECT**] Address of object buffer to receive the calculated objects. Allow at least 16 objects.

Return Value

Number of objects found. The type of objects returned is determined by the *ObjType* parameter.

Comment

None.

Related Topics:


TosoGeoDistance


TosoGeoIntersection


TosoGeoPerpendicular


TosoGeoTangent


TosoGeoIncircle


TosoGeoCircumcircle

TosoGeoIncircle (Geometry and Conversion)

Syntax

```
int TosoGeoIncircle( const DPOINT* Point1,
                     const DPOINT* Point2,
                     const DPOINT* Point3,
                     DPOINT* Result );
```

Calculates the incircle's center of a triangle.

Parameters

Point1
Point2
Point3
[*const DPOINT**] Three corner-points of the triangle whose incircle's center-point is to be calculated.
Result
[*DPOINT**] Address of point buffer to receive the calculated incircle's center-point. Allow at least 1 point.

Return Value

Number of center-points points found. This value is either 0 or 1.

Comment

None.

Related Topics:


TosoGeoDistance


TosoGeoIntersection


TosoGeoPerpendicular


TosoGeoTangent


TosoGeoRadiusFit

Contents

TosoGeoCircumcircle (Geometry and Conversion)

Syntax

```
int TosoGeoCircumcircle( const DPOINT* Point1,
                         const DPOINT* Point2,
                         const DPOINT* Point3,
                         DPOINT* Result );
```

Calculates the circumcircle's center of a triangle.

Parameters

Point1
Point2
Point3
[*const DPOINT**] Three corner-points of the triangle whose circumcircle's center-point is to be calculated.

Result
[*DPOINT**] Address of point buffer to receive the calculated circumcircle's center-point. Allow at least 1 point.

Return Value

Number of center-points points found. This value is either 0 or 1.

Comment

None.

Related Topics:

TosoGeoDistance

TosoGeoIntersection

TosoGeoPerpendicular

TosoGeoTangent

TosoGeoRadiusFit

Contents

TosoMatrixSeparate (Geometry and Conversion)

Contents

Syntax

```
BOOL TosoMatrixSeparate( const MATRIX* Matrix,
                         double* ScaleX,
                         double* ScaleY,
                         double* Rotation,
                         double* Distortion,
                         double* MoveX,
                         double* MoveY );
```

Separates a matrix into its basic transformational elements.

Contents

Parameters

Matrix
[*const MATRIX**] Address of the matrix to be separated.

ScaleX
[*double**] Address of value to receive the horizontal scaling factor.

ScaleY
[*double**] Address of value to receive the vertical scaling factor.

Rotation
[*double**] Address of value to receive the rotation angle in [rad].

Distortion
[*double**] Address of value to receive the distortion angle in [rad].

MoveX
[*double**] Address of value to receive the horizontal movement.

MoveY
[*double**] Address of value to receive the vertical movement.

Contents

Return Value

Returns TRUE if successful. If FALSE is returned, the matrix could not be separated. In this case, all values but *MoveX* and *MoveY* will be set to 0.0.

Contents

Comment

The resulting values can be use to retrieve and modify a specific transformation stored in the matrix, and then be passed to TosoMatrixAssemble to create a matrix based on the modified transformation.

Contents

Related Topics:

Contents TosoMatrixAssemble

TosoMatrixAssemble (Geometry and Conversion)

Contents

Syntax

```
BOOL TosoMatrixAssemble( MATRIX* Matrix,
                         double ScaleX,
                         double ScaleY,
                         double Rotation,
                         double Distortion,
                         double MoveX,
                         double MoveY );
```

Assembles a matrix out of its basic transformation elements.

Contents

Parameters

Matrix
[*MATRIX**] Address of the matrix to be filled.

ScaleX
[*double*] Horizontal scaling factor.

ScaleY
[*double*] Vertical scaling factor.

Rotation
[*double*] Rotation angle in [rad].

Distortion
[*double*] Distortion angle in [rad].

MoveX
[*double*] Horizontal movement.

MoveY
[*double*] Vertical movement.

Contents

Return Value

Returns TRUE if successful, else FALSE.

Contents

Comment

To easily analize and modify a given matrix, first separate it by means of the TosoMatrixSeparate procedure and then pass modified values to TosoMatrixAssemble to create the modified matrix.

Contents

Related Topics:

Contents TosoMatrixSeparate

Contents TosoMatrixInvert

TosoMatrixInvert (Geometry and Conversion)

Syntax

```
BOOL TosoMatrixInvert( const MATRIX* Matrix,
                       MATRIX* InvertMatrix );
```

Inverts the given matrix.

Parameters

Matrix

[*const MATRIX**] Matrix to be inverted.

InvertMatrix

[*MATRIX**] Address of the matrix to receive the inverted matrix. This address may be equal to *Matrix*.

Return Value

Returns TRUE if successful. If FALSE is returned, the matrix pointed to by *InvertMatrix* is not modified.

Comment

None.

Related Topics:

TosoMatrixSeparate

TosoMatrixAssemble

TosoMatrixInit

TosoMatrixRotate

TosoMatrixMove

TosoMatrixScale

TosoMatrixSheer

Contents

TosoMatrixInit (Geometry and Conversion)

Contents **Syntax**

```
void TosoMatrixInit( MATRIX* Matrix );
```

Initialises the given matrix, i.e. fills it with the values (1.0, 0.0, 0.0, 1.0, 0.0, 0.0).

Contents **Parameters**

Matrix

[*MATRIX**] Address of the matrix to be initialised.

Contents **Return Value**

None.

Contents **Comment**

None.

Contents

Related Topics:

Contents TosoMatrixSeparate

Contents TosoMatrixAssemble

Contents TosoMatrixInvert

Contents TosoMatrixRotate

Contents TosoMatrixMove

Contents TosoMatrixScale

Contents TosoMatrixSheer

Contents TosoMatrixMultiply

TosoMatrixRotate (Geometry and Conversion)

Syntax

```
void TosoMatrixRotate( MATRIX* Matrix,
                       double Angle );
```

Modifies the given matrix by applying a rotation of *Angle* relative to the origin.

Parameters

Matrix

[*MATRIX**] Matrix to be modified.

Angle

[*double*] Rotation angle in [rad]. Valid range: -4pi <= Value <= 4pi.

Return Value

None.

Comment

None.

Related Topics:

TosoMatrixSeparate

TosoMatrixAssemble

TosoMatrixInvert

TosoMatrixInit

TosoMatrixMove

TosoMatrixScale

TosoMatrixSheer

Contents

TosoMatrixMove (Geometry and Conversion)

Syntax

```
void TosoMatrixMove( MATRIX* Matrix,
                     double OffsetX,
                     double OffsetY );
```

Modifies the given matrix by applying a movement of (*OffsetX*, *OffsetY*).

Parameters

Matrix
[*MATRIX**] Matrix to be modified.

OffsetX
OffsetY
[*double*] Horizontal and vertical movement in internal millimeters. Valid range: COORD_MIN <= Value <= COORD_MAX.

Return Value

None.

Comment

None.

Related Topics:

TosoMatrixSeparate

TosoMatrixAssemble

TosoMatrixInvert

TosoMatrixInit

TosoMatrixRotate

TosoMatrixScale

TosoMatrixScale (Geometry and Conversion)

Syntax

```
void TosoMatrixScale( MATRIX* Matrix,
                      double FactorX,
                      double FactorY );
```

Modifies the given matrix by applying a scaling of (*FactorX*, *FactorY*).

Parameters

Matrix

[*MATRIX**] Matrix to be modified.

FactorX

FactorY

[*double*] Horizontal and vertical scaling factors. Valid range: COORD_MIN <= Value <= COORD_MAX.

Return Value

None.

Comment

None.

Related Topics:

TosoMatrixSeparate

TosoMatrixAssemble

TosoMatrixInvert

TosoMatrixInit

TosoMatrixRotate

TosoMatrixMove

TosoMatrixSheer (Geometry and Conversion)

Contents Syntax

```
void TosoMatrixSheer( MATRIX* Matrix,
                      double FactorX,
                      double FactorY );
```

Modifies the given matrix by applying a sheering of (*FactorX*, *FactorY*).

Contents Parameters

Matrix

[*MATRIX**] Matrix to be modified.

FactorX

FactorY

[*double*] Horizontal and vertical sheering factors. Valid range: COORD_MIN <= Value <= COORD_MAX.

Contents Return Value

None.

Contents Comment

None.

Contents

Related Topics:

Contents TosoMatrixSeparate

Contents TosoMatrixAssemble

Contents TosoMatrixInvert

Contents TosoMatrixInit

Contents TosoMatrixRotate

Contents TosoMatrixMove

TosoMatrixMultiply (Geometry and Conversion)

Syntax

```
void TosoMatrixMultiply( MATRIX* Matrix,
                         const MATRIX* Factor );
```

Modifies the given matrix by applying another matrix to it.

Parameters

Matrix

[*MATRIX**] Matrix to be modified.

Factor

[*const MATRIX**] Matrix to be applied to the given matrix.

Return Value

None.

Comment

None.

Related Topics:

TosoMatrixSeparate

TosoMatrixAssemble

TosoMatrixInvert

TosoMatrixInit

TosoMatrixRotate

TosoMatrixMove

TosoMatrixScale

Contents

TosoGetCommandTitle (Miscellaneous)

Syntax

```
BOOL TosoGetCommandTitle( LPSTR Text,
                          int CommandID );
```

Retrieves the textual descrition of a given command.

Parameters

Text

[*LPSTR*] Text buffer to receive the command's title. Allow at least NAME_LENGTH_BLOCK characters!

CommandID

[*int*] Identification of the command whose title shall be returned. For a list of all command IDs, see Command IDs.

Return Value

Returns TRUE if successful, else FALSE.

Comment

None.

Related Topics:

TosoGetCommandIcon

TosoGetStandardIcon

TosoDrawingGetNumber

TosoDrawingGetInfo

TosoLibraryGetNumber

TosoLibraryGetInfo

TosoGetCommandIcon (Miscellaneous)

Contents Syntax

```
HBITMAP TosoGetCommandIcon( int CommandID );
```

Retrieves the icon bitmap of a given command.

Contents Parameters

CommandID

[*int*] Identification of the command whose icon bitmap be returned. For a list of all command IDs, see Command IDs.

Contents Return Value

Returns a bitmap handle of the command's icon bitmap. This bitmap is a monochrone bitmap of 48 by 40 pixels in size, the occupied area is 40 by 40 pixels. If *CommandID* is invalid, the returned handle is NULL.

Contents Comment

The module is responsible for freeing the bitmap identified by the handle as soon as it does not require it any more by calling the Win32 command `DeleteBitmap` .

Contents

Related Topics:

Contents TosoGetCommandTitle

Contents TosoGetStandardIcon

Contents TosoDrawingGetNumber

Contents TosoDrawingGetInfo

Contents TosoLibraryGetNumber

Contents TosoLibraryGetInfo

TosoGetStandardIcon (Miscellaneous)

Contents

Syntax

```
HBITMAP TosoGetStandardIcon( const LPSTR IconID );
```

Retrieves a standard icon from the serving application.

Contents

Parameters

IconID

[*const LPSTR*] Identification string of the icon to be retrieved. Available icon identification strings are:

`IDI_ORIGIN`
Icon used to display the current origin (crossmark).

`IDI_ARROW0` .. `IDI_ARROW6`
Icons for the seven possible arrow types in dimension (none, arrow, diagonal line, circle, etc.).

`IDI_DIMLINE0` .. `IDI_DIMLINE2`
Icons for the three possible dimension/extension line modes in dimensions (vertical extension line, non-vertical extension line, partial dimension line).

`IDI_DISTMOD0` .. `IDI_DISTMOD2`
Icons for the three possible dimension line distance modes in dimensions (defined by point or numerical).

`IDI_ENDCAP0` .. `IDI_ENDCAP2`
Icons for the three possible line cap modes (round, square, flat).

`IDI_EXTLINE0` .. `IDI_EXTLINE1`
Icons for the two possible extension line modes in arc length dimensions (radial, parallel).

`IDI_FILLMOD0` .. `IDI_FILLMOD4`
Icons for the three possible filling modes of objects (framed, filled, eraser, etc.).

`IDI_GRIDMOD0` .. `IDI_GRIDMOD4`
Icons for the five possible grid modes (none, cartesian, isometric, dimetric 1 and 2).

`IDI_INFOMOD0` .. `IDI_INFOMOD3`
Icons for the four possible modes of the "edit properties" dialog window (object properties, text properties, dimension line properties, dimension text properties).

`IDI_JOIN0` .. `IDI_JOIN2`
Icons for the three possible line join modes (round, bevel, miter).

`IDI_NUMBER0` .. `IDI_NUMBER2`
Icons for the three possible number display modes in dimensions (all fractional digits, at least one fractional digit, minimum number of fractional digits).

`IDI_ORIENT0` .. `IDI_ORIENT3`
Icons for the four possible dimension line orientation modes for dimensions (parallel, vertical, horizontal, defined by point).

`IDI_PAGEMOD0` .. `IDI_PAGEMOD2`

Icons for the three possible page orientation modes (undefined, portrait, landscape).

`IDI_RFARROW0` .. `IDI_RFARROW6`
Icons for the seven possible reference arrow line modes for reference texts (straight, bend 45°, bend 90°).

`IDI_RFFRAME0` .. `IDI_RFFRAME3`
Icons for the four possible frame types for reference texts (rectangle, quadrangle, circle, ellipse).

`IDI_TEXTMOD0` .. `IDI_TEXTMOD3`
Icons for the four possilbe text orientation modes (left, centered, right, justified).

`IDI_WINMODE0` .. `IDI_WINMODE7`
Icons for the eight possible drawing window arrangements (one window, two window besides, two windows beneath, etc.)

Please note that these identifiers are not constants but *strings*, i.e. they have to be stated in quotes!

Return Value

Returns an icon handle of the desired standard icon. This icon is always a 32 x 32 pixel, 16 color icon. If *IconID* is invalid, the returned handle is NULL.

Comment

As these icons are defined in the language-dependent DLL of the serving application, it is *not* possible to retrieve these icon handles by a direct call to `LoadIcon` using the *hMainInst* passed to the `TosoModuleInit` procedure!

Related Topics:

TosoGetCommandTitle

TosoGetCommandIcon

TosoDrawingGetNumber

TosoDrawingGetInfo

TosoLibraryGetNumber

TosoLibraryGetInfo

TosoDrawingGetNumber (Miscellaneous)

Contents **Syntax**

```
int TosoDrawingGetNumber( void );
```

Retrieves the number of drawings that are currently loaded.

Contents **Parameters**

None.

Contents **Return Value**

Returns the number of drawings loaded.

Contents **Comment**

None.

Contents

Related Topics:

Contents TosoGetCommandTitle

Contents TosoGetCommandIcon

Contents TosoGetStandardIcon

Contents TosoDrawingGetInfo

Contents TosoLibraryGetNumber

Contents TosoLibraryGetInfo

TosoDrawingGetInfo (Miscellaneous)

Contents Syntax

```
BOOL TosoDrawingGetInfo( int LibraryNum,
                         LPSTR FileName,
                         FILE_HEADER* FileHeader );
```

Retrieves information about a drawing. Especially the drawing's title (located in its header) is used to identify that drawing.

Contents Parameters

DrawingNum

[*int*] Zeo-based index of the drawing whose title shall be retrieved. Valid range: 0 <= Value < TVG_DRAWING_MAX.

FileName

[*LPSTR*] Address of buffer to receive the drawing's file name. Allow at least MAX_PATH characters. If *FileName* is NULL, no file name will be returned.

FileHeader

[*FILE_HEADER**] Address of buffer to receive the drawing's header. If *FileHeader* is NULL, no header will be returned.

Contents Return Value

Returns TRUE if successful. If FALSE is returned, the given *DrawingNum* was invalid.

Contents Comment

None.

Contents

Related Topics:

Contents TosoGetCommandTitle

Contents TosoGetCommandIcon

Contents TosoGetStandardIcon

Contents TosoDrawingGetNumber

Contents TosoLibraryGetNumber

Contents

TosoLibraryGetNumber (Miscellaneous)

Contents **Syntax**

```
int TosoLibraryGetNumber( void );
```

Retrieves the number of libraries that are currently loaded (including those that are cached).

Contents **Parameters**

None.

Contents **Return Value**

Returns the number of libraries available.

Contents **Comment**

None.

Contents

Related Topics:

Contents TosoGetCommandTitle

Contents TosoGetCommandIcon

Contents TosoGetStandardIcon

Contents TosoDrawingGetNumber

Contents TosoDrawingGetInfo

Contents TosoLibraryGetInfo

TosoLibraryGetInfo (Miscellaneous)

Syntax

```
BOOL TosoLibraryGetInfo( int LibraryNum,
                         LPSTR FileName,
                         FILE_HEADER* FileHeader );
```

Retrieves information about a library. Especially the library's title (located in its header) is used to identify that library inside instances.

Parameters

LibraryNum

[*int*] Zeo-based index of the library whose title shall be retrieved.

FileName

[*LPSTR*] Address of buffer to receive the library's file name. Allow at least MAX_PATH characters. If *FileName* is NULL, no file name will be returned.

FileHeader

[*FILE_HEADER**] Address of buffer to receive the library's header. If *FileHeader* is NULL, no header will be returned.

Return Value

Returns TRUE if successful. If FALSE is returned, the given *LibraryNum* was invalid.

Comment

None.

Related Topics:

TosoGetCommandTitle

TosoGetCommandIcon

TosoGetStandardIcon

TosoDrawingGetNumber

TosoDrawingGetInfo

Contents

TosoImportGetNumber (Miscellaneous)

Syntax

```
int TosoImportGetNumber( void );
```

Determines the number of import filters that are currently loaded.

Parameters

None.

Return Value

Number of import filters loaded.

Comment

None.

Related Topics:

TosoImportGetData

TosoExportGetNumber

TosoExportGetData

TosoModuleGetNumber

TosoModuleGetData

TosoImportGetData (Miscellaneous)

Contents

Syntax

```
BOOL TosoImportGetData( int Index,
                        MODULE_DATA* ModuleData );
```

Returns an information structure of a currently loaded import filter.

Contents

Parameters

Index

[*int*] Index of the import filter whose information data is to be returned. Valid range: 0 <= Value <= TVG_FILTER_MAX.

ModuleData

[*MODULE_DATA**] Address of a structure that is to be filled with the information about the import filter identified by *Index*.

Contents

Return Value

Returns TRUE if the information data was copied successfully, else FALSE (indicating an invalid value of *Index* or *ModuleData*).

Contents

Comment

None.

Contents

Related Topics:

Contents TosoImportGetNumber
Contents TosoExportGetNumber
Contents TosoExportGetData
Contents TosoModuleGetNumber
Contents TosoModuleGetData

TosoExportGetNumber (Miscellaneous)

Syntax

```
int TosoExportGetNumber( void );
```

Determines the number of export filters that are currently loaded.

Parameters

None.

Return Value

Number of export filters loaded.

Comment

None.

Related Topics:

 TosoImportGetNumber

 TosoImportGetData

 TosoExportGetData

 TosoModuleGetNumber

 TosoModuleGetData

TosoExportGetData (Miscellaneous)

Syntax

```
BOOL TosoExportGetData( int Index,
                        MODULE_DATA* ModuleData );
```

Returns an information structure of a currently loaded export filter.

Parameters

Index

[*int*] Index of the export filter whose information data is to be returned. Valid range: 0 <= Value <= TVG_FILTER_MAX.

ModuleData

[*MODULE_DATA**] Address of a structure that is to be filled with the information about the export filter identified by *Index*.

Return Value

Returns TRUE if the information data was copied successfully, else FALSE (indicating an invalid value of *Index* or *ModuleData*).

Comment

None.

Related Topics:

TosoImportGetNumber

TosoImportGetData

TosoExportGetNumber

TosoModuleGetNumber

TosoModuleGetData

TosoModuleGetNumber (Miscellaneous)

Contents **Syntax**

```
int TosoModuleGetNumber( void );
```

Determines the number of modules that are currently loaded.

Contents **Parameters**

None.

Contents **Return Value**

Number of modules loaded.

Contents **Comment**

None.

Contents

Related Topics:

Contents TosoImportGetNumber

Contents TosoImportGetData

Contents TosoExportGetNumber

Contents TosoExportGetData

Contents TosoModuleGetData

TosoModuleGetData (Miscellaneous)

Syntax

```
BOOL TosoModuleGetData( int Index,
                        MODULE_DATA* ModuleData );
```

Returns an information structure of a currently loaded module.

Parameters

Index

[*int*] Index of the module whose information data is to be returned. Valid range: 0 <= Value <= TVG_MODULE_MAX.

ModuleData

[*MODULE_DATA**] Address of a structure that is to be filled with the information about the module identified by *Index*.

Return Value

Returns TRUE if the information data was copied successfully, else FALSE (indicating an invalid value of *Index* or *ModuleData*).

Comment

None.

Related Topics:

TosoImportGetNumber

TosoImportGetData

TosoExportGetNumber

TosoExportGetData

TosoModuleGetNumber

TosoInitProperty (Miscellaneous)

Contents **Syntax**

```
void TosoInitProperty( PROPERTY* Data );
```

Initializes a standard property set (without pen, transmission, and layer).

Contents **Parameters**

Data

[*PROPERTY**] Address of a property set to be initialized.

Contents **Return Value**

None.

Contents **Comment**

None.

Contents

Related Topics:

Contents TosoInitXProperty

Contents TosoInitFontDef

Contents TosoInitDimLine

Contents TosoInitDimSmall

Contents TosoInitDimLarge

Contents TosoInitTextStandard

Contents TosoInitTextFrame

TosoInitXProperty (Miscellaneous)

Syntax

```
void TosoInitXProperty( XPROPERTY* Data );
```

Initializes an extended property set (including pen, transmission, and layer).

Parameters

Data

[*XPROPERTY**] Address of an extended property set to be initialized.

Return Value

None.

Comment

None.

Related Topics:

TosoInitProperty

TosoInitFontDef

TosoInitDimLine

TosoInitDimSmall

TosoInitDimLarge

TosoInitTextStandard

TosoInitTextFrame

TosoInitFontDef (Miscellaneous)

Contents **Syntax**

```
void TosoInitFontDef( FONTDEF* Data );
```

Initializes a font definition.

Contents **Parameters**

Data

[*FONTDEF**] Address of a font definition to be initialized.

Contents **Return Value**

None.

Contents **Comment**

None.

Contents

Related Topics:

Contents TosoInitProperty

Contents TosoInitXProperty

Contents TosoInitDimLine

Contents TosoInitDimSmall

Contents TosoInitDimLarge

Contents TosoInitTextStandard

Contents TosoInitTextFrame

TosoInitDimLine (Miscellaneous)

Contents **Syntax**

```
void TosoInitDimLine( DIMLINE* Data );
```

Initializes a dimension line parameter set.

Contents **Parameters**

Data

[*DIMLINE**] Address of a dimension line parameter set to be initialized.

Contents **Return Value**

None.

Contents **Comment**

None.

Contents

Related Topics:

Contents TosoInitProperty

Contents TosoInitXProperty

Contents TosoInitFontDef

Contents TosoInitDimSmall

Contents TosoInitDimLarge

Contents TosoInitTextStandard

Contents TosoInitTextFrame

TosoInitDimSmall (Miscellaneous)

Syntax

```
void TosoInitDimSmall( DIMSMALL* Data,
                       BOOL UseGlobal );
```

Initializes a small dimension parameter set.

Parameters

Data

[*DIMSMALL**] Address of a small dimension parameter set to be initialized.

UseGlobal

[*BOOL*] If this value is TRUE, the elements *TextFont*, *TextSize1*, *TextSize2*, *NumAccuracy* and *System* of the structure pointed to by *Data* will be initialized with the current global dimension parameters.

Return Value

None.

Comment

None.

Related Topics:

TosoInitProperty

TosoInitXProperty

TosoInitFontDef

TosoInitDimLine

TosoInitDimLarge

TosoInitTextStandard

TosoInitTextFrame

TosoInitDimLarge (Miscellaneous)

Syntax

```
void TosoInitDimLarge( DIMLARGE* Data,
                       BOOL UseGlobal );
```

Initializes a large dimension parameter set.

Parameters

Data

[*DIMLARGE**] Address of a large dimension parameter set to be initialized.

UseGlobal

[*BOOL*] If this value is TRUE, the elements *TextFont*, *TextSize1*, *TextSize2*, *NumAccuracy* and *System* of the structure pointed to by *Data* will be initialized with the current global dimension parameters.

Return Value

None.

Comment

None.

Related Topics:

TosoInitProperty

TosoInitXProperty

TosoInitFontDef

TosoInitDimLine

TosoInitDimSmall

TosoInitTextStandard

TosoInitTextFrame

TosoInitTextStandard (Miscellaneous)

Contents **Syntax**

```
void TosoInitTextStandard( TEXTSTANDARD* Data1,
                           TEXTREFERENCE* Data2 );
```

Initializes a standard text parameter set.

Contents **Parameters**

Data1

[*TEXTSTANDARD**] Address of a standard text parameter set to be initialized. If this address is NULL, the standard text parameters will not be initialized.

Data2

[*TEXTREFERENCE**] Address of a reference text parameter set to be initialized.. If this address is NULL, the reference text parameters will not be initialized.

Contents **Return Value**

None.

Contents **Comment**

None.

Contents

Related Topics:

Contents TosoInitProperty

Contents TosoInitXProperty

Contents TosoInitFontDef

Contents TosoInitDimLine

Contents TosoInitDimSmall

Contents TosoInitDimLarge

Contents

TosoInitTextFrame (Miscellaneous)

Contents **Syntax**

```
void TosoInitTextFrame( TEXTFRAME* Data );
```

Initializes a frame text parameter set.

Contents **Parameters**

Data

[*TEXTFRAME**] Address of a frame text parameter set to be initialized.

Contents **Return Value**

None.

Contents **Comment**

None.

Contents

Related Topics:

Contents TosoInitProperty

Contents TosoInitXProperty

Contents TosoInitFontDef

Contents TosoInitDimLine

Contents TosoInitDimSmall

Contents TosoInitDimLarge

Contents TosoInitTextStandard

TosoHookPositionStart (Miscellaneous)

Syntax

```
BOOL TosoHookPositionStart( TOSOHOOKPOSITION_PROC CallBack );
```

Establishes a hook on position calculation. As a result, the given *CallBack* procedure will be called each time the user moves the cursor, allowing the module to view and alter the resulting cursor coordinates.

Parameters

CallBack

[*TOSOHOOKPOSITION_PROC*] Address of the callback procedure to be enabled.

Return Value

Returns TRUE is the hook has been established successfully, else FALSE.

Comment

Each module should not establish more than one position hook at a time, as the total number of hooks available is limited!

Related Topics:

TosoHookPositionEnd

TosoHookMouseStart

TosoHookMouseEnd

TosoHookKeyStart

TosoHookKeyEnd

TosoHookPositionEnd (Miscellaneous)

Syntax

```
void TosoHookPositionEnd( TOSOHOOKPOSITION_PROC CallBack );
```

Removes a hook on position calculation. As a result, the given *CallBack* procedure will no longer be called when the user moves the cursor.

Parameters

CallBack

[*TOSOHOOKPOSITION_PROC*] Address of the callback procedure to be disabled.

Return Value

None.

Comment

Each module should not establish more than one position hook at a time, as the total number of hooks available is limited!

Related Topics:

TosoHookPositionStart

TosoHookMouseStart

TosoHookMouseEnd

TosoHookKeyStart

TosoHookKeyEnd

TosoHookMouseStart (Miscellaneous)

Syntax

```
BOOL TosoHookMouseStart( TOSOHOOKMOUSE_PROC CallBack );
```

Establishes a hook on mouse events. As a result, the given *CallBack* procedure will be called each time a mouse event occurs.

Parameters

CallBack

[*TOSOHOOKMOUSE_PROC*] Address of the callback procedure to be enabled.

Return Value

Returns TRUE is the hook has been established successfully, else FALSE.

Comment

Each module should not establish more than one mouse hook at a time, as the total number of hooks available is limited!

Related Topics:

TosoHookPositionStart

TosoHookPositionEnd

TosoHookMouseEnd

TosoHookKeyStart

TosoHookKeyEnd

TosoHookMouseEnd (Miscellaneous)

Contents **Syntax**

```
void TosoHookMouseEnd( TOSOHOOKMOUSE_PROC CallBack );
```

Removes a hook on mouse events. As a result, the given *CallBack* procedure will no longer be called when a mouse event occurs.

Contents **Parameters**

CallBack

[*TOSOHOOKMOUSE_PROC*] Address of the callback procedure to be disabled.

Contents **Return Value**

None.

Contents **Comment**

Each module should not establish more than one mouse hook at a time, as the total number of hooks available is limited!

Contents

Related Topics:

Contents TosoHookPositionStart

Contents TosoHookPositionEnd

Contents TosoHookMouseStart

Contents TosoHookKeyStart

Contents TosoHookKeyEnd

TosoHookKeyStart (Miscellaneous)

Syntax

```
BOOL TosoHookKeyStart( TOSOHOOKKEY_PROC CallBack );
```

Establishes a hook on key events. As a result, the given *CallBack* procedure will be called each time a key event occurs.

Parameters

CallBack

[*TOSOHOOKKEY_PROC*] Address of the callback procedure to be enabled.

Return Value

Returns TRUE is the hook has been established successfully, else FALSE.

Comment

Each module should not establish more than one key hook at a time, as the total number of hooks available is limited!

Related Topics:

TosoHookPositionStart

TosoHookPositionEnd

TosoHookMouseStart

TosoHookMouseEnd

TosoHookKeyEnd

TosoHookKeyEnd (Miscellaneous)

Syntax

```
void TosoHookKeyEnd( TOSOHOOKKEY_PROC CallBack );
```

Removes a hook on key events. As a result, the given *CallBack* procedure will no longer be called when a key event occurs.

Parameters

CallBack

[*TOSOHOOKKEY_PROC*] Address of the callback procedure to be disabled.

Return Value

None.

Comment

Each module should not establish more than one key hook at a time, as the total number of hooks available is limited!

Related Topics:

TosoHookPositionStart

TosoHookPositionEnd

TosoHookMouseStart

TosoHookMouseEnd

TosoHookKeyStart

DRAWITEMSTRUCT (Basic Types)

Contents **Syntax**

```
typedef struct {
  UINT          CtlType;
  UINT          CtlID;
  UINT          itemID;
  UINT          itemAction;
  UINT          itemState;
  HWND          hwndItem;
  HDC           hDC;
  RECT          rcItem;
  DWORD         itemData;
} DRAWITEMSTRUCT;
```

This structure contains data that identifies a custom control used inside a window or dialog box. This structure is defined in `WINDOWS.H`.

Contents **Element Description**

CtlType

[*UINT*] Specifies the control type. This member can be one of the following values:

ODT_BUTTON
: Owner-drawn button.

ODT_COMBOBOX
: Owner-drawn combo box.

ODT_LISTBOX
: Owner-drawn list box.

ODT_MENU
: Owner-drawn menu item.

ODT_STATIC
: Owner-drawn static control.

This value is ignored by the serving application.

CtlID

[*UINT*] Specifies the identifier of the combo box, list box, button, or static control. This member is not used for a menu item.

This value is ignored by the serving application.

itemID

[*UINT*] Specifies the menu item identifier for a menu item or the index of the item in a list box or combo box. For an empty list box or combo box, this member can be -1. This allows the application to draw only the focus rectangle at the coordinates specified by the *rcItem* member even though there are no items in the control. This indicates to the user whether the list box or combo box has the focus. How the bits are set in the *itemAction* member determines whether the rectangle is to be drawn as though the list box or combo box has the focus.

This value is ignored by the serving application.

itemAction

[*UINT*] Specifies the drawing action required. This member can be one or more of the following

values:

ODA_DRAWENTIRE
The entire control needs to be drawn.

ODA_FOCUS
The control has lost or gained the keyboard focus. The itemState member should be checked to determine whether the control has the focus.

ODA_SELECT
The selection status has changed. The itemState member should be checked to determine the new selection state.

This value is ignored by the serving application.

itemState
[*UINT*] Specifies the visual state of the item after the current drawing action takes place. This member can be a combination of the following values:

ODS_CHECKED
The item is to be checked.

ODS_COMBOBOXEDIT
The drawing takes place in the selection field (edit control) of an ownerdrawn combo box.

ODS_DEFAULT
The item is the default item.

ODS_DISABLED
The item is to be drawn as disabled.

ODS_FOCUS
The item has the keyboard focus.

ODS_GRAYED
The item is to be grayed.

ODS_SELECTED
The menu item's status is selected.

Only the values ODS_CHECKED, ODS_DEFAULT and ODS_DISABLED are recognized by the serving application.

hwndItem
[*HWND*] Identifies the control for combo boxes, list boxes, buttons, and static controls. For menus, this member identifies the menu containing the item.

This value is ignored by the serving application.

hDC
[*HDC*] Identifies a device context (DC); this DC must be used when performing drawing operations on the control.

rcItem
[*RECT*] Specifies a rectangle that defines the boundaries of the control to be drawn. This rectangle is in the DC specified by the *hDC* member. Windows automatically clips anything the owner window draws in the DC for combo boxes, list boxes, and buttons, but does not clip menu items. When drawing menu items, the owner window must not draw outside the boundaries of the rectangle defined by the *rcItem* member.

itemData
[*DWORD*] Specifies the application-defined 32-bit value associated with the menu item. For a control, this parameter specifies the value last assigned to the list box or combo box by the

LB_SETITEMDATA or CB_SETITEMDATA message. If the list box or combo box has the LBS_HASSTRINGS or CBS_HASSTRINGS style, this value is initially zero. Otherwise, this value is initially the value that was passed to the list box or combo box in the *lParam* parameter of one of the following messages:

CB_ADDSTRING
CB_INSERTSTRING
LB_ADDSTRING
LB_INSERTSTRING

If *ctlType* is ODT_BUTTON or ODT_STATIC, *itemData* is zero.

This value is ignored by the serving application.

Comment

None.

OBJPTR (Basic Types)

Syntax

```
typedef void*           OBJPTR;
```

Pointer to any kind of unit.

Comment

Defined as `void*` to assure that this pointer will always be type-casted before usage.

OFFSET (Basic Types)

Syntax

```
typedef unsigned int    OFFSET;
```

Offset pointer for navigation inside a unit.

Comment

None.

STRx (Basic Types)

Syntax

```
typedef char            STR32   [32],
                        STR64   [64],
                        STR256  [256],
                        STR8192 [8192];
```

Comment

Standard text lengths that are frequently used inside other structure definitions. Please note that the *Toso Interface* does currently only support the 8-bit ANSI character set, no Unicode!

MATRIX (Basic Types)

Contents **Syntax**

```
typedef struct {
  double        m11, m12,
                m21, m22,
                m31, m32;
} MATRIX;
```

This structure contains a simple 3 x 2 matrix for modification purposes.

Contents **Element Description**

m11, *m12*
: [*double*] First line of a 3 × 2 matrix.

m21, *m22*
: [*double*] Second line of a 3 × 2 matrix.

m31, *m32*
: [*double*] Third line of a 3 × 2 matrix.

Contents **Comment**

The data type *MATRIX* is used to store a 3 × 3 matrix. Such matrices are used to store a combination of one or more of the following operations: translation (moving), scaling, rotation, distortion and reflection. To store all of these operations, a 3 × 3 matrix is required. 3 × 3 matrices use the following representation:

```
[ m11 m12 m13 ]
[ m21 m22 m23 ]
[ m31 m32 m33 ]
```

As the *Toso Interface* does only handle two-dimensional data and therefore only two-dimensional operations, all resulting matrices have the following, simplified form:

```
[ m11 m12 0.0 ]
[ m21 m22 0.0 ]
[ m31 m32 1.0 ]
```

Due to this simplification, only the first two columns of the matrix are stored. The third column is set to $[0.0\ 0.0\ 1.0]_T$ implicitly.

DPOINT (Basic Types)

Contents **Syntax**

```
typedef struct {
  double        x, y;
} DPOINT;
```

This structure contains the description of a single point.

Contents **Element Description**

x, *y*

[*double*] X- and Y-Coordinate of the point. The coordinates are in [mm] relative to the page center.

DRECT (Basic Types)

Contents **Syntax**

```
typedef struct {
  double        x1, y1, x2, y2;
} DRECT;
```

This structure contains the description of a rectangle.

Contents **Element Description**

x1, *y1*
: [*double*] X- and Y-Coordinate of the first corner-point of the frame. The coordinates are in [mm] relative to the page center.

x2, *y2*
: [*double*] X- and Y-Coordinate of the second corner-point of the frame. The coordinates are in [mm] relative to the page center.

Contents **Comment**

In most cases, the values in this data type have to be sorted, i.e. *x1* <= *x2* and *y1* <= *y2*. This speeds up further calculations with these frames.

FONTDEF (Basic Types)

Contents **Syntax**

```
typedef struct {
  int           Type,
                Style,
                Weight;
  STR64         Name;
} FONTDEF;
```

This structure contains the description of a font including type, style and weight.

Contents **Element Description**

Type

[*int*] Type of the font. This value determines, if the specified font is a internal font, a TrueType font or a device font (esp. PostScript). Possible values are:

FONTTYPE_TOSO
: Internal font (e.g. "DINDRAFT").

FONTTYPE_TRUETYPE
: TrueType font (e.g. "Times New Roman").

FONTTYPE_DEVICE
: PostScript or device font (e.g. "Palatino").

Style

[*int*] Style of the font. This value is a bitwise OR combination of several of the following styles:

FONTSTYLE_REGULAR
: No special style.

FONTSTYLE_ITALIC
: Italic. If this bit is set the font will be displayed italic (if possible).

FONTSTYLE_UNDERLINE
: Underline. If this bit is set the font will be displayed underlined (if possible).

FONTSTYLE_OVERLINE
: Underline. If this bit is set the font will be displayed overlined (if possible). Currently, this bit is only used within dimensions.

FONTSTYLE_STRIKEOUT
: Strikeout. If this bit is set the font will be displayed striked out (if possible). This bit is ignored at the moment!

FONTSTYLE_SYMBOL
: Symbol font. This bit has to be set if the font is a symbol font (e.g. "Symbol" or "Wingdings").

For internal fonts, this value should be set to FONTSTYLE_REGULAR, as it has no effect in this case.

Weight

[*int*] Weight of the font. The weight of the font is defined analogous to the weight definition of TrueType fonts. Possible values are:

0 Undefined

100	Thin
200	Extra Light
300	Light
400	Regular
500	Medium
600	Semibold
700	Bold
800	Extrabold
900	Black

For internal fonts, this value should be set to 400 (Regular), as it has no effect in this case.

Name

[*STR64*] Name of the font, up to 63 characters. Names of TrueType and PostScript fonts may be up to 31 characters long.

PAGEDEF (Drawing Types)

Contents **Syntax**

```
typedef struct {
  int           PageType,
                PageOrient;
  double        PageXSize,
                PageYSize;
} PAGEDEF;
```

This structure contains a page description.

Contents **Element Description**

PageType

[*int*] This value determines the page's size. Possible values are:

0x0000	User-defined format, i.e. the values of *PageXSize* and *PageYSize* state the page size. In this case, *PageOrient* should also be 0x0000.	
0x0001	DIN A4	210 × 297 mm
0x0002	DIN A3	297 × 420 mm
0x0003	DIN A3.2	297 × 594 mm
0x0004	DIN A3.1	297 × 841 mm
0x0005	DIN A3.0	297 × 1189 mm
0x0006	DIN A2	420 × 594 mm
0x0007	DIN A2.1	420 × 841 mm
0x0008	DIN A2.0	420 × 1189 mm
0x0009	DIN A1	594 × 841 mm
0x000a	DIN A1.0	594 × 1189 mm
0x000b	DIN A0	841 × 1189 mm
0x000c	DIN 2A0	1189 × 1682 mm
0x000d	ISO A4×3	297 × 630 mm
0x000e	ISO A4×4	297 × 841 mm
0x000f	ISO A4×5	297 × 1051 mm
0x0010	ISO A4×6	297 × 1261 mm
0x0011	ISO A3×3	420 × 891 mm
0x0012	ISO A3×4	420 × 1189 mm
0x0013	ISO A2×3	594 × 1261 mm
0x0014	DIN B5	176 × 250 mm
0x0015	DIN B4	250 × 353 mm
0x0016	DIN B3	353 × 500 mm
0x0017	DIN B2	500 × 707 mm
0x0018	DIN B1	707 × 1000 mm
0x0019	DIN B0	1000 × 1414 mm
0x001a	DIN C5	162 × 229 mm
0x001b	DIN C4	229 × 324 mm
0x001c	DIN C3	324 × 458 mm
0x001d	DIN C2	458 × 648 mm
0x001e	DIN C1	648 × 917 mm
0x001f	DIN C0	917 × 1297 mm
0x0020	US Half	5.5 × 8.5 inch

0x0021	ANSI A / US Letter	8.5 × 11.0 inch
0x0022	ANSI B / US Tabloid	11.0 × 17.0 inch
0x0023	ANSI C	17.0 × 22.0 inch
0x0024	ANSI D	22.0 × 34.0 inch
0x0025	ANSI E	34.0 × 44.0 inch
0x0026	US Legal	8.5 × 14.0 inch

PageOrient

[*int*] This value determines whether the page shall have landscape or portrait orientation. Possible values are:

0x0000	User-defined format, i.e. the values of *PageXSize* and *PageYSize* state the page size. In this case, *PageType* should also be 0x0000.
0x0001	Portrait
0x0002	Landscape

PageXSize

[*double*] This value states the page's horizontal size in millimeters. It must be valid even if *PageType* states an explicit page format. The valid range is 1.0 to 4000.0 inclusive.

PageYSize

[*double*] This value states the page's vertical size in millimeters. It must be valid even if *PageType* states an explicit page format. The valid range is 1.0 to 4000.0 inclusive.

DEFAULTDEF (Drawing Types)

Contents **Syntax**

```
typedef struct {
  int           Pens    [TVG_DEFAULT_MAX],
                Layers  [TVG_DEFAULT_MAX];
} DEFAULTDEF;
```

This structure contains the default settings for pen and layer assignment.

Contents **Element Description**

Pens

[*int[]*] Indices of pens that will automatically be assigned to object created in some standard situations. Use the following constants as field indices to access elements of this array:

DEFPEN_DIMLINE
: Default pen for dimension lines.

DEFPEN_DIMTEXT
: Default pen for dimension texts.

DEFPEN_TEXT
: Default pen for standard, frame and reference texts.

DEFPEN_REFLINE
: Default pen for additional reference text elements (frame and arrow).

DEFPEN_GEOMETRY
: Default pen for geometry objects.

If any of these indices is -1, there is no default pen defined for this situation, i.e. the object will be assigned to the currently active pen.

Layers

[*int[]*] Indices of layers that will automatically be assigned to object created in some standard situations. Use the following constants as field indices to access elements of this array:

DEFLAYER_MARK
: Default layer for markings.

DEFLAYER_OUTLINE
: Default layer for generated outlines (surfaces).

DEFLAYER_DIM
: Default layer for dimension lines and texts.

DEFLAYER_TEXT
: Default layer for texts.

DEFLAYER_HATCH
: Default layer for hatchings.

DEFLAYER_BLOCK
: Default layer for instances.

DEFLAYER_GEOMETRY

Default layer for geometry objects.

If any of these indices is -1, there is no default layer defined for this situation, i.e. the object will be assigned to the currently active layer.

SYSTEMDEF (Drawing Types)

Contents **Syntax**

```
typedef struct {
  STR32          SystemName;

  double         SystemRotate,
                 SystemScale;
  int            SystemOption;

  int            SystemOrgMode;
  double         SystemXOrg,
                 SystemYOrg;

  int            SystemLenUnit,
                 SystemLineUnit,
                 SystemAngleUnit,
                 SystemFraction,
                 SystemAccuracy;

  int            SystemGridMode;
  double         SystemXGrid,
                 SystemYGrid;

  int            SystemSnapMode;
  double         SystemXSnap,
                 SystemYSnap;

  MATRIX         SystemMMToUnit,
                 SystemUnitToMM;
} SYSTEMDEF;
```

This structure contains a coordinate system definition.

Contents **Element Description**

SystemName

[*STR32*] Name of the coordinate system, up to 31 characters.

SystemRotate

[*double*] Rotation of the display in [rad]. The rotation is valid for screen display only, not during output.

SystemScale

[*double*] Drawing scale (100.0 represents a scale of 100:1, 0.02 represents a scale of 1:50 etc.). Valid range is 1e-10 to 1e10 inclusive.

SystemOption

[*int*] Distortion mode of the display. The distortion is valid for screen display only. This option allows to distort the screen display of a drawing in a way that isometric or dimetric views inside this drawing will be displayed rectangular, plain and without distortion. By this means, isometric and dimetric drawings can be produced without having to calculate the exact angles and lengths of lines and ellipses. For an illustration of the effect of this option, have a close look at the application. Possible values are:

0x0000	No distortion
0x0001	Left view of an isometric drawing

0x0002 Right view of an isometric drawing
0x0003 Top view of an isometric drawing
0x0004 Left view of an dimetric drawing 1 (-7°)
0x0005 Right view of an dimetric drawing 1 (-7°)
0x0006 Top view of an dimetric drawing 1 (-7°)
0x0007 Left view of an dimetric drawing 2 (+7°)
0x0008 Right view of an dimetric drawing 2 (+7°)
0x0009 Top view of an dimetric drawing 2 (+7°)

This distortion is independent from the grid settings (see *SystemGridMode* and *SystemSnapMode*). Anyway, a distorted view should never be combined with a distorted grid - this will lead to deep confusion of the user.

SystemOrgMode

[*int*] Placement of the coordinate system's origin. Possible values are:

0x0000 The origin is located at the position stated in *SystemXOrg* and *SystemYOrg*.
0x0001 The origin is located at the upper left page corner.
0x0002 The origin is located at the center of the upper page edge.
0x0003 The origin is located at the upper right page corner.
0x0004 The origin is located at the center of the left page edge.
0x0005 The origin is located at the page's center.
0x0006 The origin is located at the center of the right page edge.
0x0007 The origin is located at the lower left page corner.
0x0008 The origin is located at the center of the lower page edge.
0x0009 The origin is located at the lower right page corner.

Internal drawing coordinates are *always* relative to the page's center, independent of the origin's placement!

SystemXOrg
SystemYOrg

[*double*] Coordinates of the origin in millimeters relative to the page's center. They must be valid even if *SystemOrgMode* states an explicit origin placement. The valid range is -1e100 to 1e100.

SystemLenUnit

[*int*] Length unit to be used. This unit will be used for in- and output of coordinates and all values that are depending on the drawing's scale (like dimensions, perimeter, area etc.). Possible values are:

0x0000 [µm]
0x0001 [mm]
0x0002 [cm]
0x0003 [dm]
0x0004 [m]
0x0005 [km]
0x0006 [mil]
0x0007 [inch]
0x0008 [foot]
0x0009 [yard]
0x000a [mile]
0x000b [dp]
0x000c [pt]
0x000d [bp]
0x000e [cic]

SystemLineUnit

[*int*] Line unit to be used. This unit will be used for in- and output of values that are not depending on the drawing's scale (line width, font size etc.). Possible values are:

0x0000 [µm]

0x0001 [mm]
0x0002 [cm]
0x0006 [mil]
0x0007 [inch]
0x000b [dp]
0x000c [pt]
0x000d [bp]
0x000e [cic]

SystemAngleUnit

[*int*] Angle unit to be used. This unit will be used for in- and output of angle values. Possible values are:

0x0000 [deg]
0x0001 [gra]
0x0002 [rad]
0x0003 [rel]

SystemFraction

[*int*] This value determines how non-integer values will be displayed. Possible values are:

0x0000 The value will be displayed using floating point representaion, i.e. with a decimal separator and following fractional digits, e.g. `10.75`. How trailing zeros are handled depends on user settings in the application.

0x0001 The value will be displayed using a mixed fraction representation, i.e. the integer value will be followed by a fraction with a power of 2 as denominator, e.g. `10 3/4`. The fraction will be reduced automatically.

0x0002 The value will be split into [foot] and [inch] elements, where the value itself is assumed to be in [inch]. The [foot] element will always be integer, the [inch] element will use floating point representation (see 0x0000). A value of 170.75 inch will be displayed as `14'2.75"`.

0x0003 The value will be split into [foot] and [inch] elements, where the value itself is assumed to be in [inch]. The [foot] element will always be integer, the [inch] element will use mixed fraction representation (see 0x0001). A value of 170.75 inch will be displayed as `14'2 3/4"`.

0x0004 The value will be split into [yard], [foot] and [inch] elements, where the value itself is assumed to be in [inch]. The [yard] and [foot] elements will always be integer, the [inch] element will use floating point representation (see 0x0000). A value of 170.75 inch will be displayed as `4yd2'2.75"`.

0x0005 The value will be split into [yard], [foot] and [inch] elements, where the value itself is assumed to be in [inch]. The [yard] and [foot] elements will always be integer, the [inch] element will use mixed fraction representation (see 0x0001). A value of 170.75 inch will be displayed as `4yd2'2 3/4"`.

0x0006 The value will be split into [degree], [minute] and [second] elements, where the value itself is assumed to be in [degree]. The [degree] and [minute] elements will always be integer, the [second] element will use floating point representation (see 0x0000). A value of 37.331 degree will be displayed as `37°19'51.6"`.

0x0007 The value will be split into [degree], [minute] and [second] elements, where the value itself is assumed to be in [degree]. The [degree] and [minute] elements will always be integer, the [second] element will use mixed fraction representation (see 0x0001). A value of 37.331 degree will be displayed as `37°19'51 5/8"`.

0x0008 The value will be displayed using floating point representation, i.e. with a decimal separator and following fractional digits, e.g. `10.75`. Values below one will be multiplied by 100 before display. This setting in mainly used for

architectural drawings based on meters. How trailing zeros are handled depends on user settings in the application.

Note: Some fraction modes make only sense for length values or coordinates, others only for angle values. Anyway, all modes are allowed in both cases.

SystemAccuracy

[*int*] This value determines the accuracy of non-integer value output to the screen. If values are displayed using floating point representation (*SystemFraction* = 0x0000, 0x0002, 0x0004, 0x0006 or 0x0008), this value states the maximum number of fractional digits. If values are displayed using mixed fraction representation (*SystemFraction* = 0x0001, 0x0003, 0x0005 or 0x0007), this value states the maximum power of 2 the denominator will have. In both cases, the valid range is 0 to 9 inclusive.

SystemGridMode

[*int*] Current mode of the display grid. The display grid will be displayed on the screen, it does not influence the cursor's movement. Possible values are:

GRID_OFF
: No grid.

GRID_CARTESIAN
: Cartesian grid.

GRID_ISOMETRIC
: Isometric grid.

GRID_DIMETRIC1
: Dimetric grid 1 (-7°).

GRID_DIMETRIC2
: Dimetric grid 2 (+7°).

SystemXGrid
SystemYGrid

[*double*] Distance of two display grid points in X- and Y-direction in the current unit. Valid range is 0.0 to 1e10 inclusive. If *SystemXGrid* ans/or *SystemYGrid* are 0.0, the grid is invalid and will not be displayed.

SystemSnapMode

[*int*] Current mode of the position grid. The position grid influences the cursor's movement, it will not be displayed on the screen. Possible values are:

GRID_OFF
: No grid.

GRID_CARTESIAN
: Cartesian grid.

GRID_ISOMETRIC
: Isometric grid.

GRID_DIMETRIC1
: Dimetric grid 1 (-7°).

GRID_DIMETRIC2
: Dimetric grid 2 (+7°).

SystemXSnap
SystemYSnap

[*double*] Distance of two position grid points in X- and Y-direction in the current unit. Valid range is 0.0 to 1e10 inclusive. If either *SystemXSnap* or *SystemYSnap* is 0.0, the cursor's movement will be

restricted only in one direction. If both values are 0.0, the grid is invalid and will have no effect.

SystemMMToUnit

[*MATRIX*] This matrix can be used to convert internal point coordinates (millimeters relative to the page's center) into coordinates based on the system's origin, scale and distortion. If a *SYSTEMDEF* structure is used to create a new or modify an existing coordinate system, it is not necessary to initialize this matrix, it will automatically be updated by the system.

SystemUnitToMM

[*MATRIX*] This matrix can be used to convert coordinates based on the system's origin, scale and distortion into internal point coordinates (millimeters relative to the page's center). If a *SYSTEMDEF* structure is used to create a new or modify an existing coordinate system, it is not necessary to initialize this matrix, it will automatically be updated by the system.

LINEDEF (Drawing Types)

Contents **Syntax**

```
typedef struct {
  STR32          LineName;

  int            LineNum,
                 LineMode;
  int            LineData[TVG_LINEPART_MAX];
} LINEDEF;
```

This structure contains a line pattern definition.

Contents **Element Description**

LineName

[*STR32*] Name of the line pattern, up to 31 characters.

LineNum

[*int*] Number of pairs of "line" and "hole". Valid range: 0 <= Value <= TVG_LINEPART_MAX / 2.

LineMode

[*int*] Definition mode of the line pattern. Possible values are:

0x0000	The partial line's lengths will be stated in 1/100 of the line's width. If the line's width is less than 0.1 mm, the calculation will be based on a line width of 0.1 mm.
0x0001	The partial line's lengths will be stated in 1/100 mm.

LineData

[*int[]*] Partial line length pairs. Each line length may be between 100 and 10000 inclusive. The first value of each pair defines a "line", the second value defines a "hole". This list should contain exactly two times the number of line lengths as the value stored in *LineNum* states.

PROPERTY (Drawing Types)

Contents **Syntax**

```
typedef struct {
  int           FillMode;
  COLORREF      FillColor,
                LineColor;
  int           LineType;
  double        LineWidth;
} PROPERTY;
```

Contents **Element Description**

FillMode

[*int*] The value *FillMode* determines, which parts of an object are to be drawn. Following values are defined:

FILLMODE_FRAMED
: The outline of the object is drawn.

FILLMODE_FILLED
: The object is filled.

FILLMODE_FILLED_FRAMED
: The object is filled and its outline is drawn.

FILLMODE_ERASER
: The object is erased (i.e. filled in background color).

FILLMODE_ERASER_FRAMED
: The object is erased (i.e. filled in background color) and its outline is drawn.

Some objects do not surround a closed surface (e.g. a line or a circular arc). If such an object is drawn using a *FillMode* of FILLMODE_FILLED or FILLMODE_ERASER, it will not be visible.

FillColor

[*COLORREF*] Color of the object's surface in RGB notation.

LineColor

[*COLORREF*] Color of the object's outline in RGB notation.

LineType

[*int*] Index of the line pattern used to draw the outline. Valid range: 0 <= Value <= TVG_LINE_MAX.

LineWidth

[*double*] Width of the object's outline in [mm] between 0.0 and 100.0 (including). A width of 0.0 always results in a line of the minimum width possible on the respective device (one pixel).

PENDEF (Drawing Types)

Contents **Syntax**

```
typedef struct {
  STR64          PenName;

  PROPERTY       PenIntern,
                 PenExtern;
  int            PenLayer;
} PENDEF;
```

This structure contains a pen definition.

Contents **Element Description**

PenName
: [*STR64*] Name of the pen, up to 63 characters.

PenIntern
: [*PROPERTY*] Property set for screen display. This property set will only be used if the pen transmission is not identical for display and output.

PenExtern
: [*PROPERTY*] Property set for output (printer, plotter, clipboard, metafile, export etc.).

PenLayer
: [*int*] Index of the layer that is to be activated if this pen is activated. If *PenLayer* is -1, the current layer will not be changed. Valid range: -1 <= Value <= TVG_PEN_MAX.

XPROPERTY (Drawing Types)

Contents **Syntax**

```
typedef struct {
  int           Flag,
                Pen;

  int           FillMode;
  COLORREF      FillColor,
                LineColor;
  int           LineType;
  double        LineWidth;

  int           Layer;
} XPROPERTY;
```

This structure contains an extended property set.

Contents **Element Description**

Flag

[*int*] In units of type "Instance" or "Block", the transmission flags indicate whether to transmit a specific property or not. This is stored in the value *Flag*, which is a bitwise OR combination of one or more of the following values:

USE_NULL
: No property is transmitted / fixed.

USE_PEN
: Transmit / fix the pen index *Pen*.

USE_FILLMODE
: Transmit / fix the filling mode *FillMode*.

USE_FILLCOLOR
: Transmit / fix the filling color *FillColor*.

USE_LINECOLOR
: Transmit / fix the line color *LineColor*.

USE_LINEWIDTH
: Transmit / fix the line width *LineWidth*.

USE_LINETYPE
: Transmit / fix the line type *LineType*.

USE_LAYER
: Transmit / fix the layer index *Layer*.

The value USE_PEN has the lowest priority, the value USE_LAYER has the highest priority. Assuming that both a pen index (and consequently the pen's line width) and an explicit line width are transmitted, the explicit line width (transmitted by setting the USE_LINEWIDTH flag) has a higher priority than the transmitted pen's line width (transmitted by setting the USE_PEN flag).
In units of type "Object", the transmission flags indicate whether to accept the transmission or not. This is stored in the value *Flag*, which is a bitwise OR combination of one or more of the same values

as stated above. If one of these flags is set, the transmission of the corresponding layer/pen property will not be accepted, i.e. this object property is fixed.

Pen

[*int*] Index of the pen. Valid range: 0 <= Value <= TVG_PEN_MAX.

FillMode

[*int*] The value *FillMode* determines, which parts of an object are to be drawn. Following values are defined:

FILLMODE_FRAMED
: The outline of the object is drawn.

FILLMODE_FILLED
: The object is filled.

FILLMODE_FILLED_FRAMED
: The object is filled and its outline is drawn.

FILLMODE_ERASER
: The object is erased (i.e. filled in background color).

FILLMODE_ERASER_FRAMED
: The object is erased (i.e. filled in background color) and its outline is drawn.

Some objects do not surround a closed surface (e.g. a line or a circular arc). If such an object is drawn using a *FillMode* of FILLMODE_FILLED or FILLMODE_ERASER, it will not be visible.

FillColor

[*COLORREF*] Color of the object's surface in RGB notation.

LineColor

[*COLORREF*] Color of the object's outline in RGB notation.

LineType

[*int*] Index of the line pattern used to draw the outline. Valid range: 0 <= Value <= TVG_LINE_MAX.

LineWidth

[*double*] Width of the object's outline in [mm] between 0.0 and 100.0 (including). A width of 0.0 always results in a line of the minimum width possible on the respective device (one pixel).

Layer

[*int*] Index of the layer. Valid range: 0 <= Value <= TVG_LAYER_MAX.

LAYERDEF (Drawing Types)

Contents **Syntax**

```
typedef struct {
  STR64          LayerName;

  XPROPERTY      LayerIntern,
                 LayerExtern;
  int            LayerMode;
} LAYERDEF;
```

This structure contains a layer definition.

Contents **Element Description**

LayerName

[*STR64*] Name of the layer, up to 63 characters.

LayerIntern

[*XPROPERTY*] This set of properties defines single or multiple properties to be transmitted to all entities assigned to this layer during display. This transmission has a higher priority then the pen transmission, i.e. it will override properties transmitted by the pen. However if an object property is fixed any transmission will be ignored. A transmission of the layer index will be ignored, of course. This extended property set will only be used if the layer transmission is not identical for display and output.

LayerExtern

[*XPROPERTY*] This set of properties defines single or multiple properties to be transmitted to all entities assigned to this layer during output. This transmission has a higher priority then the pen transmission, i.e. it will override properties transmitted by the pen. However if an object property is fixed any transmission will be ignored. A transmission of the layer index will be ignored, of course.

LayerMode

[*long*] Determines whether the layer is frozen, displayed and/or idle. The display can separately be set for screen display and output. *LayerMode* is a bitwise OR combination of the following values:

LAYERMODE_DISPLAY

The layer will be displayed on the screen.

LAYERMODE_OUTPUT

The layer will be output to printer, clipboard etc.

LAYERMODE_FREEZE

The layer is frozen, i.e. objects assigned to that layer cannot be modified.

LAYERMODE_IDLE

The layer is idle, i.e. objects assigned to that layer will be ignored during snapping.

LAYERMODE_GRAY

The layer is grayed (shaded), i.e. objects assigned will be displayed in a "gray" color that is defined by the user. This is achieved by transmitting this color as line and fill color to all objects and instances in that layer. In this special case objects are even grayed (shaded) if their line or fill color is fixed.

HATCHDEF (Drawing Types)

Contents **Syntax**

```
typedef struct {
  STR64          HatchName;

  BOOL           HatchLine1Active;
  int            HatchLine1;
  double         HatchLine1Rotate;

  BOOL           HatchLine2Active;
  int            HatchLine2;
  double         HatchLine2Rotate;

  BOOL           HatchBlockActive;
  STR64          HatchLibraryName,
                 HatchBlockName;
  double         HatchBlockRotate,
                 HatchBlockScale,
                 HatchXStep1,
                 HatchXStep2,
                 HatchYStep1,
                 HatchYStep2,
                 HatchLineStep1,
                 HatchLineStep2;

  double         HatchRotate,
                 HatchOffset1,
                 HatchOffset2;
} HATCHDEF;
```

This structure contains a hatching type definition.

Contents **Element Description**

HatchName
[*STR64*] Name of the hatching type, up to 63 characters.

HatchLine1Active
[*BOOL*] Determines whether the first line sequence it active or not.

HatchLine1
[*int*] Index of the first line sequence to be used. Valid range: 0 <= Value <= TVG_MULTILINE_MAX.

HatchLine1Rotate
[*double*] Rotation angle of the first line sequence.

HatchLine2Active
[*BOOL*] Determines whether the second line sequence it active or not.

HatchLine2
[*int*] Index of the first second sequence to be used. Valid range: 0 <= Value <= TVG_MULTILINE_MAX.

HatchLine2Rotate
[*double*] Rotation angle of the second line sequence.

HatchBlockActive
[*BOOL*] Determines whether a hatching block is to be used.

HatchLibraryName

[*STR64*] Name of the library containing the desired block, maximum 63 characters. If the desired hatching block is located inside the drawing, set this name to "*".

HatchBlockName

[*STR64*] Name of the block, maximum 63 characters.

HatchBlockRotate

[*double*] This value determines the rotation of the hatching block (based on its insertion point).

HatchBlockScale

[*double*] This value determines the scaling of the hatching block.

HatchBlockXStep1

HatchBlockXStep2

[*double*] These two values determine the horizontal advance of the block-based hatching.

HatchBlockYStep1

HatchBlockYStep2

[*double*] These two values determine the vertical advance of the block-based hatching.

HatchBlockLineStep1

HatchBlockLineStep2

[*double*] These two values determine the horizontal line offset of the block-based hatching.

HatchRotate

[*double*] This value determines the global rotation of the complete hatching.

HatchOffset1

HatchOffset2

[*double*] These two values determine the offset of the hatching.

MULTILINE (Drawing Types)

Contents **Syntax**

```
typedef struct {
  double        Distance;
  XPROPERTY     XProperty;
  BOOL          Use;
} MULTILINE;
```

This structure contains a the definition of a single multiline sub-line.

Contents **Element Description**

Distance

[*double*] Determines the distance between this line and the next active line in internal mm. The valid range is 1e-10 to 1e10 inclusive.

XProperty

[*XPROPERTY*] This set of properties defines single or multiple properties to be transmitted to the corresponding hatching line.

Use

[*BOOL*] Determines whether the corresponding line is to be used or not.

MULTILINEDEF (Drawing Types)

Contents **Syntax**

```
typedef struct {
  STR32          MultiLineName;
  MULTILINE      MultiLineData[TVG_MULTILINEPART_MAX];
} MULTILINEDEF;
```

This structure contains a multiline definition.

Contents **Element Description**

MultiLineName

[*STR32*] Name of the line sequence, up to 31 characters.

MultiLineData

[*MULTILINE[]*] List of lines that make up the line sequence. Each line may be active or not and has its own property set.

COLORDEF (Drawing Types)

Contents **Syntax**

```
typedef struct {
  STR32          ColorName;
  COLORREF       ColorValue;
} COLORDEF;
```

This structure contains a custom color definition.

Contents **Element Description**

ColorName

[*STR32*] Name of the custom color, up to 31 characters.

ColorValue

[*COLORREF*] RGB definition of the color.

WINDOWDEF (Drawing Types)

Contents **Syntax**

```
typedef struct {
  int           WindowSystem;
  BOOL          WindowGrid,
                WindowSnap;
  double        WindowXCenter,
                WindowYCenter,
                WindowZoom;
} WINDOWDEF;
```

This structure contains a custom color definition.

Contents **Element Description**

WindowSystem

[*long*] Index of the coordinate systems that was active in the drawing window when the drawing was saved (referring to *SystemIndex*). Valid range: 0 <= Value <= TVG_SYSTEM_MAX.

WindowGrid

[*BOOL*] Determines whether display grid of the drawing window was active when the drawing was saved.

WindowSnap

[*BOOL*] Determines whether position grid of the drawing window was active when the drawing was saved.

WindowXCenter

WindowYCenter

[*double*] Coordinates of the center-point of the drawing window in millimeters relative to the page's center. The valid range is -1e100 to 1e100.

WindowZoom

[*double*] View size of the drawing window. The valid range is -1e12 to 1e12. This value determines the zoom factor relative to original size.

FILE_HEADER (Drawing Types)

Contents **Syntax**

```
typedef struct {
  STR64          Title,
                 Theme,
                 Author1,
                 Date1,
                 Author2,
                 Date2;
  STR256         Comment;
} FILE_HEADER;
```

This structure contains a file description header for TVG 4.0 files.

Contents **Element Description**

Title

[*STR64*] Title of the file. This text usually contains a detailed description of the file.

Theme

[*STR64*] Theme of the file. This text usually contains a description of the project the file belongs to.

Author1

[*STR64*] Name of the user that created the file. This text is initialized when creating a file.

Date1

[*STR64*] Date and time when the file was created. In the English release, this text might e.g. be "Tuesday, October 15 1991, 5:15 pm".

Author2

[*STR64*] Name of the user that modified this file for the last time. This text is initialized each time the file it saved.

Date2

[*STR64*] Date and time when this file was saved for the last time. This time is initialized each time the file is saved.

Comment

[*STR256*] This text contains any further comment on the file.

TOKEN_DATA (Module Interface Types)

Contents **Syntax**

```
typedef struct {
  char          Newline[4],
                Comma,
                Semi,
                Color,
                Escape,
                String,
                Keyword,
                Comment;
} TOKEN_DATA;
```

This structure contains a set of options for text-based file I/O operations.

Contents **Element Description**

Newline
: [*char[4]*] This text is used when a new line inside a text file is started. The default value of *Newline* is TAX_NEWLINE (sequence Ansi 13, Ansi 10, Ansi 0).

Comma
: [*char*] This character is used to separate single values (sometimes referred to as "field separator"). The default value of *Comma* is TAX_COMMA (Ansi 44).

Semi
: [*char*] This character is used to separate value sets (sometimes referred to as "record separator"). The default value of *Comma* is TAX_SEMI (Ansi 59).

Color
: [*char*] This character is used to separate the three single values of color descriptions. The default value of *Color* is TAX_COLOR (Ansi 47).

Escape
: [*char*] This character is used to encode control characters inside a quoted text. The default value of *Escape* is TAX_ESCAPE (Ansi 92).

String
: [*char*] This character is used to delimit texts. The default value of *String* is TAX_STRING (Ansi 34).

Keyword
: [*char*] This character is used to delimit keywords. The default value of *Keyword* is TAX_KEYWORD (Ansi 61).

Comment
: [*char*] This character is used to delimit comments. The default value of *Comment* is TAX_COMMENT (Ansi 124).

DIMLINE (Object Info Types)

Contents **Syntax**

```
typedef struct {
  int           ArrowStartForm,
                ArrowStartMode,
                ArrowEndForm,
                ArrowEndMode;
} DIMLINE;
```

This structure contains settings for a dimension line.

Contents **Element Description**

ArrowStartForm

[*int*] The value *ArrowStartForm* determines the form of the dimension arrow at the start-point of the dimension line. Possible values are:

0x0000	No dimension arrow.
0x0001	Filled triangular arrow. The triangle has an opening angle of 20° and a side length of 10.0 times the dimension line's width.
0x0002	Non-filled triangular arrow. The triangle has an opening angle of 20° and a side length of 10.0 times the dimension line's width.
0x0003	Open triangular arrow. The triangle has an opening angle of 60° and a side length of 10.0 times the dimension line's width.
0x0004	Diagonal stroke. The stroke has a relative angle of 45° to the dimension line and a length of 12.0 times the dimension line's width.
0x0005	Filled circle. The filled circle has a radius of 1.5 times the dimenion line's width.
0x0006	Non-filled circle. The filled circle has a radius of 2.5 times the dimenion line's width.

ArrowStartMode

[*int*] The values *ArrowStartMode* determines whether the dimension ends at the dimension's start-point or whether it is extended (which would result in rotated dimension arrows). Possible values are:

0x0000	The dimension line ends at the dimension's start-point, the dimension arrows will not be rotated.
0x0001	The dimension line is extended, the dimension arrows will be rotated.
0x0002	Automatic length detection. If the dimension is less than 30.0 times the dimension line's width, the dimension lines will be extended and the dimension arrows will be rotated.

The extension of the dimension depends on the dimension arrow type. For types 0x0001, 0x0002 and 0x0003, it is 30.0 times the dimension line's width, for types 0x0004, 0x0005 and 0x0006, it is 5.0 times the dimension line's width. Type 0x0000 has no extension.

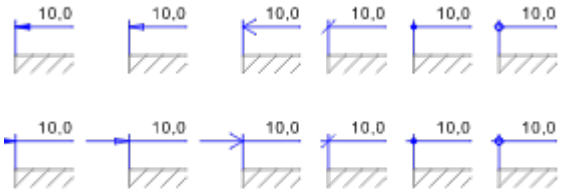
ArrowEndForm

[*int*] Equivalent to *ArrowStartForm*, but referring to the end-point.

ArrowEndMode

[*int*] Equivalent to *ArrowStartMode*, but referring to the end-point.

The following image shows all types of dimension arrows, once in normal presentation (upper row), once in rotated presentation with extended dimension line (lower row):


10,0
10,0
10,0
10,0
10,0
10,0
10,0
10,0
10,0
10,0
10,0
10,0

DIMSMALL (Object Info Types)

Contents **Syntax**

```
typedef struct {
  FONTDEF        TextFont;
  XPROPERTY      TextXProperty;
  double         TextSize1,
                 TextSize2;

  int            NumAccuracy;
  BOOL           NumRefresh,
                 NumRotate;
  int            System;
} DIMSMALL;
```

This structure contains settings for a dimension without dimension line.

Contents **Element Description**

TextFont

[*FONTDEF*] Description of the font to be used for the dimension texts (dimension and tolerances).

TextXProperty

[*XPROPERTY*] Properties for the dimension texts (dimension and tolerances). The properties of the unit itself are only valid for dimension line and dimension arrow.

TextSize1

[*double*] Font size of the dimension in mm.

TextSize2

[*double*] Font size of the tolerences in mm.

NumAccuracy

[*int*] The value *NumAccuracy* determines the accuracy of the dimension's display.
If numeric values are displayed as decimal numbers, this value determines the number of fractional digits. The value may be between 0 (no fractional digit) and 9 (nine fractional digits). Whether trailing zeros will be displayed or not depends of user-dependent settings in the application.
If numeric values are displayed as fractional numbers, this value determines the maximum power of two that the denominator will have. The value may be between 0 (no fraction) and 9 (maximum denominator 512). The resulting fraction will be reduced. If the numeric value is 2.1, the resulting fraction will be `2 3/32` for a *NumAccurary* of 6 and `2 51/512` for a *NumAccuracy* of 9.

NumRefresh

[*int*] The value *NumRefresh* determines whether the dimension shall be recalculated after each modification or not. Possible values are:

0x0000	The dimension will only recalculated on demand.
0x0001	The dimension will be recalculated after each modification.

NumRotate

[*int*] The value *NumRotate* determines, how the dimension number shall be rotated. Possible values are:

0x0000	The dimension number is parallel to the dimension line, with an angle of its base line bewteen -90° and +90°, i.e. the text can either be read from below or from the right.
0x0001	The dimension number is parallel to a line running throught the points (zx1,zy1) und (zx2,zy2).

System

[*int*] Index of the coordinate system the dimension shall be based on. If this coordinate system is set to a distorting display (isometric or dimetric), the dimension will be calculated accordingly.

DIMLARGE (Object Info Types)

Contents **Syntax**

```
typedef struct {
  FONTDEF       TextFont;
  XPROPERTY     TextXProperty;
  double        TextSize1,
                TextSize2;

  int           NumAccuracy;
  BOOL          NumRefresh,
                NumCentered,
                NumTight,
                NumRotate;

  int           ArrowStartForm,
                ArrowStartMode,
                ArrowEndForm,
                ArrowEndMode;

  BOOL          ExtStartDisplay,
                ExtEndDisplay,
                LineDisplay;
  int           LineOrientation,
                LineType,
                LineDistMode;
  double        LineDistance;
  int           System;
} DIMLARGE;
```

This structure contains settings for a dimension with dimension line.

Contents **Element Description**

TextFont
[*FONTDEF*] Description of the font to be used for the dimension texts (dimension and tolerances).

TextXProperty
[*XPROPERTY*] Properties for the dimension texts (dimension and tolerances). The properties of the unit itself are only valid for dimension line and dimension arrow.

TextSize1
[*double*] Font size of the dimension in mm.

TextSize2
[*double*] Font size of the tolerences in mm.

NumAccuracy
[*int*] The value *NumAccuracy* determines the accuracy of the dimension's display.
If numeric values are displayed as decimal numbers, this value determines the number of fractional digits. The value may be between 0 (no fractional digit) and 9 (nine fractional digits). Whether trailing zeros will be displayed or not depends of user-dependent settings in the application.
If numeric values are displayed as fractional numbers, this value determines the maximum power of two that the denominator will have. The value may be between 0 (no fraction) and 9 (maximum denominator 512). The resulting fraction will be reduced. If the numeric value is 2.1, the resulting fraction will be `2 3/32` for a *NumAccurary* of 6 and `2 51/512` for a *NumAccuracy* of 9.

NumRefresh

[*int*] The value *NumRefresh* determines whether the dimension shall be recalculated after each modification or not. Possible values are:

0x0000	The dimension will only recalculated on demand.
0x0001	The dimension will be recalculated after each modification.

NumCentered

[*int*] The value *NumCentered* determines whether the dimension number shall always be placed centered to the dimension line or not. Possible values are:

0x0000	The dimension number can be placed anywhere. In this case, (zx1,zy1) determines the center of the base line of the dimension number.
0x0001	The dimension number is always placed centered to the dimension line. The resulting position is the base point of a perpendicular dropped from (zx1,zy1) onto the mid-perpendicular of the dimension line. A rotation angle defined by the points (zx1,zy1) and (zx2,zy2) remains unchanged.

If both *NumTight* and *NumCentered* are non-zero, first the calculation of *NumCentered* is executed, the calculation of *NumTight*.

NumTight

[*int*] The value *NumTight* determines whether the dimension number shall always be placed tight to the dimension line or not. If so, the distance between the text's base line and the dimension line is one-quarter of the dimension's font size. Possible values are:

0x0000	The dimension number can be placed anywhere. In this case, (zx1,zy1) determines the center of the base line of the dimension number.
0x0001	The dimension numer is always placed tight to the dimension line. The resulting position is on the perpendicular dropped from (zx1,zy1) onto the dimension line, having a distance of one-quarter of the dimension's font size to the dimension line. A rotation angle defined by the points (zx1,zy1) and (zx2,zy2) remains unchanged.

If both *NumTight* and *NumCentered* are non-zero, first the calculation of *NumCentered* is executed, the calculation of *NumTight*.

NumRotate

[*int*] The value *NumRotate* determines, how the dimension number shall be rotated. Possible values are:

0x0000	The dimension number is parallel to the dimension line, with an angle of its base line bewteen -90° and +90°, i.e. the text can either be read from below or from the right.
0x0001	The dimension number is parallel to a line running throught the points (zx1,zy1) und (zx2,zy2).

ArrowStartForm

[*int*] The value *ArrowStartForm* determines the form of the dimension arrow at the start-point of the dimension line. Possible values are:

0x0000	No dimension arrow.
0x0001	Filled triangular arrow. The triangle has an opening angle of 20° and a side length of 10.0 times the dimension line's width.
0x0002	Non-filled triangular arrow. The triangle has an opening angle of 20° and a side length of 10.0 times the dimension line's width.
0x0003	Open triangular arrow. The triangle has an opening angle of 60° and a side length of 10.0 times the dimension line's width.
0x0004	Diagonal stroke. The stroke has a relative angle of 45° to the dimension line and a length of 12.0 times the dimension line's width.

0x0005 Filled circle. The filled circle has a radius of 1.5 times the dimenion line's width.
0x0006 Non-filled circle. The filled circle has a radius of 2.5 times the dimenion line's width.

ArrowStartMode

[*int*] The values *ArrowStartMode* determines whether the dimension ends at the dimension's start-point or whether it is extended (which would result in rotated dimension arrows). Possible values are:

0x0000 The dimension line ends at the dimension's start-point, the dimension arrows will not be rotated.
0x0001 The dimension line is extended, the dimension arrows will be rotated.
0x0002 Automatic length detection. If the dimension is less than 30.0 times the dimension line's width, the dimension lines will be extended and the dimension arrows will be rotated.

The extension of the dimension depends on the dimension arrow type. For types 0x0001, 0x0002 and 0x0003, it is 30.0 times the dimension line's width, for types 0x0004, 0x0005 and 0x0006, it is 5.0 times the dimension line's width. Type 0x0000 has no extension.

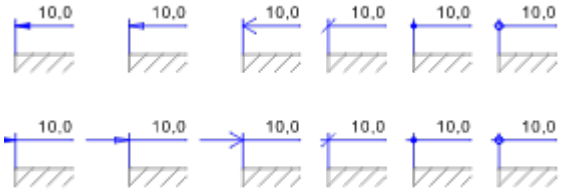
ArrowEndForm

[*int*] Equivalent to *ArrowStartForm*, but referring to the end-point.

ArrowEndMode

[*int*] Equivalent to *ArrowStartMode*, but referring to the end-point.

The following image shows all types of dimension arrows, once in normal presentation (upper row), once in rotated presentation with extended dimension line (lower row):



ExtStartDisplay

[*int*] Determines whether the dimension extension line at the start-point of the dimension shall be drawn or not. Possible values are:

0x0000 The dimension extension line at the start-point will not be drawn.
0x0001 The dimension extension line at the start-point will be drawn.

ExtEndDisplay

[*int*] Determines whether the dimension extension line at the end-point of the dimension shall be drawn or not. Possible values are:

0x0000 The dimension extension line at the end-point will not be drawn.
0x0001 The dimension extension line at the end-point will be drawn.

LineDisplay

[*int*] Determines whether to display dimension line and dimension extension line at all or not. Possibel values are:

0x0000 The dimension line and optionally the dimension extension lines will not be drawn.
0x0001 The dimension line and optionally the dimension extension lines will be drawn.

LineOrientation,
LineType,
LineDistMode

[*int*] The usage of these values depends on the object type the data block is defined in. Please refer to the description of the corresponding object type for detailed information.

LineDistance

[*double*] This value determines the distance bewteen the dimension line and corresponding object in

mm. It will only be used if *LineDistMode* is set to a non-zero value.

System

[*int*] Index of the coordinate system the dimension shall be based on. If this coordinate system is set to a distorting display (isometric or dimetric), the dimension will be calculated accordingly.

TEXTSTANDARD (Object Info Types)

Contents Syntax

```
typedef struct {
  FONTDEF        TextFont;
  XPROPERTY      TextXProperty;
  MATRIX         TextMatrix;

  double         CharDistance,
                 TabDistance,
                 LineDistance;
  int            TextMode;
} TEXTSTANDARD;
```

This structure contains settings for a standard or basic settings for a reference text.

Contents Element Description

TextFont

[*FONTDEF*] Decription of the font to be used for this text.

TextXProperty

[*XPROPERTY*] Properties of the text. The properties of the unit itself are not used for texts.

TextMatrix

[*MATRIX*] Display matrix of the text. The text will be multiplied with this matrix before display. The matrix contains all operations like translation, rotation, scaling and distortion. The font size is coded as scaling relative to 1 mm.

All transformations apply to the *complete* text, not to single characters, i.e. a rotation will rotate the complete text, not the single characters.

CharDistance

[*double*] The value *CharDistance* determines the gap between two characters. This gap is stated relative to the font size. A value of 0.1 at a font size of 10pt will result in a character gap of 1pt. Allowed values are -10.0 to +10.0. The default value for TrueType and device fonts should be 0.0, for internal fonts 0.125.

TabDistance

[*double*] The value *TabDistance* determines the distance between two tabulators. This distance is stated relative to the font size. A value of 4.0 at a font size of 5 mm will result in a tabulator distance of 20 mm. Allowed values are -100.0 to 100.0. The default value is 4.0.

LineDistance

[*double*] The value *LineDistance* determines the offset between two lines of text, measured from baseline to baseline. This offset is stated relative to the font size. A value of 1.2 at a font size of 10pt will lead to a line offset of 12pt. Allowed values are -100.0 to 100.0. The default value is 1.0.

TextMode

[*int*] The value *TextMode* states the position of the text relative to the insertion point. It can be one of the following values:

0x0000	The insertion point defines the left end-point of the text's baseline, i.e. the text will be displayed left-aligned.
0x0001	The insertion point defines the center-point of the text's baseline, i.e. the text will be displayed centered.
0x0002	The insertion point defines the right end-point of the text's baseline, i.e. the text will be displayed right-aligned.

TEXTFRAME (Object Info Types)

Contents Syntax

```
typedef struct {
  FONTDEF       TextFont;
  XPROPERTY     TextXProperty;
  double        TextSize;

  double        CharDistance,
                TabDistance,
                LineDistance;
  int           TextMode;
} TEXTFRAME;
```

This structure contains settings for a frame text.

Contents Element Description

TextFont

[*FONTDEF*] Decription of the font to be used for this text.

TextXProperty

[*XPROPERTY*] Properties of the text. The properties of the unit itself are not used for texts.

TextSize

[*double*] Size of the font. This size is handled differently depending of the font's type. If the font is a TrueType or device font, this value determines the typographical font size, i.e. the minimum offset between two lines of texts.
If the font is internal, this value determines the actual character height. Usually, an internal font will be displayed about 25% larger with the same value of *TextSize*.

CharDistance

[*double*] The value *CharDistance* determines the gap between two characters. This gap is stated relative to the font size. A value of 0.1 at a font size of 10pt will result in a character gap of 1pt. Allowed values are -10.0 to +10.0. The default value for TrueType and device fonts should be 0.0, for internal fonts 0.125.

TabDistance

[*double*] The value *TabDistance* determines the distance between two tabulators. This distance is stated relative to the font size. A value of 4.0 at a font size of 5 mm will result in a tabulator distance of 20 mm. Allowed values are -100.0 to 100.0. The default value is 4.0.

LineDistance

[*double*] The value *LineDistance* determines the offset between two lines of text, measured from baseline to baseline. This offset is stated relative to the font size. A value of 1.2 at a font size of 10pt will lead to a line offset of 12pt. Allowed values are -100.0 to 100.0. The default value is 1.0.

TextMode

[*int*] The value *TextMode* states the position of the text relative to the surrounding frame. It can be one of the following values:

0x0000	The text will be displayed left-aligned inside the surrounding frame.
0x0001	The text will be displayed centered inside the surrounding frame.
0x0002	The text will be displayed right-aligned inside the surrounding frame.
0x0003	The text will be displayed justified. The last line of each paragraph and lines containing a single word will be display left-aligned. For justification, the word gaps made up by the character ' ' (Ansi 32) will be

enlarged. Word gaps made up by the character ' ' (Ansi 160) will *not* be enlarged, i.e. Ansi 160 is a "fixed space".

TEXTREFERENCE (Object Info Types)

Contents **Syntax**

```
typedef struct {
  int           ArrowForm,
                ArrowMode,
                FrameForm;
  double        FrameOffset;
} TEXTREFERENCE;
```

This structure contains extended settings for a reference text.

Contents **Element Description**

ArrowForm

[*int*] The value *ArrowForm* determines the form of the arrow at the start-point (x,y) of the reference line. Possible values are:

0x0000	No dimension arrow.
0x0001	Filled triangular arrow. The triangle has an opening angle of 20° and a side length of 10.0 times the dimension line's width.
0x0002	Non-filled triangular arrow. The triangle has an opening angle of 20° and a side length of 10.0 times the dimension line's width.
0x0003	Open triangular arrow. The triangle has an opening angle of 60° and a side length of 10.0 times the dimension line's width.
0x0004	Diagonal stroke. The stroke has a relative angle of 45° to the dimension line and a length of 12.0 times the dimension line's width.
0x0005	Filled circle. The filled circle has a radius of 1.5 times the dimenion line's width.
0x0006	Non-filled circle. The filled circle has a radius of 2.5 times the dimenion line's width.

ArrowMode

[*int*] The value *ArrowMode* determines the form of the reference line. Allowed values are:

0x0000	Straight
0x0001	Bend, 45° angle at the arrow's side
0x0002	Bend, 45° angle at the text's side
0x0003	Bend, 90° angle at the arrow's side
0x0004	Bend, 90° angle at the text's side
0x0005	Horizontal
0x0006	Vertical

FrameForm

[*int*] The value *FrameForm* determines, which form the surrounding frame of the text shall have. Allowed values are:

0x0000	Rectangle
0x0001	Rhomb
0x0002	Circle
0x0003	Ellipse

FrameOffset

[*double*] The value *FrameOffset* determines the minimum distance between text itself and its surrounding frame.

CLIPSURFACE (Object Info Types)

Contents **Syntax**

```
typedef struct {
  XPROPERTY      XProperty;
  STR64          LibraryName,
                 BlockName;
  MATRIX         DisplayMatrix;
  int            IgnoreBlock;

  int            LibraryNum,
                 BlockNum;
  BOOL           NotFound;
} CLIPSURFACE;
```

This structure contains settings for a clipping surface.

Contents **Element Description**

XProperty

[*XPROPERTY*] Properties of the instance including transmission.

LibraryName

[*STR64*] Name of the library containg the desired block, maximum 63 characters. If the instance references a block located in the same drawing / library as the instance, set this name to "*".

BlockName

[*STR64*] Name of the block, maximum 63 characters. If the first character is 0x00, this instance is invalid and will neither be loaded nor stored!

If *LibraryName* is set to "*", the content of *BlockName* may have a special form. If the first character is '#' (Ansi 35), the block is a special, internally used and handled block (see TVG 4.0 Documentation (TVG40.HLP) - Entity "Group" and TVG 4.0 Documentation (TVG40.HLP) - Entity "Position Number"). The function of this block then depends on the character following the '#' sign. Such instances are only allowed inside drawings, *not* inside libraries!

DisplayMatrix

[*MATRIX*] Display matrix of the block. All entities stored in the block have to be multiplied with this matrix before display. It contains translation, rotation, scaling and distortion.

IgnoreBlock

[*int*] This value determines, how a clipping surface behaves during user interaction. Allowed values are:

0x0000	The block referenced by the clipping surface will be used during all operations. In this case, a clipping surface will behave like a surface plus an instance.
0x0001	The block referenced by the clipping surface is ignored during operations like object selection. This can be used to "hide" the internal structure of the clipping surface from the user. In this case, a clipping surface will behave like a standard filled surface, i.e. it can only be identified by clicking onto its outline.

LibraryNum, *BlockNum*

[*int*] Zero-based indices of the block in the internal cache. These values must not be modified!

NotFound

[*BOOL*] Indicates whether the referenced block has been found. This value must not be modified!

BITMAPREF (Object Info Types)

Contents **Syntax**

```
typedef struct {
  STR256        BitmapName;
  MATRIX        DisplayMatrix;
} BITMAPREF;
```

This structure contains settings for a bitmap reference.

Contents **Element Description**

BitmapName

[*STR256*] Filename of the bitmap to be displayed. The file must contain a valid Windows-style bitmap. The maximum size of such a bitmap is 32,000 by 32,000 pixels, using color depths of 1 bit, 4 bit, 8 bit or 24 bit.

DisplayMatrix

[*MATRIX*] Display matrix of the bitmap. The position stored in the matrix places the lower left corner of the bitmap, the scaling and rotation information determines the size and orientation of the display. The size is relative to the default bitmap resolution stored in the bitmap's header. If the bitmap does not contain a valid resolution information, its resolution is assumed to be 300 dpi.

BLOCK_HEADER (Data Block Types)

Contents **Syntax**

```
typedef struct {
  int           Size;
  short         Ident,
                Flag;

  short         BlockOwner,
                BlockType,

                ElemType,
                ElemCount;
} BLOCK_HEADER;
```

This structure contains basic information about a data block.

Contents **Element Description**

Size

[*int*] Size of the data block in bytes. Add this value to the data block's address to get the address of the next data block. This value must not be modified!

Ident

[*short*] Identification ID of the object. This value must not be modified!

Flag

[*short*] Status of the object. This value must not be modified!

BlockOwner

[*short*] This value is a unique identification of the creator of the module that created the data block. The value DB_OWNER_TOSO is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.
Third-party vendors that plan to implement an external module should contact TommySoftware® to receive their unique identification (see Getting Your Private Owner ID).

BlockType

[*short*] This value is an internal identification of the object. This value is used by the external module that created this object. For information about this value see the documentation of the module which should include a description of all user-defined objects used.
Data blocks created by third-party vendors should use values between 1000 and 29999 (inclusive), the other values are reserved for direct use by TommySoftware®.

ElemType

[*short*] Identifies the element type of the data block. Possible values are:

0x0000	Native Data Block
0x0001	BLOCK_LONG
0x0002	BLOCK_DOUBLE
0x0003	BLOCK_POINT
0x0004	BLOCK_COLORREF
0x0005	BLOCK_PROPERTY
0x0006	BLOCK_XPROPERTY
0x0007	BLOCK_FONTDEF
0x0008	BLOCK_TEXT
0x0009	BLOCK_BINARY

ElemCount

[*short*] Number of values that are stored in this data block. The value must be between 1 and 16000 (inclusive).

BLOCK_ANY (Data Block Types)

Contents **Syntax**

```
typedef struct {
  BLOCK_HEADER  Header;          // ElemType = 0..9, ElemCount = ?
} BLOCK_ANY;

typedef BLOCK_ANY       *BLOCK_PTR;
```

This structure describes a void data block.

Contents **Element Description**

Header

[*BLOCK_HEADER*] This structure contains general information about the data block.

BLOCK_LONG (Data Block Types)

Contents **Syntax**

```
typedef struct {
  BLOCK_HEADER  Header;          // ElemType = 1, ElemCount = # of LONGs ( >0 )
  long          Data[];
} BLOCK_LONG;

typedef BLOCK_LONG      *BLOCK_LONG_PTR;
```

This structure describes a data block storing "long" data types.

Contents **Element Description**

Header

[*BLOCK_HEADER*] This structure contains general information about the data block.

Data

[*long[]*] List of data elements stored in the data block.

BLOCK_DOUBLE (Data Block Types)

Syntax

```
typedef struct {
  BLOCK_HEADER  Header;          // ElemType = 2, ElemCount = # of DOUBLEs ( >0 )
  double        Data[];
} BLOCK_DOUBLE;

typedef BLOCK_DOUBLE    *BLOCK_DOUBLE_PTR;
```

This structure describes a data block storing "double" data types.

Element Description

Header

[*BLOCK_HEADER*] This structure contains general information about the data block.

Data

[*double[]*] List of data elements stored in the data block.

BLOCK_POINT (Data Block Types)

Contents **Syntax**

```
typedef struct {
  BLOCK_HEADER  Header;          // ElemType = 3, ElemCount = # of DPOINTs ( >0 )
  DPOINT        Data[];
} BLOCK_POINT;

typedef BLOCK_POINT     *BLOCK_POINT_PTR;
```

This structure describes a data block storing "DPOINT" data types.

Contents **Element Description**

Header

[*BLOCK_HEADER*] This structure contains general information about the data block.

Data

[*DPOINT[]*] List of data elements stored in the data block.

BLOCK_COLORREF (Data Block Types)

Contents **Syntax**

```
typedef struct {
  BLOCK_HEADER  Header;          // ElemType = 4, ElemCount = # of COLORREFs ( >0 )
  COLORREF      Data[];
} BLOCK_COLORREF;

typedef BLOCK_COLORREF  *BLOCK_COLORREF_PTR;
```

This structure describes a data block storing "COLORREF" data types.

Contents **Element Description**

Header

[*BLOCK_HEADER*] This structure contains general information about the data block.

Data

[*COLORREF[]*] List of data elements stored in the data block.

BLOCK_PROPERTY (Data Block Types)

Contents **Syntax**

```
typedef struct {
  BLOCK_HEADER  Header;          // ElemType = 5, ElemCount = # of PROPERTYs ( >0 )
  PROPERTY      Data[];
} BLOCK_PROPERTY;

typedef BLOCK_PROPERTY  *BLOCK_PROPERTY_PTR;
```

This structure describes a data block storing "PROPERTY" data types.

Contents **Element Description**

Header

[*BLOCK_HEADER*] This structure contains general information about the data block.

Data

[*PROPERTY[]*] List of data elements stored in the data block.

BLOCK_XPROPERTY (Data Block Types)

Contents **Syntax**

```
typedef struct {
  BLOCK_HEADER  Header;          // ElemType = 6, ElemCount = # of XPROPERTYs
( >0 )
  XPROPERTY     XProperty[];
} BLOCK_XPROPERTY;

typedef BLOCK_XPROPERTY *BLOCK_XPROPERTY_PTR;
```

This structure describes a data block storing "XPROPERTY" data types.

Contents **Element Description**

Header

[*BLOCK_HEADER*] This structure contains general information about the data block.

Data

[*XPROPERTY[]*] List of data elements stored in the data block.

BLOCK_FONTDEF (Data Block Types)

Contents **Syntax**

```
typedef struct {
  BLOCK_HEADER  Header;          // ElemType = 7, ElemCount = # of FONTDEFs ( >0 )
  FONTDEF       Data[];
} BLOCK_FONTDEF;

typedef BLOCK_FONTDEF    *BLOCK_FONTDEF_PTR;
```

This structure describes a data block storing "FONTDEF" data types.

Contents **Element Description**

Header

[*BLOCK_HEADER*] This structure contains general information about the data block.

Data

[*FONTDEF[]*] List of data elements stored in the data block.

BLOCK_TEXT (Data Block Types)

Contents **Syntax**

```
typedef struct {
  BLOCK_HEADER  Header;          // ElemType = 8, ElemCount = Text length ( <0
static, >0 dynamic )
  char          Text[];
} BLOCK_TEXT;

typedef BLOCK_TEXT      *BLOCK_TEXT_PTR;
```

This structure describes a data block storing texts.

Contents **Element Description**

Header

[*BLOCK_HEADER*] This structure contains general information about the data block.

Data

[*char[]*] List of data elements stored in the data block.

BLOCK_BINARY (Data Block Types)

Contents **Syntax**

```
typedef struct {
  BLOCK_HEADER  Header;          // ElemType = 9, ElemCount = Data length ( >0 )
  BYTE          Data[];
} BLOCK_BINARY;

typedef BLOCK_BINARY    *BLOCK_BINARY_PTR;
```

This structure describes a data block storing binary data.

Contents **Element Description**

Header

[*BLOCK_HEADER*] This structure contains general information about the data block.

Data

[*BYTE[]*] List of data elements stored in the data block.

BLOCK_ATTRIBUTE (Data Block Types)

Contents **Syntax**

```
typedef struct {
  BLOCK_HEADER  Header;          // ElemType = 0, ElemCount = Attribute length ( <0
static, >0 dynamic )
  STR32         Name;
  char          Text[];
} BLOCK_ATTRIBUTE;

typedef BLOCK_ATTRIBUTE         *BLOCK_ATTRIBUTE_PTR;
```

This structure describes a data block storing attributes.

Contents **Element Description**

Header

[*BLOCK_HEADER*] This structure contains general information about the data block.

Name

[*STR32*] Name of the attribute, up to 31 characters.

Text

[*char[]*] Content of the attribute.

BLOCK_DIMLINE (Data Block Types)

Contents **Syntax**

```
typedef struct {
  BLOCK_HEADER  Header;          // ElemType = 0, ElemCount = 0
  DIMLINE       Data;
} BLOCK_DIMLINE;

typedef BLOCK_DIMLINE           *BLOCK_DIMLINE_PTR;
```

This structure describes a data block storing "DIMLINE" data types.

Contents **Element Description**

Header

[*BLOCK_HEADER*] This structure contains general information about the data block.

Data

[*DIMLINE*] Data element stored in the data block.

BLOCK_DIMSMALL (Data Block Types)

Contents **Syntax**

```
typedef struct {
  BLOCK_HEADER  Header;          // ElemType = 0, ElemCount = 0
  DIMSMALL      Data;
} BLOCK_DIMSMALL;

typedef BLOCK_DIMSMALL           *BLOCK_DIMSMALL_PTR;
```

This structure describes a data block storing "DIMSMALL" data types.

Contents **Element Description**

Header

[*BLOCK_HEADER*] This structure contains general information about the data block.

Data

[*DIMSMALL*] Data element stored in the data block.

BLOCK_DIMLARGE (Data Block Types)

Contents **Syntax**

```
typedef struct {
  BLOCK_HEADER  Header;          // ElemType = 0, ElemCount = 0
  DIMLARGE      Data;
} BLOCK_DIMLARGE;

typedef BLOCK_DIMLARGE           *BLOCK_DIMLARGE_PTR;
```

This structure describes a data block storing "DIMLARGE" data types.

Contents **Element Description**

Header

[*BLOCK_HEADER*] This structure contains general information about the data block.

Data

[*DIMLARGE*] Data element stored in the data block.

BLOCK_TEXTSTANDARD (Data Block Types)

Contents **Syntax**

```
typedef struct {
  BLOCK_HEADER  Header;          // ElemType = 0, ElemCount = 0
  TEXTSTANDARD  Data;
} BLOCK_TEXTSTANDARD;

typedef BLOCK_TEXTSTANDARD      *BLOCK_TEXTSTANDARD_PTR;
```

This structure describes a data block storing "TEXTSTANDARD" data types.

Contents **Element Description**

Header

[*BLOCK_HEADER*] This structure contains general information about the data block.

Data

[*TEXTSTANDARD*] Data element stored in the data block.

BLOCK_TEXTFRAME (Data Block Types)

Contents **Syntax**

```
typedef struct {
  BLOCK_HEADER  Header;          // ElemType = 0, ElemCount = 0
  TEXTFRAME     Data;
} BLOCK_TEXTFRAME;

typedef BLOCK_TEXTFRAME          *BLOCK_TEXTFRAME_PTR;
```

This structure describes a data block storing "TEXTFRAME" data types.

Contents **Element Description**

Header

[*BLOCK_HEADER*] This structure contains general information about the data block.

Data

[*TEXTFRAME*] Data element stored in the data block.

BLOCK_TEXTREFERENCE (Data Block Types)

Contents **Syntax**

```
typedef struct {
  BLOCK_HEADER  Header;          // ElemType = 0, ElemCount = 0
  TEXTREFERENCE Data;
} BLOCK_TEXTREFERENCE;

typedef BLOCK_TEXTREFERENCE     *BLOCK_TEXTREFERENCE_PTR;
```

This structure describes a data block storing "TEXTREFERENCE" data types.

Contents **Element Description**

Header

[*BLOCK_HEADER*] This structure contains general information about the data block.

Data

[*TEXTREFERENCE*] Data element stored in the data block.

BLOCK_CLIPSURFACE (Data Block Types)

Contents **Syntax**

```
typedef struct {
  BLOCK_HEADER  Header;          // ElemType = 0, ElemCount = 0
  CLIPSURFACE  Data;
} BLOCK_CLIPSURFACE;

typedef BLOCK_CLIPSURFACE       *BLOCK_CLIPSURFACE_PTR;
```

This structure describes a data block storing "CLIPSURFACE" data types.

Contents **Element Description**

Header

[*BLOCK_HEADER*] This structure contains general information about the data block.

Data

[*CLIPSURFACE*] Data element stored in the data block.

BLOCK_BITMAPREF (Data Block Types)

Contents **Syntax**

```
typedef struct {
  BLOCK_HEADER  Header;          // ElemType = 0, ElemCount = 0
  BITMAPREF     Data;
} BLOCK_BITMAPREF;

typedef BLOCK_BITMAPREF          *BLOCK_BITMAPREF_PTR;
```

This structure describes a data block storing "BITMAPREF" data types.

Contents **Element Description**

Header

[*BLOCK_HEADER*] This structure contains general information about the data block.

Data

[*BITMAPREF*] Data element stored in the data block.

BLOCK_T001 (Data Block Types)

Syntax

```
typedef struct {
  BLOCK_HEADER  Header;          // ElemType = 0, ElemCount = 0
  double        ArcAngle;
} BLOCK_T001;

typedef BLOCK_T001      *BLOCK_T001_PTR;
```

This structure contains temporary data precalculated by the serving application. Data stored in such a data block is subject to change at any time.

Element Description

Temporary data blocks should not be used by external modules. Simply ignore them.

BLOCK_T002 (Data Block Types)

Syntax

```
typedef struct {
  BLOCK_HEADER  Header;          // ElemType = 0, ElemCount = 0
  double        StartAngle,
                ArcAngle;
} BLOCK_T002;

typedef BLOCK_T002      *BLOCK_T002_PTR;
```

This structure contains temporary data precalculated by the serving application. Data stored in such a data block is subject to change at any time.

Element Description

Temporary data blocks should not be used by external modules. Simply ignore them.

BLOCK_T003 (Data Block Types)

Contents **Syntax**

```
typedef struct {
  BLOCK_HEADER  Header;          // ElemType = 0, ElemCount = 0
  double        ArrowStartAngle,
                ArrowEndAngle;
} BLOCK_T003;

typedef BLOCK_T003      *BLOCK_T003_PTR;
```

This structure contains temporary data precalculated by the serving application. Data stored in such a data block is subject to change at any time.

Contents **Element Description**

Temporary data blocks should not be used by external modules. Simply ignore them.

BLOCK_T004 (Data Block Types)

Syntax

```
typedef struct {
  BLOCK_HEADER  Header;          // ElemType = 0, ElemCount = 0
  double        ArcAngle,
                StartAngle,
                EndAngle,
                ArrowStartAngle,
                ArrowEndAngle;
} BLOCK_T004;

typedef BLOCK_T004      *BLOCK_T004_PTR;
```

This structure contains temporary data precalculated by the serving application. Data stored in such a data block is subject to change at any time.

Element Description

Temporary data blocks should not be used by external modules. Simply ignore them.

BLOCK_T005 (Data Block Types)

Contents **Syntax**

```
typedef struct {
  BLOCK_HEADER  Header;          // ElemType = 0, ElemCount = 0
  int           ArrowStartForm,
                ArrowEndForm;
  double        ArrowStartAngle,
                ArrowEndAngle,
                NumberAngle;
  BOOL          ExtStartDisplay,
                ExtEndDisplay,
                LineDisplay;
  double        x1, y1,
                x2, y2,
                x3, y3,
                x4, y4,
                x7, y7;
} BLOCK_T005;

typedef BLOCK_T005      *BLOCK_T005_PTR;
```

This structure contains temporary data precalculated by the serving application. Data stored in such a data block is subject to change at any time.

Contents **Element Description**

Temporary data blocks should not be used by external modules. Simply ignore them.

BLOCK_T006 (Data Block Types)

Contents **Syntax**

```
typedef struct {
  BLOCK_HEADER  Header;          // ElemType = 0, ElemCount = 0
  int           ArrowStartForm,
                ArrowEndForm;
  double        ArcAngle,
                StartAngle,
                EndAngle,
                ArrowStartAngle,
                ArrowEndAngle,
                NumberAngle;
  BOOL          ExtStartDisplay,
                ExtEndDisplay,
                LineDisplay;
  double        x1, y1,
                x2, y2,
                x3, y3,
                x4, y4,
                x5, y5,
                x6, y6,
                x7, y7;
} BLOCK_T006;

typedef BLOCK_T006      *BLOCK_T006_PTR;
```

This structure contains temporary data precalculated by the serving application. Data stored in such a data block is subject to change at any time.

Contents **Element Description**

Temporary data blocks should not be used by external modules. Simply ignore them.

BLOCK_T007 (Data Block Types)

Syntax

```
typedef struct {
  BLOCK_HEADER  Header;          // ElemType = 0, ElemCount = 0
  double        NumberAngle;
  double        x7, y7;
} BLOCK_T007;

typedef BLOCK_T007      *BLOCK_T007_PTR;
```

This structure contains temporary data precalculated by the serving application. Data stored in such a data block is subject to change at any time.

Element Description

Temporary data blocks should not be used by external modules. Simply ignore them.

BLOCK_T008 (Data Block Types)

Syntax

```
typedef struct {
  BLOCK_HEADER  Header;          // ElemType = 0, ElemCount = 0
  int           ArrowForm,
                ArrowMode;
  double        ArrowAngle;
  int           FrameForm;
  double        FrameOffset;
  double        x1, y1,
                x2, y2,
                x3, y3,
                x4, y4,
                x5, y5,
                x6, y6,
                x7, y7;
} BLOCK_T008;

typedef BLOCK_T008      *BLOCK_T008_PTR;
```

This structure contains temporary data precalculated by the serving application. Data stored in such a data block is subject to change at any time.

Element Description

Temporary data blocks should not be used by external modules. Simply ignore them.

UNIT_MEMORY (Unit Types)

Contents **Syntax**

```
typedef struct {
  long          Units,
                Bytes;
  OBJPTR        Prev,
                Next;
} UNIT_MEMORY;
```

This structure contains basic memory-related information about an entity.

Contents **Element Description**

Units

[*long*] Size of the complete unit (including all data block) in units of 16 bytes. This value must not be modified!

Bytes

[*long*] Size of the complete unit (inlcuding all data blocks) in bytes. This value must not be modified!

Prev

[*OBJPTR*] Address of the previous object in the current list. This value must not be modified!

Next

[*OBJPTR*] Address of the next object in the current list. This value must not be modified!

UNIT_HEADER (Unit Types)

Contents **Syntax**

```
typedef struct {
  DRECT          Rect;

  int            Ident,
                 Flag,

                 Undo1,
                 Undo2;

  int            UnitOwner,
                 UnitType;
} UNIT_HEADER;
```

This structure contains basic application-related information anout an entity.

Contents **Element Description**

Rect

[*DRECT*] Surrounding frame of the unit. This value must not be modified!

Ident

[*int*] Identification index. This value must not be modified!

Flag

[*int*] Bitwise OR combination of some general flags indicating the state of the unit. This value must not be modified! Possible values are:

FLAG_SELECT
: This flag is set if the unit is permanently selected.

FLAG_IDENT
: This flag is set if the unit is currently highlighted for final selection due to an ambigous identification.

FLAG_USE
: This flag is set if the unit has been identified in a multi-object identification.

FLAG_POINT
: This flag is set if at least one definition point inside the unit is selected.

FLAG_INTERNAL
: This flag is set if the unit is a temporary unit.

FLAG_NODISPLAY
: This flag is set if the unit lies in layer that will not be displayed on screen.

FLAG_NOOUTPUT
: This flag is set if the unit lies in layer that will not be output to printer, clipboard etc.

FLAG_FROZEN
: This flag is set if the unit lies in a frozen layer.

FLAG_IDLE
: This flag is set if the unit lies in an ignored layer.

Undo1,
Undo2

[*int*] Undo time interval inside which the unit is valid, based on the internal undo count. These values must not be modified!

UnitOwner

[*int*] This value is a unique identification of the external module that created the unit. The value DB_OWNER_TOSO is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement an external module that produces user-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

UnitType

[*int*] Specifies the type of the unit. Possible values are:

TYPE_OBJECT

See UNIT_OBJECT.

TYPE_BLOCK

See UNIT_BLOCK.

TYPE_INST

See UNIT_INSTANCE.

TYPE_USER

See UNIT_USER.

UNIT_ANY (Unit Types)

Contents **Syntax**

```
typedef struct {
  UNIT_MEMORY   Memory;
  UNIT_HEADER   Header;

  char          Data[];
} UNIT_ANY;

typedef UNIT_ANY        *UNIT_PTR;
```

This structure describes a non-specific entity.

Contents **Element Description**

Memory

[*UNIT_MEMORY*] This structure contains internal information of the storage management. Do not modify any of the values stored in this structure!

Header

[*UNIT_HEADER*] This structure contains internal information about the unit.

Data

[*char[]*] This data section contains a list of data blocks. As the memory-internal format of data blocks is rather complex, this section should usually not be modified.

UNIT_OBJECT (Unit Types)

Contents **Syntax**

```
typedef struct {
  UNIT_MEMORY    Memory;
  UNIT_HEADER    Header;          // UnitType = 0

  XPROPERTY      XProperty;
  int            ObjectType;

  int            FontNum;

  char           Data[];
} UNIT_OBJECT;

typedef UNIT_OBJECT      *UNIT_OBJECT_PTR;
```

This structure describes an entity of type TYPE_OBJECT.

Contents **Element Description**

Memory

[*UNIT_MEMORY*] This structure contains internal information of the storage management. Do not modify any of the values stored in this structure!

Header

[*UNIT_HEADER*] This structure contains internal information about the unit.

XProperty

[*XPROPERTY*] Properties of the object including transmission.

ObjectType

[*int*] Type of the object (like "line", "circle" etc.). Possible object types are:

OBJ_LINE

Straight line. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_HATCH

Collection of curves, usually used for hatchings. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_CIRCLE

Circle. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_ARC

Circular arc. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_SECTOR

Circular sector. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_SEGMENT

Circular segment. For information about the internal data block structure of this object type, see

the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_ZIGZAG

Zigzag line. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_SPLINE

Spline curve. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_CURVE

Curve consisting of lines, Bézier curves and circular arcs. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_SURFACE

Surface consisting of lines, Bézier curves and circular arcs. May contain nested contours. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_ELLIPSE

Ellipse. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_EARC

Elliptical arc. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_ESECTOR

Elliptical sector. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_ESEGMENT

Elliptical segment. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_DLINE

Straight dimension line. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_DARC

Curved dimension line. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_DDISTANCE

Dimension with straight dimension line, used for length and distance dimensioning. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_DRADIUS

Dimension with straight dimension line, used for radius dimensioning. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_DDIAMETER

Dimension with straight dimension line, used for diameter dimensioning. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_DANGLE

Dimension with curved dimension line, used for angle dimensioning. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_DARCLENGTH

Dimension with curved dimension line, used for arc-length dimensioning. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_DCOORDINATE

Dimension without dimension line, used for coordinate dimensioning. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_DAREA

Dimension without dimension line, used for area dimensioning. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_DPERIMETER

Dimension without dimension line, used for perimeter dimensioning. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_TEXTSTANDARD

Standard text defined by an insertion point. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_TEXTFRAME

Frame text defined by a surrounding frame. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_TEXTREFERENCE

Reference text defined by an insertion point, plus a frame and an arrow. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_COMMENT

Comment (non-printed object). For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_MARK

Marking (optionally printed object). For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_CLIPSURFACE

Clipping surface. For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_BITMAPREF

Bitmap reference (optionally printed object). For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_GEOLINE

Geometry line (optionally printed object). For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_GEOCIRCLE

Geometry circle (optionally printed object). For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

OBJ_GEOELLIPSE

Geometry ellipse (optionally printed object). For information about the internal data block structure of this object type, see the corresponding paragraph in the TVG 4.0 Documentation (TVG40.HLP).

FontNum

[*int*] Zero-based index of the font in the internal font list. This value must not be modified!

Data

[*char[]*] This data section contains a list of data blocks. As the memory-internal format of data blocks is rather complex, this section should usually not be modified.

UNIT_BLOCK (Unit Types)

Contents **Syntax**

```
typedef struct {
  UNIT_MEMORY    Memory;
  UNIT_HEADER    Header;          // UnitType = 2

  XPROPERTY      XProperty;
  STR64          BlockName;
  DRECT          BlockRect;

  OBJPTR         BlockFirst,
                 BlockLast;

  char           Data[];
} UNIT_BLOCK;

typedef UNIT_BLOCK       *UNIT_BLOCK_PTR;
```

This structure describes an entity of type TYPE_BLOCK.

Contents **Element Description**

Memory
: [*UNIT_MEMORY*] This structure contains internal information of the storage management. Do not modify any of the values stored in this structure!

Header
: [*UNIT_HEADER*] This structure contains internal information about the unit.

XProperty
: [*XPROPERTY*] Properties of the block including transmission (a block does usually not contain any transmission information unless explicitly enforced by the user).

BlockName
: [*STR64*] Name of the block, up to 63 characters. If the first character of the block's name is 0x00, the block is invalid and will neither be loaded nor saved!
If the first character of the block's name is '#' (Ansi 35), the block is a special, internally used and handled block (see TVG 4.0 Documentation (TVG40.HLP) - Entity "Group" and TVG 4.0 Documentation (TVG40.HLP) - Entity "Position Number"). The function of this block then depends on the character following the '#' sign. Such blocks are only allowed inside drawings, *not* inside libraries!

BlockRect
: [*DRECT*] Rectangular frame surrounding the character cell, valid only if the block is a character inside a font library (see TVG 4.0 Documentation (TVG40.HLP) - Font Libraries). In this case, (*BlockRect.x1* / *BlockRect.y1*) is the lower left corner; (*BlockRect.x2* / *BlockRect.y2*) is the upper right corner of the cell.

BlockFirst,
BlockLast
: [*OBJPTR*] Adresses of the first and the last entity contained in the block. These values must not be modified!

Data
: [*char[]*] This data section contains a list of data blocks. As the memory-internal format of data blocks is rather complex, this section should usually not be modified.

UNIT_INSTANCE (Unit Types)

Contents **Syntax**

```
typedef struct {
  UNIT_MEMORY    Memory;
  UNIT_HEADER    Header;          // UnitType = 3

  XPROPERTY      XProperty;
  STR64          LibraryName,
                 BlockName;
  MATRIX         DisplayMatrix;

  int            LibraryNum,
                 BlockNum;
  BOOL           NotFound;

  char           Data[];
} UNIT_INSTANCE;

typedef UNIT_INSTANCE    *UNIT_INSTANCE_PTR;
```

This structure describes an entity of type TYPE_INSTANCE.

Contents **Element Description**

Memory
: [*UNIT_MEMORY*] This structure contains internal information of the storage management. Do not modify any of the values stored in this structure!

Header
: [*UNIT_HEADER*] This structure contains internal information about the unit.

XProperty
: [*XPROPERTY*] Properties of the instance including transmission.

LibraryName
: [*STR64*] Name of the library containg the desired block, maximum 63 characters. If the instance references a block located in the same drawing / library as the instance, set this name to "*".

BlockName
: [*STR64*] Name of the block, maximum 63 characters. If the first character is 0x00, this instance is invalid and will neither be loaded nor stored!
If *LibraryName* is set to "*", the content of *BlockName* may have a special form. If the first character is '#' (Ansi 35), the block is a special, internally used and handled block (see TVG 4.0 Documentation (TVG40.HLP) - Entity "Group" and TVG 4.0 Documentation (TVG40.HLP) - Entity "Position Number"). The function of this block then depends on the character following the '#' sign. Such instances are only allowed inside drawings, *not* inside libraries!

DisplayMatrix
: [*MATRIX*] Display matrix of the block. All entities stored in the block have to be multiplied with this matrix before display. It contains translation, rotation, scaling and distortion.

LibraryNum,
BlockNum
: [*int*] Zero-based indices of the block in the internal cache. These values must not be modified!

NotFound
: [*BOOL*] Indicates whether the referenced block has been found. This value must not be modified!

Data

[*char[]*] This data section contains a list of data blocks. As the memory-internal format of data blocks is rather complex, this section should usually not be modified.

UNIT_USER (Unit Types)

Contents **Syntax**

```
typedef struct {
  UNIT_MEMORY    Memory;
  UNIT_HEADER    Header;          // UnitType = 9

  XPROPERTY      XProperty;
  STR64          LibraryName,
                 BlockName;
  MATRIX         DisplayMatrix;
  int            UserType;

  int            LibraryNum,
                 BlockNum;
  BOOL           NotFound;

  char           Data[];
} UNIT_USER;

typedef UNIT_USER       *UNIT_USER_PTR;
```

This structure describes an entity of type TYPE_USER.

Contents **Element Description**

Memory
[*UNIT_MEMORY*] This structure contains internal information of the storage management. Do not modify any of the values stored in this structure!

Header
[*UNIT_HEADER*] This structure contains internal information about the unit.

XProperty
[*XPROPERTY*] Properties of the display instance including transmission.

LibraryName
[*STR64*] Name of the library containg the desired display block, maximum 63 characters. If the instance references a block located in the same drawing / library as the instance, set this name to "*".

BlockName
[*STR64*] Name of the display block, maximum 63 characters. If the first character is 0x00, this instance is invalid and will neither be loaded nor stored!
If *LibraryName* is set to "*", the content of *BlockName* may have a special form. If the first character is '#' (Ansi 35), the block is a special, internally used and handled block (see TVG 4.0 Documentation (TVG40.HLP) - Entity "Group" and TVG 4.0 Documentation (TVG40.HLP) - Entity "Position Number"). The function of this block then depends on the character following the '#' sign. Such instances are only allowed inside drawings, *not* inside libraries!

DisplayMatrix
[*MATRIX*] Display matrix of the display block. All entities stored in the block have to be multiplied with this matrix before display. It contains translation, rotation, scaling and distortion.

UserType
[*int*] Internal identification of the user-defined entity. For detailed information about this identification, see the documentation of the program or external module that created this entity.

LibraryNum,

BlockNum

[*int*] Zero-based indices of the display block in the internal cache. These values must not be modified!

NotFound

[*BOOL*] Indicates whether the referenced display block has been found. This value must not be modified!

Data

[*char[]*] This data section contains a list of data blocks. As the memory-internal format of data blocks is rather complex, this section should not be modified unless the module has created this object or at least knows its internal structure.

MENU_DATA (Module Interface Types)

Contents **Syntax**

```
typedef struct {
  LPSTR         MenuEntry,
                Description;
} MENU_DATA;
```

This structure contains a single menu entry / command title text pair.

Contents **Element Description**

MenuEntry

[*LPSTR*] Address of a null-terminated string that contains a command's menu entry that will be listed in a menu or submenu.

All command entries of a standard command module should start with a shortcut key separated from the command name by two spaces, e.g. `"&1  Command One"` . Usually, the shortcuts should be the digits '1' to '0', followed by capital characters 'A' to 'H'.

If the module does have a submenu, this submenu should end with a separator followed by the menu item `"&+  About..."` in English modules, `"&+  Infos..."` in German modules or the equivalent in other foreign languages. This menu item should display a dialog window stating the module's name, its creator and the current version number.

If the module is either an import or export filter, *MenuEntry* should contain a brief description of the file format(s) handled by the filter, e.g. `"DXF Drawings (*.DXF)"` . In the menu, a shortcut key will appear before this text. As a result, this text should *not* contain a shortcut key definition!

Description

[*LPSTR*] Address of a null-terminated string that contains a command's complete description. If this command lies within a custon menu or submenu, it should state the module's name (possibly shortened) and the command's name, separated by a space and the character '>' (Ansi 62). If, e.g., *MenuEntry* points to the submenu command entry `"&1  Command One"` , and the module is titled `"Sample Module"` , then *Description* should point to the command name `"Sample Module >Command One"` . If the module does only have one command, the module name alone will suffice.

If the module is either an import or export filter, *Description* must contain the module's complete name. This name should contain the phrase `"Import >"` if the module is an import filter, or `"Export >"` if the module is an export filter, plus the description of the file format(s) handled by the filter.

COMMAND_DATA (Module Interface Types)

Contents **Syntax**

```
typedef struct {
  int           CommandMode,
                PointNumber,
                PointTypes    [TVG_INPUT_DEF_MAX],
                PointRelative[TVG_INPUT_DEF_MAX];
  LPSTR         PointNames    [TVG_INPUT_DEF_MAX];
} COMMAND_DATA;
```

This structure contains a command description. It states the point sequence that is to be entered by the user.

Contents **Element Description**

CommandMode

[*int*] This value determines what kind of command is coded in this structure. Possible values are:

COMMAND_DIRECT

The command does not require any point entry, i.e. it will be executed immediately.

COMMAND_LOCAL

The command requires a fixed number of point entries and is local. "Local" means that the command is active as long as the user does not terminate it. After doing so, the previously active command will be activated again. An example of a "local" command is **Library > Block > Insert...** .

COMMAND_FIXED

The command requires a fixed number of point entries and is global. "Global" means that this command will stay active until another global command is selected instead. An example of a "global" command with a fixed point number is **Draw > Line > Standard**.

COMMAND_VARIABLE

The command requires a variable number of point entries and is global. "Global" means that this command will stay active until another global command is selected instead. An example of a "global" command with a variable point number is **Draw > Line > Polyline**.

PointNumber

[*int*] This value determines the number of point types in the PointTypes array. If *CommandMode* is either COMMAND_LOCAL or COMMAND_FIXED, this is the number of points to be entered by the user.

If *CommandMode* is COMMAND_VARIABLE, this values determines the number of predefined point types. The last point type will automatically be used for all further points (see below). Valid range: 0 <= Value < TVG_INPUT_DEF_MAX.

PointTypes

[*int[]*] This array contains the point types to be entered by the user. For a description of all available point types, see Point Types.

If *CommandMode* is COMMAND_VARIABLE, the value of the last valid *PointTypes* entry determines the point types for all further points

PointRelative

[*int[]*] This array contains information on relativeness of the coordinate display of each point. It contains the index of the point to which the current shall be relative, or -1 if no relative point does

exist. E.g., during a line input, the start-point (index 0) has no relative point, so *PointRelative[0]* is set to -1. The end-point (index 1) has the start-point (index 0) as its relative point, so *PointRelative[1]* is set to 0.

If *CommandMode* is COMMAND_VARIABLE, the value of the last valid *PointRelative* entry determines to what points all further points will be relative. Possible values are:

-2	All points will be relative to their previous point (like in a polyline)
-1	All points will have no relative value
0	All points will be relative to the first point
1	All points will be relative to the second point
2	etc.

PointNames

[*LPSTR[]*] This array contains the point names of the point to be entered by the user. If an entry is NULL, the default name for the corresponding point type is used. For a description of all available point types, see Point Types.

If *CommandMode* is COMMAND_VARIABLE, the value of the last valid *PointNames* entry determines the point names for all further points

MODULE_COMMAND_DATA (Module Interface Types)

Contents **Syntax**

```
typedef struct {
  int           Type;
  COMMAND_DATA  InputData;
  MENU_DATA     MenuData;
  HBITMAP       IconHandle;
  int           IconXOffset,
                IconYOffset,
                IconMode;
} MODULE_COMMAND_DATA;
```

This structure contains a command's complete description, i.e. its title, its menu appearence, its icon and the points to be entered by the user.

Contents **Element Description**

Type

[*int*] The use of this value depends on whether this structure is used to identify a module or a module's command.
When identifying a module, this value determines the module's intended use (command extension, import filter or export filter). Allowed values are:

MODULETYPE_CUSTOM
: The module contains one or more external commands. It will be listed as a separate menu.

MODULETYPE_IMPORT
: The module contains one or more import filters. It will be listed in the import filter list.

MODULETYPE_EXPORT
: The module contains one or more export filters. It will be listed in the export filter list.

MODULETYPE_FILE
: The module contains one or more external commands. It will be listed in the menu "File" and in the module list.

MODULETYPE_EDIT
: The module contains one or more external commands. It will be listed in the menu "Edit" and in the module list.

MODULETYPE_CONFIG
: The module contains one or more external commands. It will be listed in the menu "Configure" and in the module list.

MODULETYPE_SHAPE
: The module contains one or more external commands. It will be listed in the menu "Shape" and in the module list.

MODULETYPE_DRAW
: The module contains one or more external commands. It will be listed in the menu "Draw" and in the module list.

MODULETYPE_GEO
: The module contains one or more external commands. It will be listed in the menu "Geometry"

and in the module list.

MODULETYPE_TRIM
: The module contains one or more external commands. It will be listed in the menu "Trimming" and in the module list.

MODULETYPE_DIM
: The module contains one or more external commands. It will be listed in the menu "Dimension" and in the module list.

MODULETYPE_LIB
: The module contains one or more external commands. It will be listed in the menu "Library" and in the module list.

MODULETYPE_EXTRA
: The module contains one or more external commands. It will be listed in the menu "Extra" and in the module list.

MODULETYPE_HELP
: The module contains one or more external commands. It will be listed in the menu "Help" and in the module list.

When identifying a module's command, this value contains command type flags of the module's custom menu or submenu entry. It is a bitwise OR combination of the following flags:

IDM_ENTRY
: If this flag is set, the corresponding command is a top-level menu entry.

IDM_POPUP_LABEL
: If this flag is set, the corresponding command is a submenu label. Such a command should be followed by a sequence of commands where IDM_POPUP_ENTRY is set.

IDM_POPUP_ENTRY
: If this flag is set, the corresponding command is a submenu entry.

IDM_DEFAULT
: If this flag is set, the corresponding command is the default command of a menu or submenu, i.e. the command that will be started if the user double-click on the corresponding menu or submenu in the popup menu.

IDM_SEPARATOR
: If this flag is set, the corresponding command is a separating line. The corresponding *MenuData* may be empty, i.e. the *MENU_DATA* component may contain NULL pointers.

IDM_END
: This flag is used to terminate the command list. The entry in which IDM_END is set will *not* be used.

InputData
: [*COMMAND_DATA*] This structure contains information about the required point entries to be performed by the user.

MenuData
: [*MENU_DATA*] This structure contains a textual description of a command or menu and the corresponding menu entry text.

IconHandle
: [*HBITMAP*] This bitmap handle identifies the bitmap that contains the command's or menu's icon. This bitmap must be a monochrome bitmap. Bits set to 0 ("black pixel") will be displayed in foreground color (usually dark red), bits set to 1 ("white pixel") will be displayed in background color

(usually light gray), i.e. the bitmap should be created black on white.
This bitmap handle must be valid throughout the module's lifetime, i.e. it should have been be created inside the TosoModuleInit procedure and should not be deleted until inside the TosoModuleExit procedure!

IconXOffset

IconYOffset

[*int*] These two values determine the offset of the command's or menu's icon inside the bitmap in pixels, starting from the upper left corner with positive value going right and down. If used as a command icon, the serving application will use a square section of 40 by 40 pixels starting at the pixel specified by the offsets. If used as a menu icon, the serving application will use a square of 24 by 24 pixels.

IconMode

[*int*] This value determines whether the given *IconHandle* is valid or not. If *IconMode* is set to zero, the bitmap identified by *IconHandle* must contain the command's or menu's icon at the location specified by *IconXOffset* and *IconYOffset*.
If *IconMode* is a positive non-zero value, this value is interpreted as a command identifier of an internal command whose command icon is to be used instead (e.g. IDM_DRAW_POLYLINE for a polyline). If *IconMode* does not specify a valid command identifier (e.g. -1), the default icon for external commands is used. In the latter two cases, the values of *IconHandle*, *IconXOffset* and *IconYOffset* will be ignored. These two cases are *not* available for menu icons, i.e. a menu must *always* supply a valid *IconHandle* plus *IconXOffset* and *IconYOffset*!

MODULE_PROC (Module Interface Types)

Contents **Syntax**

```
typedef struct {
  TOSOINPUTPOINTINIT_PROC       InputPointInitProc;
  TOSOINPUTPOINTMOVE_PROC       InputPointMoveProc;
  TOSOINPUTPOINTEXIT_PROC       InputPointExitProc;
  TOSOINPUTDISPLAY_PROC         InputDisplayProc;
  TOSOINPUTPARAMETER_PROC       InputParameterProc;
  TOSOINPUTCANCEL_PROC          InputCancelProc;
  TOSOINPUTFINISH_PROC          InputFinishProc;
} MODULE_PROC;
```

This structure contains all callback procedures that a module can provide for command handling.

Contents **Element Description**

InputPointInitProc

[*TOSOINPUTPOINTINIT_PROC*] This callback procedure is called at the beginning of each point entry. If this value is NULL, the serving application continues as if InputPointInitProc had returned INPUT_OK.

InputPointMoveProc

[*TOSOINPUTPOINTMOVE_PROC*] This callback procedure is called each time the user move the cursor. If this value is NULL, the serving application continues as if InputPointMoveProc had returned TRUE.

InputPointExitProc

[*TOSOINPUTPOINTEXIT_PROC*] This callback procedure is called at the end of each point entry. If this value is NULL, the serving application continues as if InputPointExitProc had returned INPUT_OK.

InputDisplayProc

[*TOSOINPUTDISPLAY_PROC*] This callback procedure is called each time the current input status needs to be updated. If this value is NULL, the serving application assumes that no input status has to be drawn.

InputParameterProc

[*TOSOINPUTPARAMETER_PROC*] This callback procedure is called if the user wants to edit the command's parameters. If this value is NULL, the serving application assumes that the command does not offer any parameters to be edited.

InputCancelProc

[*TOSOINPUTCANCEL_PROC*] This callback procedure is called if the user presses the right mouse button during point entry of a command requiring a variable point number. If this value is NULL, the serving application continues as if InputCancelProc had returned INPUT_FINISH, i.e. the command is finished based on all previously entered points.

InputFinishProc

[*TOSOINPUTFINISH_PROC*] This callback procedure is called if the command's point entry has been finished successfully. This value must not be NULL!

MODULE_ID (Module Interface Types)

Contents **Syntax**

```
typedef struct {
  short                 OwnerID,
                        ModuleID;
  int                   ModuleCTRL;
  MODULE_PROC           ModuleProc;
  MODULE_COMMAND_DATA   ModuleData;
  MODULE_COMMAND_DATA*  CommandData;
} MODULE_ID;
```

This structure contains a module's complete command description. This structure list all commands supplied by the module and states the callback procedures required to handle these commands.

Contents **Element Description**

OwnerID

[*short*] This value is a unique identification of the creator of the module. The value DB_OWNER_TOSO is reserved for use by TommySoftware®.
Third-party vendors that plan to implement an external module should contact TommySoftware® to receive their unique identification (see Getting Your Private Owner ID).

ModuleID

[*short*] This value is a module identification. This value can freely be managed by the module's creator. Be sure to keep this ID unique over all modules created. This is important to avoid interference between two modules of the same creator.

ModuleCTRL

[*int*] This value determines on which levels of the application this module will run on. It allows modules to assure that they will not be used on non-registered versions or on demo versions. The value is a bitwise OR combination of the following values:

MODULECTRL_ALL
: Module may run on any program version.

MODULECTRL_DEMO
: Module may run on a demo version.

MODULECTRL_TEST
: Module may run on a trial version.

MODULECTRL_LEVEL1
: Module may run on a level 1 version.

MODULECTRL_LEVEL2
: Module may run on a level 2 version.

MODULECTRL_LEVEL3
: Module may run on a level 3 version.

MODULECTRL_LEVEL4
: Module may run on a level 4 version.

MODULECTRL_CHECKED
: Module will only run if it has been approved. This flag is reserved for use by TommySoftware®

and approved suppliers. A module marked with this flag will only run if it has either been installed using the proper installation program, or if the user enters the corrent module code during first startup.
This flag will only be checked in full versions. In demo and test versions, the flags MODULECTRL_DEMO and MODULECTRL_TEST determine whether the module will work or not, independent of any approval.

ModuleProc

[*MODULE_PROC*] This structure contains the addresses of callback procedures required to handle user input during the execution of module-defined command.

ModuleData

[*MODULE_COMMAND_DATA*] This structure contains information about the module's main menu entry and its icon. The use of this information depends on the stated *ModuleData->Type*:

MODULETYPE_CUSTOM

In this case, *ModuleData* contains the description of the menu and its menu icon. *CommandType* and *CommandData* should contain further valid command descriptions to build the menu.

MODULETYPE_IMPORT
MODULETYPE_EXPORT

In these cases, *ModuleData* contains the one and only command that the filter offers. Import and export filters are not allowed to create custom menus or submenus! In these cases, both *CommandType* and *CommandData* should be set to NULL.

MODULETYPE_FILE
MODULETYPE_EDIT
MODULETYPE_CONFIG
MODULETYPE_SHAPE
MODULETYPE_DRAW
MODULETYPE_GEO
MODULETYPE_TRIM
MODULETYPE_DIM
MODULETYPE_LIB
MODULETYPE_EXTRA
MODULETYPE_HELP

In these cases, *ModuleData* contains the description of the command that is to be added to the top level of the corresponding menu. If *CommandType* and *CommandData* contain further valid command descriptions, this command will serve as the popup menu label, else it will be a standard menu item.

In all three cases, the resulting command identifier of this command it zero. Commands stored in *CommandType* and *CommandData* receive command identifiers beginning with one.

CommandData

[*MODULE_DATA**] This array contains information about the module's custom menu or submenu entries and their icons. The use of this information depends on the stated *ModuleData->Type*:

MODULETYPE_CUSTOM

In this case, *CommandData* contains the description of the commands and submenus to be listed in the custom menu.

MODULETYPE_IMPORT
MODULETYPE_EXPORT

In these cases, *CommandData* should be to NULL.

MODULETYPE_FILE
MODULETYPE_EDIT

MODULETYPE_CONFIG
MODULETYPE_SHAPE
MODULETYPE_DRAW
MODULETYPE_GEO
MODULETYPE_TRIM
MODULETYPE_DIM
MODULETYPE_LIB
MODULETYPE_EXTRA
MODULETYPE_HELP

In these cases, *CommandData* contains command descriptions to be listed in a submenu.

In all three cases, the resulting command identifiers of these command begin with `1`, going up to TVG_MODULE_MENU_MAX.

MODULE_DATA (Module Interface Types)

Contents **Syntax**

```
typedef struct {
  MODULE_ID                    ID;
  HINSTANCE                    Handle;
  TOSOMODULECOMMAND_PROC       ProcCommand;
  TOSOMODULEMODIFY_PROC        ProcModify;
} MODULE_DATA;
```

This structure contains a module's complete description as stored by the serving application. In addition to the module's command description, it states the callback procedure for command execution and module-defined entity modification.

Contents **Element Description**

ID

[*MODULE_ID**IDX_MODULE_ID] Description of a module that has been loaded.

Handle

[*HINSTANCE*] Instance handle of that module. This handle can be used to find resources or procedures in that module.

ProcCommand

[*TOSOMODULECOMMAND_PROC*] Address of the TosoModuleCommand procedure of the module. This address can be used to call a module's command.

ProcModify

[*TOSOMODULEMODIFY_PROC*] Address of the TosoModuleModify procedure of the module. This address can be used to initiate the modification of a module-supplied entity. If *ProcModify* is NULL, this module cannot handle module-supplied entities.

ENUMDEF_DATA (Module Interface Types)

Syntax

```
typedef struct {
  int                   EnumData,
                        EnumCount,

                        EnumFlag1,
                        EnumFlag2,
                        EnumMode;

  BOOL                  EnumStopped;

  XPROPERTY             XProperty;

  FONTDEF               Font;
  double                CharDistance,
                        TabDistance,
                        LineDistance;
  int                   TextMode;

  DPOINT*               PointPtr;
  int*                  TypePtr;
  MATRIX*               MatrixPtr;
  LPSTR                 TextPtr,
                        NamePtr;

  UNIT_OBJECT_PTR       ObjPtr;
  UNIT_INSTANCE_PTR     InstPtr;
  UNIT_BLOCK_PTR        BlockPtr;
  UNIT_USER_PTR         UserPtr;
} ENUMDEF_DATA;
```

This structure contains a complete enumeration data description.

Element Description

EnumData

[*int*] This value indicates what type of data is coded in this structure. It can be one of the following values:

ENUMDATA_INVALID

The structure does not contain any valid data - return immediately. The value *EnumCount* can be ignored.

ENUMDATA_CURVE

The structure contains an open polyline or curve, described by points stored in *PointPtr* and point types in *TypePtr*. The value *EnumCount* states the number of points. On how a curve is coded in *PointPtr* and *TypePtr*, see the Comment section below. This type of data is resulting from an *EnumMode* containing at least one of the values ENUMMODE_LINES, ENUMMODE_POLYLINES, ENUMMODE_FILLS, ENUMMODE_ARCS or ENUMMODE_BEZIERS.

ENUMDATA_AREA

The structure contains a closed polyline or curve, described by points stored in *PointPtr* and point

types in *TypePtr*. The value *EnumCount* states the number of points. On how a curve is coded in *PointPtr* and *TypePtr*, see the Comment section below. This type of data is resulting from an *EnumMode* containing at least one of the values ENUMMODE_LINES, ENUMMODE_POLYLINES, ENUMMODE_FILLS, ENUMMODE_ARCS or ENUMMODE_BEZIERS.

ENUMDATA_MARK

The structure contains a list of markings in *PointPtr* and *TypePtr*. The content of *TypePtr* can be ignored, as it is DB_POINT_MARK for all points. The value *EnumCount* states the number of marks. This type of data is resulting from an *EnumMode* containing at least one of the values ENUMMODE_LINES, ENUMMODE_POLYLINES, ENUMMODE_FILLS, ENUMMODE_ARCS or ENUMMODE_BEZIERS.

ENUMDATA_CHAR

The structure contains a list of characters with according display matrices, stored in *TextPtr* and *MatrixPtr*. The font to be used is stored in *Font*. The value *EnumCount* states the number of characters. This type of data is resulting from an *EnumMode* containing ENUMMODE_SOLID_CHARS and/or ENUMMODE_OTHER_CHARS.

ENUMDATA_TEXT

The structure contains a null-terminated text string with an according display matrix, stored in *TextPtr* and *MatrixPtr*. The font to be used is stored in *Font*, further format information is stored in *CharDistance*, *TabDistance*, *LineDistance* and *TextMode*. The value *EnumCount* states the number of characters. This type of data is resulting from an *EnumMode* containing ENUMMODE_SOLID_TEXTS or ENUMMODE_OTHER_TEXTS.

ENUMDATA_INST

The structure contains an instance. The block name is stored in *NamePtr*, the library name in *TextPtr* and the display matrix in *MatrixPtr*. The value *EnumCount* can be ignored.This type of data is resulting from an *EnumMode* containing ENUMMODE_STRUCTURED.

ENUMDATA_USED_BLOCK

The structure contains the block name (in *NamePtr*) and library name (in *TextPtr*) of a referenced block. The value *EnumCount* can be ignored. This type of data is resulting from an *EnumMode* containing ENUMMODE_USED_BLOCKS.

ENUMDATA_USED_FONT

The structure contains a referenced font in *Font*. The value *EnumCount* can be ignored. This type of data is resulting from an *EnumMode* containing ENUMMODE_USED_FONTS.

ENUMDATA_NATIVE_OBJ

The structure contains only a native object's address in *ObjPtr*. The value *EnumCount* can be ignored. This type of data is resulting from an *EnumMode* containing ENUMMODE_NATIVE.

ENUMDATA_NATIVE_INST

The structure contains only a native instance's address in *InstPtr*. The value *EnumCount* can be ignored. This type of data is resulting from an *EnumMode* containing ENUMMODE_NATIVE.

ENUMDATA_NATIVE_BLOCK

The structure contains only a native block's address in *BlockPtr*. The value *EnumCount* can be ignored. This type of data is resulting from an *EnumMode* containing ENUMMODE_NATIVE.

ENUMDATA_NATIVE_USER

The structure contains only a native module-defined object's address in *UserPtr*. The value *EnumCount* can be ignored. This type of data is resulting from an *EnumMode* containing ENUMMODE_NATIVE.

EnumCount

[*int*] Number of elements currently available. What type of elements are currently available depends on the the value of *EnumData*. Be sure never to access more elements than stated here!

EnumFlag1

[*int*] First enumeration flag as passed to the enumeration procedure that produced this structure (see e.g. TosoEnumerateAll).

EnumFlag2

[*int*] Second enumeration flag as passed to the enumeration procedure that produced this structure (see e.g. TosoEnumerateAll).

EnumMode

[*int*] Enumeration mode as passed to the enumeration procedure that produced this structure (see e.g. TosoEnumerateAll).

EnumStopped

[*BOOL*] Indicates whether the enumeration has been stopped by a callback procedure returning FALSE.

XProperty

[*XPROPERTY*] Extended properties of the current enumeration data (valid for ENUMDATA_CURVE, ENUMDATA_AREA, ENUMDATA_MARK, ENUMDATA_CHAR, ENUMDATA_TEXT and ENUMDATA_INST).

Font

[*FONTDEF*] Font description of the current enumeration data (valid for ENUMDATA_CHAR, ENUMDATA_TEXT and ENUMDATA_USED_FONT).

CharDistance

[*double*] The value *CharDistance* determines the gap between two characters (valid for ENUMDATA_TEXT). This gap is stated relative to the font size. A value of 0.1 at a font size of 10pt will result in a character gap of 1pt. Allowed values are -10.0 to +10.0. The default value for TrueType and device fonts should be 0.0, for internal fonts 0.125.

TabDistance

[*double*] The value *TabDistance* determines the distance between two tabulators (valid for ENUMDATA_TEXT). This distance is stated relative to the font size. A value of 4.0 at a font size of 5 mm will result in a tabulator distance of 20 mm. Allowed values are -100.0 to 100.0. The default value is 4.0.

LineDistance

[*double*] The value *LineDistance* determines the offset between two lines of text, measured from baseline to baseline (valid for ENUMDATA_TEXT). This offset is stated relative to the font size. A value of 1.2 at a font size of 10pt will lead to a line offset of 12pt. Allowed values are -100.0 to 100.0. The default value is 1.0.

TextMode

[*int*] The value *TextMode* states the position of the text relative to the insertion point (valid for ENUMDATA_TEXT). It can be one of the following values:

0x0000	The insertion point defines the left end-point of the text's baseline, i.e. the text will be displayed left-aligned.
0x0001	The insertion point defines the center-point of the text's baseline, i.e. the text will be displayed centered.
0x0002	The insertion point defines the right end-point of the text's baseline, i.e. the text will be displayed right-aligned.

PointPtr

[*DPOINT**] Definition point coordinates of a curve or mark list (valid for ENUMDATA_CURVE, ENUMDATA_AREA and ENUMDATA_MARK).

TypePtr

[*int**] Definition point types of a curve or mark list (valid for ENUMDATA_CURVE,

ENUMDATA_AREA and ENUMDATA_MARK).

MatrixPtr

[*MATRIX**] Display matrix(es) (valid for ENUMDATA_CURVE, ENUMDATA_AREA, ENUMDATA_MARK, ENUMDATA_CHAR, ENUMDATA_TEXT and ENUMDATA_INST). Be sure to always apply this matrix if *EnumMode* includes the ENUMMODE_MATRIX flag!

TextPtr

[*LPSTR*] List of characters (valid for ENUMDATA_CHAR), address of a null-terminated string (valid for ENUMDATA_TEXT) or address of library name (valid for ENUMDATA_INST and ENUMDATA_USED_BLOCK).

NamePtr

[*LPSTR*] Block name (valid for ENUMDATA_INST and ENUMDATA_USED_BLOCK).

ObjPtr

[*UNIT_OBJECT_PTR*] Address of a native object (valid for ENUMDATA_NATIVE_OBJ).

InstPtr

[*UNIT_INSTANCE_PTR*] Address of a native instance (valid for ENUMDATA_NATIVE_INST).

BlockPtr

[*UNIT_BLOCK_PTR*] Address of a native block (valid for ENUMDATA_NATIVE_BLOCK).

UserPtr

[*UNIT_USER_PTR*] Address of a native module-defined object (valid for ENUMDATA_NATIVE_USER).

Contents

Comment

If *EnumData* is either ENUMDATA_CURVE or ENUMDATA_AREA, this structure describes a "poly-curve". Such a curve is the standard representation form of all output data created be the serving application. Its structure is based on the structure of the object 13 "Surface" as described in TVG 4.0 Documentation (TVG40.HLP). Following is a short description of this object, adapted to the different circumstances. Each point type in *TypePtr[]* is directly corresponding to the point stored in *PointPtr[]*.

A poly-curve is a collection of several curves, each defining one (open or closed) area. A curve starts with a start-point (type DB_POINT_START) followed by a sequence of curve elements. Three types of curve elements are available:

Line

A line is simply defined by stating its end-point (type DB_POINT_END). The line is drawn from the curve's current end-point to the line's end-point. The line's end-point then becomes the curve's current end-point.

Bézier curve

A Bézier curve is defined by stating two pivot points (type DB_POINT_PIVOT1 and DB_POINT_PIVOT2) and an end-point (type DB_POINT_END). The Bézier curve is drawn from the curve's current end-point (named 'S') to the Bézier curve's end-point (named 'E'), influenced by the two pivot points (named P1 and P2). The Bézier curve's end-point then becomes the curve's current end-point.

The points P of such a Bézier curve are calculated using the following equation:

$$P = (1-t)^3 \times S + 3t(1-t)^2 \times P1 + 3t^2(1-t) \times P2 + t^3 \times E \quad (0<=t<=1)$$

Circular arc

A circular arc is defined by its end-point (type DB_POINT_ARC) and its orientation and curvature (type DB_ALL_CURVE, *Curvature* in x-coordinate, *Orientation* in y-coordinate of the point). The arc is drawn from the curve's current end-point (named 'S') to the arc's end-point (named 'E'),

influenced by its orientation and curvature. The arc's end-point then becomes the curve's current end-point.
The value *Orientation* determines whether to draw the arc in clockwise direction (*Orientation* < 0) or in counter-clockwise direction (*Orientation* >= 0).
The value *Curvature* determines the radius of the arc in indirect manner. In can be handled in two ways. In geometrical view, the absolute value of *Curvature* is 1/(2 tan(ß/2)), where ß is the arc-angle of the circular arc. The sign of *Curvature* determines, which of the two possible arcs to use.

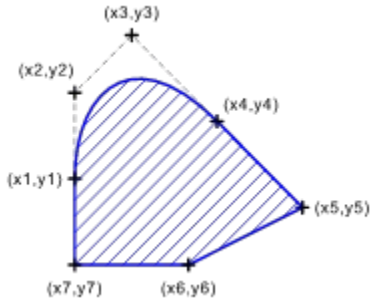
In practical use, this definition is not very handy. Instead, *Curvature* should be used to perform a simple vector calculation to obtain the arc's defining center-point M:

```
x(M) = 0.5 * ( x(S) + x(E) ) - Curvature * ( y(E) - y(S) )
y(M) = 0.5 * ( y(S) + y(E) ) + Curvature * ( x(E) - y(S) )
```

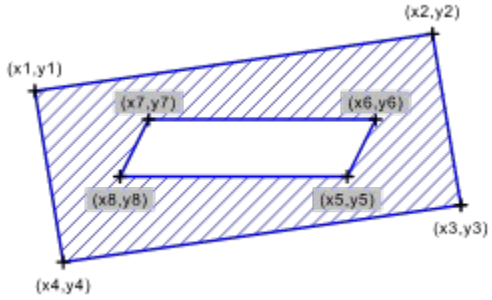
After calculating the center-point, all points required to draw a standard circular arc are known. The allowed value range of *Curvature* is ±1e100.

Each curve ends if either a new curve is started a new start-point (type DB_POINT_START) or the complete surface is ended. If *EnumData* is ENUMDATA_AREA, each curve is defined as closed, i.e. its start-point and end-point are to be connected by a line.

If a filled poly-curve consists of only one curve, simply the inside area of that curve will be filled:



If a filled poly-curve consists of multiple curves, they will be filled alternately, i.e. only areas overlapped by an *odd* number of curve's insides will be filled. If, e.g., a small round curve lies inside a large one, this small round curve will be transparent, as its inside in overlapped by two curves (*even* number!):



If a poly-curve is drawn with a non-solid line pattern, this line pattern has to be continued during each curve. A poly-curve may contain up to 100 nested curves.

GEO_OBJECT (Module Interface Types)

Contents **Syntax**

```
typedef struct {
  DPOINT          p1,
                  p2,
                  p3,
                  p4,
                  p5;

  int             Pen,
                  Layer;

  double          Len,
                  Start,
                  Arc;

  DRECT           Rect;

  int             Type,
                  Flag,
                  Count;

  BOOL            Orient,
                  Valid;
} GEO_OBJECT;
```

This structure contains the description of a basic geometric object.

Contents **Element Description**

p1,
p2,
p3,
p4,
p5

[*DPOINT*] These five points describe the object. The use of these points depends on the object's type stored in *Type*. See the Comment section below for further information.

Pen

[*int*] Zero-based pen index of the object. Do not modify!

Layer

[*int*] Zero-based layer index of the object. Do not modify!

Len

[*double*] Used for internal purposes. Do not modify!

Start

[*double*] Start-angle of an elliptical arc (not set for circular arcs!).

Arc

[*double*] Arc-angle of an circular or elliptical arc (signed, i.e. positive for counter-clockwise arc direction, negative for clockwise arc direction).

Rect

[*DRECT*] Used for internal purposes. Do not modify!

Type

[*int*] Type of the object stored in this structure. Possible values are:

OBJ_LINE
: Straight line.

OBJ_CIRCLE
: Circle.

OBJ_ARC
: Circular arc.

OBJ_SECTOR
: Circular sector.

OBJ_SEGMENT
: Circular segment.

OBJ_CURVE
: Bezier curve.

OBJ_ELLIPSE
: Ellipse.

OBJ_EARC
: Elliptical arc.

OBJ_ESECTOR
: Elliptical sector.

OBJ_ESEGMENT
: Elliptical segment.

OBJ_MARK
: Marking (optionally printed object), sometimes used to define a point.

OBJ_GEOLINE
: Geometry line (optionally printed object).

OBJ_GEOCIRCLE
: Geometry circle (optionally printed object).

OBJ_GEOELLIPSE
: Geometry ellipse (optionally printed object).

Flag
: [*int*] Used for internal purposes. Do not modify!

Count
: [*int*] Used for internal purposes. Do not modify!

Orient
: [*BOOL*] Orientation of an arc. TRUE for positive arc angle (i.e. counter-clockwise), FALSE for negative arc angle (i.e. clockwise).

Valid
: [*BOOL*] Used for internal purposes. Do not modify!

Contents **Comment**

The meaning of the five points depends on the object's types.

OBJ_MARK

p1 Point

OBJ_LINE
OBJ_GEOLINE
p1 Start-point
p2 End-point

OBJ_CIRCLE
OBJ_GEOCIRCLE
p1 Center-point
p2 Radius definition point

OBJ_ARC
OBJ_SECTOR
OBJ_SEGMENT
p1 Center-point
p2 Radius definition point
p4 Start-angle definition point
p5 End-angle definition point

OBJ_CURVE
p1 Start-point
p2 End-point
p4 First pivot point
p5 Second pivot point

OBJ_ELLIPSE
OBJ_GEOELLIPSE
p1 Center-point
p2 First vector definition point
p3 Second vector definition point

OBJ_EARC
OBJ_ESECTOR
OBJ_ESEGMENT
p1 Center-point
p2 First vector definition point
p3 Second vector definition point
p4 Start-angle definition point
p5 End-angle definition point

TVG 4.0 Constants (Constants)

Contents

Syntax and Description

```
#define TOSO_INTERFACE_VERSION  400
```
Current version of the interface at compiling time. Use this value to compare with the *InterfaceVersion* parameter of the TosoModuleInit procedure.

```
#define ENTRIES_PER_DRAWING     10000
```
Maximum number of blocks in a drawing.

```
#define ENTRIES_PER_LIBRARY     1000000
```
Maximum number of blocks in a library.

```
#define ENTRIES_PER_LEVEL       1000
```
Maximum number of blocks in each folder of a library.

```
#define ENTRIES_PER_FONT        224
```
Maximum number of characters in a font library.

```
#define ATTRIBS_PER_OBJECT      200
```
Maximum number of attributes per block or instance.

```
#define POINTS_PER_OBJECT       2000
```
Maximum number of point data blocks per object.

```
#define NAME_LENGTH_SHORT       32
```
Maximum length (in bytes) of a definition's short name (line pattern name, system name etc.).

```
#define NAME_LENGTH_LONG        64
```
Maximum length (in bytes) of a definition's long name (layer name, pen name etc.).

```
#define NAME_LENGTH_TITLE       64
```
Maximum length (in bytes) of a drawing or library title.

```
#define NAME_LENGTH_BLOCK       64
```
Maximum length (in bytes) of a block name (including folders).

```
#define NAME_LENGTH_FONT        64
```
Maximum length (in bytes) of a font name.

```
#define NAME_LENGTH_TEXTSHORT   250
```
Maximum length (in bytes) of a "short" text, as used for comments and dimensions.

```
#define NAME_LENGTH_TEXTLONG    8000
```
Maximum length (in bytes) of a "long" text, as used for standard and frame texts.

```
#define NAME_LENGTH_EDIT        250
```
Maximum length (in bytes) of an edit control's content.

```
#define DATA_PER_OBJECT         64000U
```
Maximum length (in bytes) of a unit's data section.

```
#define TVG_ID_LENGTH           22
```
Maximum length (in bytes) of TommySoftware® file identifications.

```
#define TVG_40_HEADER           "TommySoftware TVG 4.00"
```
File identification for TommySoftware® TVG 4.0 files.

```
#define TVG_30_HEADER           "TommySoftware TVG 3.00"
```
File identification for TommySoftware® TVG 3.0 files.

```
#define TVL_30_HEADER           "TommySoftware TVL 3.00"
```
File identification for TommySoftware® TVL 3.0 files.

```
#define TVG_BLOCK_ID            "*"
```
Library title of blocks.

```
#define TVG_BLOCK_INDEX         999
```
Library index of blocks.

```
#define TVG_TOOLBOX_MAX         14
```
Maximum number of buttons in the application's toolbox.

```
#define TVG_BLOCKLIST_MAX       100
```
Maximum number of buttons in the application's block list.

```
#define TVG_CUSTOMKEY_MAX       76
```

Maximum number of user-defined (custom) keys.

```
#define TVG_SYSTEMCOLOR_MAX     32
```

Maximum number of predefined system colors.

```
#define TVG_CUSTOMCOLOR_MAX     500
```

Maximum number of custom (user-defined) colors.

```
#define TVG_TOTALCOLOR_MAX      (TVG_SYSTEMCOLOR_MAX + TVG_CUSTOMCOLOR_MAX)
```

Maximum number of defined colors.

```
#define TVG_DEFAULT_MAX         8
```

Maximum number of default settings for pens and layers per drawing.

```
#define TVG_SYSTEM_MAX          50
```

Maximum number of coordinate systems per drawing (excluding "*Standard").

```
#define TVG_MULTILINE_MAX       50
```

Maximum number of ilne sequences per drawing (excluding "*Standard").

```
#define TVG_MULTILINEPART_MAX   8
```

Maximum number of lines per line sequence.

```
#define TVG_HATCH_MAX           100
```

Maximum number of hatch types per drawing (excluding "*Standard").

```
#define TVG_LINE_MAX            100
```

Maximum number of line patterns per drawing (excluding "*Standard").

```
#define TVG_LINEPART_MAX        16
```

Maximum number of line parts per line pattern.

```
#define TVG_PEN_MAX             500
```

Maximum number of pen definitions per drawing (excluding "*Standard").

```
#define TVG_LAYER_MAX           500
```

Maximum number of layer definitions per drawing (excluding "*Standard").

```
#define TVG_DRAWING_MAX         1
```

Maximum number of concurrently opened drawing files.

```
#define TVG_WINDOW_MAX          4
```

Maximum number of concurrent drawing windows.

```
#define TVG_WINDOW_VIEW         999
```

Window index of view window.

```
#define TVG_FILE_MAX            20
```

Maximum number of drawing files memorised.

```
#define TVG_FONT_MAX            10
```

Maximum number of fonts opened.

```
#define TVG_LIBRARY_MAX         50
```

Maximum number of libraries opened.

```
#define TVG_CACHE_MAX           10000
```

Maximum number of blocks in the cache.

```
#define TVG_BITMAP_MAX          50
```

Maximum number of bitmaps opened.

```
#define TVG_DEFPATH_MAX         8
```

Maximum number of standard paths per file type.

```
#define TVG_PLOTTER_MAX         8
```

Maximum number of plotter pens.

```
#define TVG_FILTER_MAX          20
```

Maximum number of filters loaded. This maximum value applies to import and export filters separately, i.e. the total number of filters is `2 * TVG_FILTER_MAX`.

```
#define TVG_MODULE_MAX          40
```

Maximum number of loaded modules.

```
#define TVG_MODULE_MENU_MAX     100
```

Maximum number of items in a module's menu.

```
#define TVG_MODULE_SUBMENU_MAX  20
```

Maximum number of items in a module's submenu.

```
#define TVG_INPUT_MAX           2000
```

Maximum number of pens to be entered per object input.

```
#define TVG_INPUT_DEF_MAX       8
```

Maximum number of predefined point entries per command.

```
#define TVG_POLYGON_MAX         16000
```

Maximum number of line in output polylines.

```
#define TVG_POLYPOLY_MAX        1000
```

Maximum number of nested polygons in a poly-polygon or curves in a hatching.

```
#define COORD_MIN               (-1.0e100)
```

Minimum coordinate allowed.

```
#define COORD_MAX               1.0e100
```

Maximium coordinate allowed.

```
#define EDIT_MIN                1.0e-10
```

Minimum value allowed for most entry types.

```
#define EDIT_MAX                1.0e10
```

Maximum value allowed for most entry types.

Structure Sizes (Constants)

Contents

Syntax and Description

```
#define PAGEDEF_SIZE            sizeof( PAGEDEF )
#define DEFAULTDEF_SIZE         sizeof( DEFAULTDEF )
#define WINDOWDEF_SIZE          sizeof( WINDOWDEF )
#define SYSTEMDEF_SIZE          sizeof( SYSTEMDEF )
#define LINEDEF_SIZE            sizeof( LINEDEF )
#define PROPERTY_SIZE           sizeof( PROPERTY )
#define PENDEF_SIZE             sizeof( PENDEF )
#define XPROPERTY_SIZE          sizeof( XPROPERTY )
#define LAYERDEF_SIZE           sizeof( LAYERDEF )
#define HATCHDEF_SIZE           sizeof( HATCHDEF )
#define COLORDEF_SIZE           sizeof( COLORDEF )
```

Structure sizes of drawing data types.

```
#define UNIT_ANY_SIZE           sizeof( UNIT_ANY )
#define UNIT_OBJECT_SIZE        sizeof( UNIT_OBJECT )
#define UNIT_BLOCK_SIZE         sizeof( UNIT_BLOCK )
#define UNIT_INSTANCE_SIZE      sizeof( UNIT_INSTANCE )
#define UNIT_USER_SIZE          sizeof( UNIT_USER )
```

Structure sizes of unit data types.

Color Definitions (Constants)

Contents

Syntax and Description

```
#define RGB_WHITE       RGB( 255, 255, 255 )
#define RGB_GRAY5       RGB( 242, 242, 242 )
#define RGB_GRAY12      RGB( 224, 224, 224 )
#define RGB_GRAY25      RGB( 192, 192, 192 )
#define RGB_GRAY50      RGB( 128, 128, 128 )
#define RGB_GRAY62      RGB(  96,  96,  96 )
#define RGB_GRAY75      RGB(  64,  64,  64 )
#define RGB_BLACK       RGB(   0,   0,   0 )
#define RGB_RED100      RGB( 255,   0,   0 )
#define RGB_RED75       RGB( 192,   0,   0 )
#define RGB_RED50       RGB( 128,   0,   0 )
#define RGB_RED25       RGB(  64,   0,   0 )
#define RGB_YELLOW100   RGB( 255, 255,   0 )
#define RGB_YELLOW75    RGB( 192, 192,   0 )
#define RGB_YELLOW50    RGB( 128, 128,   0 )
#define RGB_YELLOW25    RGB(  64,  64,   0 )
#define RGB_GREEN100    RGB(   0, 255,   0 )
#define RGB_GREEN75     RGB(   0, 192,   0 )
#define RGB_GREEN50     RGB(   0, 128,   0 )
#define RGB_GREEN25     RGB(   0,  64,   0 )
#define RGB_CYAN100     RGB(   0, 255, 255 )
#define RGB_CYAN75      RGB(   0, 192, 192 )
#define RGB_CYAN50      RGB(   0, 128, 128 )
#define RGB_CYAN25      RGB(   0,  64,  64 )
#define RGB_BLUE100     RGB(   0,   0, 255 )
#define RGB_BLUE75      RGB(   0,   0, 192 )
#define RGB_BLUE50      RGB(   0,   0, 128 )
#define RGB_BLUE25      RGB(   0,   0,  64 )
#define RGB_MAGENTA100  RGB( 255,   0, 255 )
#define RGB_MAGENTA75   RGB( 192,   0, 192 )
#define RGB_MAGENTA50   RGB( 128,   0, 128 )
#define RGB_MAGENTA25   RGB(  64,   0,  64 )
```

RGB values of the 32 standard colors.

```
#define RGB_BROWN       RGB( 128,  64,   0 )
#define RGB_ORANGE      RGB( 255, 128,   0 )
```

RGB values of additional colors.

Mathematical Constants (Constants)

Contents

Syntax and Description

```
#define REAL_SIN_15      0.2588190451025
#define REAL_COS_15      0.9659258262891
#define REAL_TAN_15      0.2679491924311
#define REAL_SIN_7       0.1218693434051
#define REAL_COS_7       0.9925461516413
#define REAL_TAN_7       0.1227845609029
#define REAL_SIN_30      0.5
#define REAL_COS_30      0.8660254037844
#define REAL_TAN_30      0.5773502691896
#define REAL_SIN_42      0.6626200482157
#define REAL_COS_42      0.7489555720789
#define REAL_TAN_42      0.8847252645559
```

Trigonometrical constants used for isometric and dimetric display.

```
#define REAL_01PI        0.3141592653589793
#define REAL_05PI        1.5707963267948965
#define REAL_PI          3.141592653589793
#define REAL_15PI        4.7123889803846895
#define REAL_2PI         6.283185307179586
#define REAL_25PI        7.8539816339744825
#define REAL_3PI         9.424777960769379
#define REAL_35PI       10.9955742875642755
#define REAL_4PI        12.566370614359172
```

Trigonometrical constants used for angle representation and calculation.

```
#define REAL_DEG_RAD     0.01745329251994
#define REAL_GRA_RAD     0.01570796326795
#define REAL_REL_RAD     6.283185307179586
#define REAL_RAD_DEG    57.29577951308
#define REAL_RAD_GRA    63.66197723676
#define REAL_RAD_REL     0.1591549430919
```

Trigonometrical constants used to convert different types of angle representation into others.

```
#define REAL_INCH_MM    25.4
#define REAL_MM_INCH     0.03937007874016
```

Constants used to convert mm values to inch values and vice versa.

```
#define REAL_ROOT_3      1.732050807569
#define REAL_ROOT_2      1.414213562373
#define REAL_ROOT_05     0.7071067811865
```

Constants for geometrical calculations.

"Invalid" Values (Constants)

Syntax and Description

```
#define REAL_NOVAL      1.0e+300
#define FLOAT_NOVAL     1.0e+32
#define INT_NOVAL       (-2147483647)
#define LONG_NOVAL      (-2147483647)
```

These values are used to indicate "invalid" values of the corresponding data types. Do only use these values if indicated in a procedure's description.

```
#define NOPARAM         0
```

Use this value instead of zero to indicate that the corresponding parameter is not used (like NULL for handles and pointers).

Command IDs (Constants)

The command identifiers listed in this section are usually used when calling TosoCommandInternal in order to execute a command of the serving application. They may also be used to determine a command's title (using TosoGetCommandTitle) or icon (using TosoGetCommandIcon).

Contents **Syntax**

```
#define IDM_FIRST_ENTRY                 100

#define IDM_FILE_FIRST                  100
#define IDM_FILE_NEW                            100
#define IDM_FILE_OPEN                           101
#define IDM_FILE_SAVE                           102
#define IDM_FILE_SAVE_AS                        103
#define IDM_FILE_LIST                           104
#define IDM_FILE_IMPORT_SUB                     105
#define IDM_FILE_IMPORT_T4G                     106
#define IDM_FILE_IMPORT_EMF                     107
#define IDM_FILE_IMPORT_WMF                     108
#define IDM_FILE_IMPORT_BMP                     109
#define IDM_FILE_IMPORT_BMP_UPDATE              110
#define IDM_FILE_EXPORT_SUB                     111
#define IDM_FILE_EXPORT_T4G                     112
#define IDM_FILE_EXPORT_EMF                     113
#define IDM_FILE_EXPORT_WMF                     114
#define IDM_FILE_EXPORT_BMP                     115
#define IDM_FILE_EXPORT_BMP_AREA                116
#define IDM_FILE_EXTRA_SUB                      117
#define IDM_FILE_EXTRA_ERASE                    118
#define IDM_FILE_EXTRA_SAVE_DUP                 119
#define IDM_FILE_EXTRA_LOAD_DUP                 120
#define IDM_FILE_EXTRA_AUTOSAVE                 121
#define IDM_FILE_PRINT                          122
#define IDM_FILE_PRINT_AREA                     123
#define IDM_FILE_EXIT                           124
#define IDM_FILE_LAST                   125

#define IDM_EDIT_FIRST                  200
#define IDM_EDIT_UNDO                           200
#define IDM_EDIT_REDO                           201
#define IDM_EDIT_UNDO_ERASE                     202
#define IDM_EDIT_CUT                            203
#define IDM_EDIT_COPY                           204
#define IDM_EDIT_PASTE                          205
#define IDM_EDIT_COMMENT_SUB                    206
#define IDM_EDIT_COMMENT                        207
#define IDM_EDIT_COMMENT_PARAM                  208
#define IDM_EDIT_TVG_DETAILS                    209
#define IDM_EDIT_QUEUE                          210
#define IDM_EDIT_LAST                   211

#define IDM_CONFIG_FIRST                300
#define IDM_CONFIG_PAGE_FORMAT                  300
#define IDM_CONFIG_LAYER_SUB                    301
#define IDM_CONFIG_LAYER_LIST                   302
#define IDM_CONFIG_LAYER_EDIT                   303
#define IDM_CONFIG_LAYER_DEFAULT                304
```

```
#define IDM_CONFIG_LAYER_RESET                  305
#define IDM_CONFIG_XLAYER_SUB                   306
#define IDM_CONFIG_XLAYER_SELECT                307
#define IDM_CONFIG_XLAYER_EDIT                  308
#define IDM_CONFIG_XLAYER_ASSIGN                309
#define IDM_CONFIG_XLAYER_DISABLE               310
#define IDM_CONFIG_XLAYER_ENABLE                311
#define IDM_CONFIG_XLAYER_FREEZE                312
#define IDM_CONFIG_XLAYER_MELT                  313
#define IDM_CONFIG_XLAYER_IGNORE                314
#define IDM_CONFIG_XLAYER_USE                   315
#define IDM_CONFIG_XLAYER_SHADE                 316
#define IDM_CONFIG_XLAYER_LIGHT                 317
#define IDM_CONFIG_XLAYER_HIDE                  318
#define IDM_CONFIG_XLAYER_CONCENTRATE           319
#define IDM_CONFIG_XLAYER_UNDO                  320
#define IDM_CONFIG_SYSTEM_SUB                   321
#define IDM_CONFIG_SYSTEM_LIST                  322
#define IDM_CONFIG_SYSTEM_EDIT                  323
#define IDM_CONFIG_SYSTEM_ORIGIN                324
#define IDM_CONFIG_SYSTEM_SHOW_DISPLAY          325
#define IDM_CONFIG_SYSTEM_EDIT_DISPLAY          326
#define IDM_CONFIG_SYSTEM_SHOW_POSITION         327
#define IDM_CONFIG_SYSTEM_EDIT_POSITION         328
#define IDM_CONFIG_PEN_SUB                      329
#define IDM_CONFIG_PEN_LIST                     330
#define IDM_CONFIG_PEN_EDIT                     331
#define IDM_CONFIG_PEN_DEFAULT                  332
#define IDM_CONFIG_PEN_LINE                     333
#define IDM_CONFIG_ZOOM_SUB                     334
#define IDM_CONFIG_ZOOM_AREA                    335
#define IDM_CONFIG_ZOOM_OBJECTS                 336
#define IDM_CONFIG_ZOOM_PAGE                    337
#define IDM_CONFIG_ZOOM_ORIGINAL                338
#define IDM_CONFIG_ZOOM_FACTOR                  339
#define IDM_CONFIG_ZOOM_UNDO                    340
#define IDM_CONFIG_ZOOM_PAN                     341
#define IDM_CONFIG_ZOOM_PLUS                    342
#define IDM_CONFIG_ZOOM_MINUS                   343
#define IDM_CONFIG_SAVE_DEFAULT                 344
#define IDM_CONFIG_WINDOW_SUB                   345
#define IDM_CONFIG_WINDOW_ARRANGE               346
#define IDM_CONFIG_WINDOW2                      347
#define IDM_CONFIG_WINDOW3                      348
#define IDM_CONFIG_WINDOW4                      349
#define IDM_CONFIG_WINDOW_VIEW                  350
#define IDM_CONFIG_WINDOW_PANEL                 351
#define IDM_CONFIG_WINDOW_PEN                   352
#define IDM_CONFIG_WINDOW_LAYER                 353
#define IDM_CONFIG_WINDOW_STATUS                354
#define IDM_CONFIG_WINDOW_TOOLBOX               355
#define IDM_CONFIG_WINDOW_PROPERTY              356
#define IDM_CONFIG_WINDOW_BLOCK                 357
#define IDM_CONFIG_WINDOW_GUIDE                 358
#define IDM_CONFIG_PROGSTATUS                   359
#define IDM_CONFIG_PROFILE_SUB                  360
#define IDM_CONFIG_PROFILE_WINDOW               361
#define IDM_CONFIG_PROFILE_GENERAL              362
#define IDM_CONFIG_PROFILE_PATHS                363
#define IDM_CONFIG_PROFILE_LOADSAVE             364
```

```
#define IDM_CONFIG_TVI_SUB                      365
#define IDM_CONFIG_TVI_SCREEN                   366
#define IDM_CONFIG_TVI_OUTPUT                   367
#define IDM_CONFIG_TVI_PLOTTER                  368
#define IDM_CONFIG_TVI_ACCURACY                 369
#define IDM_CONFIG_TVI_COLORS                   370
#define IDM_CONFIG_TVI_TIME                     371
#define IDM_CONFIG_TVI_BLOCK                    372
#define IDM_CONFIG_TVI_KEY_CHANGE               373
#define IDM_CONFIG_TVI_KEY_DISPLAY              374
#define IDM_CONFIG_TVI_MOUSE                    375
#define IDM_CONFIG_TVI_LOAD                     376
#define IDM_CONFIG_TVI_SAVE_AS                  377
#define IDM_CONFIG_TVI_DETAILS                  378
#define IDM_CONFIG_TVI_SAVE_DEFAULT             379
#define IDM_CONFIG_TVI_SAVEONEXIT               380
#define IDM_CONFIG_LAST                 381

#define IDM_SHAPE_FIRST                 400
#define IDM_SHAPE_EDIT_PARAM                    400
#define IDM_SHAPE_EDIT_TEXT                     401
#define IDM_SHAPE_ERASE                         402
#define IDM_SHAPE_MOVE_SUB                      403
#define IDM_SHAPE_MOVE_STANDARD                 404
#define IDM_SHAPE_MOVE_PERP                     405
#define IDM_SHAPE_MOVE_PARALLEL                 406
#define IDM_SHAPE_MOVE_RELATIVE                 407
#define IDM_SHAPE_SCALE_SUB                     408
#define IDM_SHAPE_SCALE_FACTOR                  409
#define IDM_SHAPE_SCALE_PROP                    410
#define IDM_SHAPE_SCALE_ANY                     411
#define IDM_SHAPE_SCALE_REFERENCE               412
#define IDM_SHAPE_ROTATE_SUB                    413
#define IDM_SHAPE_ROTATE_CENTER                 414
#define IDM_SHAPE_ROTATE_ANY                    415
#define IDM_SHAPE_ROTATE_REFERENCE              416
#define IDM_SHAPE_REFLECT_SUB                   417
#define IDM_SHAPE_REFLECT_H                     418
#define IDM_SHAPE_REFLECT_V                     419
#define IDM_SHAPE_REFLECT_LINE                  420
#define IDM_SHAPE_DISTORT_SUB                   421
#define IDM_SHAPE_DISTORT_H                     422
#define IDM_SHAPE_DISTORT_V                     423
#define IDM_SHAPE_CENTER_SUB                    424
#define IDM_SHAPE_CENTER_PAGE_H                 425
#define IDM_SHAPE_CENTER_PAGE_V                 426
#define IDM_SHAPE_CENTER_PAGE_BOTH              427
#define IDM_SHAPE_CENTER_RECT_H                 428
#define IDM_SHAPE_CENTER_RECT_V                 429
#define IDM_SHAPE_CENTER_RECT_BOTH              430
#define IDM_SHAPE_PMOVE_SINGLE                  431
#define IDM_SHAPE_PMOVE_SUB                     432
#define IDM_SHAPE_PMOVE_STANDARD                433
#define IDM_SHAPE_PMOVE_PERP                    434
#define IDM_SHAPE_PMOVE_PARALLEL                435
#define IDM_SHAPE_PMOVE_RELATIVE                436
#define IDM_SHAPE_GROUP_SUB                     437
#define IDM_SHAPE_GROUP_LINK                    438
#define IDM_SHAPE_GROUP_UNLINK                  439
#define IDM_SHAPE_ORDER_SUB                     440
```

```
#define IDM_SHAPE_ORDER_BACK                    441
#define IDM_SHAPE_ORDER_BEHIND                  442
#define IDM_SHAPE_ORDER_BEFORE                  443
#define IDM_SHAPE_ORDER_FRONT                   444
#define IDM_SHAPE_LAST                  445

#define IDM_DRAW_FIRST                  500
#define IDM_DRAW_LINE_SUB                       500
#define IDM_DRAW_LINE_STANDARD                  501
#define IDM_DRAW_LINE_LINE                      502
#define IDM_DRAW_LINE_H                         503
#define IDM_DRAW_LINE_V                         504
#define IDM_DRAW_LINE_MIDPERP                   505
#define IDM_DRAW_LINE_PERP                      506
#define IDM_DRAW_LINE_PARALLEL                  507
#define IDM_DRAW_LINE_PARALLEL_NUM              508
#define IDM_DRAW_LINE_ANGLE                     509
#define IDM_DRAW_LINE_BISECTOR                  510
#define IDM_DRAW_LINE_CROSS                     511
#define IDM_DRAW_LINE_POLY                      512
#define IDM_DRAW_LINE_DISTANT                   513
#define IDM_DRAW_LINE_ZIGZAG                    514
#define IDM_DRAW_TANGENT_SUB                    515
#define IDM_DRAW_TANGENT_OP                     516
#define IDM_DRAW_TANGENT_OPH                    517
#define IDM_DRAW_TANGENT_OPV                    518
#define IDM_DRAW_TANGENT_OPE                    519
#define IDM_DRAW_TANGENT_OAE                    520
#define IDM_DRAW_TANGENT_OO                     521
#define IDM_DRAW_POLYGON_SUB                    522
#define IDM_DRAW_POLYGON_TRIANGLE               523
#define IDM_DRAW_POLYGON_QUADRANGLE             524
#define IDM_DRAW_POLYGON_PARALLEL               525
#define IDM_DRAW_POLYGON_RECTANGLE              526
#define IDM_DRAW_POLYGON_ANY                    527
#define IDM_DRAW_POLYEDER_SUB                   528
#define IDM_DRAW_POLYEDER_STANDARD              529
#define IDM_DRAW_POLYEDER_CIRCLE                530
#define IDM_DRAW_POLYEDER_CIRCUMCIRCLE          531
#define IDM_DRAW_POLYEDER_DIAMETER              532
#define IDM_DRAW_POLYEDER_SIDELENGTH            533
#define IDM_DRAW_POLYEDER_INNERRADIUS           534
#define IDM_DRAW_CIRCLE_SUB                     535
#define IDM_DRAW_CIRCLE_STANDARD                536
#define IDM_DRAW_CIRCLE_CIRCLE                  537
#define IDM_DRAW_CIRCLE_CIRCUMCIRCLE            538
#define IDM_DRAW_CIRCLE_DIAMETER                539
#define IDM_DRAW_CIRCLE_INCIRCLE                540
#define IDM_DRAW_CIRCLE_CONC                    541
#define IDM_DRAW_CIRCLE_PERP                    542
#define IDM_DRAW_CIRCLE_LLL                     543
#define IDM_DRAW_CIRCLE_PPR                     544
#define IDM_DRAW_CIRCLE_OPR                     545
#define IDM_DRAW_CIRCLE_OOR                     546
#define IDM_DRAW_ARC_SUB                        547
#define IDM_DRAW_ARC_STANDARD                   548
#define IDM_DRAW_ARC_CIRCLE                     549
#define IDM_DRAW_ARC_CIRCUMCIRCLE               550
#define IDM_DRAW_ARC_DIAMETER                   551
#define IDM_DRAW_ARC_CONC_CIRCLE                552
```

```
#define IDM_DRAW_ARC_CONC_ARC                   553
#define IDM_DRAW_ARC_PPR                        554
#define IDM_DRAW_ARC_OPR                        555
#define IDM_DRAW_ARC_OOR                        556
#define IDM_DRAW_ELLIPSE_SUB                    557
#define IDM_DRAW_ELLIPSE_STANDARD               558
#define IDM_DRAW_ELLIPSE_ROTATED                559
#define IDM_DRAW_ELLIPSE_ANY                    560
#define IDM_DRAW_ELLIPSE_ELLIPSE                561
#define IDM_DRAW_EARC_SUB                       562
#define IDM_DRAW_EARC_STANDARD                  563
#define IDM_DRAW_EARC_ANY                       564
#define IDM_DRAW_EARC_ELLIPSE                   565
#define IDM_DRAW_ARCMODE_SUB                    566
#define IDM_DRAW_ARCMODE_ARC                    567
#define IDM_DRAW_ARCMODE_SECTOR                 568
#define IDM_DRAW_ARCMODE_SEGMENT                569
#define IDM_DRAW_ARCMODE_TOGGLE                 570
#define IDM_DRAW_ARCMODE_DIRECTION              571
#define IDM_DRAW_FREEHAND                       572
#define IDM_DRAW_SPLINE                         573
#define IDM_DRAW_CURVE                          574
#define IDM_DRAW_SURFACE                        575
#define IDM_DRAW_HATCH_SUB                      576
#define IDM_DRAW_HATCH_OBJ                      577
#define IDM_DRAW_HATCH_SURFACE                  578
#define IDM_DRAW_HATCH_TYPE_LIST                579
#define IDM_DRAW_HATCH_TYPE_EDIT                580
#define IDM_DRAW_HATCH_TYPE_LINE                581
#define IDM_DRAW_HATCH_ORIGIN                   582
#define IDM_DRAW_LAST                   583

#define IDM_GEO_FIRST                   600
#define IDM_GEO_LINE_SUB                        600
#define IDM_GEO_LINE_STANDARD                   601
#define IDM_GEO_LINE_LINE                       602
#define IDM_GEO_LINE_H                          603
#define IDM_GEO_LINE_V                          604
#define IDM_GEO_LINE_MIDPERP                    605
#define IDM_GEO_LINE_PERP                       606
#define IDM_GEO_LINE_PARALLEL                   607
#define IDM_GEO_LINE_PARALLEL_NUM               608
#define IDM_GEO_LINE_ANGLE                      609
#define IDM_GEO_LINE_BISECTOR                   610
#define IDM_GEO_LINE_CROSS                      611
#define IDM_GEO_TANGENT_SUB                     612
#define IDM_GEO_TANGENT_OP                      613
#define IDM_GEO_TANGENT_H                       614
#define IDM_GEO_TANGENT_V                       615
#define IDM_GEO_TANGENT_OA                      616
#define IDM_GEO_TANGENT_OO                      617
#define IDM_GEO_CIRCLE_SUB                      618
#define IDM_GEO_CIRCLE_STANDARD                 619
#define IDM_GEO_CIRCLE_CIRCLE                   620
#define IDM_GEO_CIRCLE_CIRCUMCIRCLE             621
#define IDM_GEO_CIRCLE_DIAMETER                 622
#define IDM_GEO_CIRCLE_INCIRCLE                 623
#define IDM_GEO_CIRCLE_CONC                     624
#define IDM_GEO_CIRCLE_PERP                     625
#define IDM_GEO_CIRCLE_LLL                      626
```

```
#define IDM_GEO_CIRCLE_PPR                  627
#define IDM_GEO_CIRCLE_OPR                  628
#define IDM_GEO_CIRCLE_OOR                  629
#define IDM_GEO_ELLIPSE_SUB                 630
#define IDM_GEO_ELLIPSE_STANDARD            631
#define IDM_GEO_ELLIPSE_ROTATED             632
#define IDM_GEO_ELLIPSE_ANY                 633
#define IDM_GEO_ELLIPSE_ELLIPSE             634
#define IDM_GEO_DISPLAY                     635
#define IDM_GEO_FREEZE                      636
#define IDM_GEO_MARK                        637
#define IDM_GEO_DIVIDE_SUB                  638
#define IDM_GEO_DIVIDE_DISTANCE             639
#define IDM_GEO_DIVIDE_OBJECT               640
#define IDM_GEO_DIVIDE_ARRAY                641
#define IDM_GEO_COPY_SUB                    642
#define IDM_GEO_COPY_MARK                   643
#define IDM_GEO_COPY_ROTATE                 644
#define IDM_GEO_COPY_STEP                   645
#define IDM_GEO_COPY_DISTANCE               646
#define IDM_GEO_COPY_OBJECT                 647
#define IDM_GEO_COPY_ARRAY                  648
#define IDM_GEO_LAST                649

#define IDM_TRIM_FIRST              700
#define IDM_TRIM_TRIM_SUB                   700
#define IDM_TRIM_TRIM_CUTOUT                701
#define IDM_TRIM_TRIM_SPLIT                 702
#define IDM_TRIM_TRIM_RESOLVE               703
#define IDM_TRIM_TRIM_LENGTH_P              704
#define IDM_TRIM_TRIM_LENGTH_OBJ            705
#define IDM_TRIM_TRIM_ANGLE_P               706
#define IDM_TRIM_TRIM_ANGLE_OBJ             707
#define IDM_TRIM_CURVE_SUB                  708
#define IDM_TRIM_CURVE_EDIT                 709
#define IDM_TRIM_CURVE_EXTENT               710
#define IDM_TRIM_CURVE_CUT                  711
#define IDM_TRIM_SURFACE_SUB                712
#define IDM_TRIM_SURFACE_GENERATE           713
#define IDM_TRIM_SURFACE_OR                 714
#define IDM_TRIM_SURFACE_AND                715
#define IDM_TRIM_SURFACE_MINUS              716
#define IDM_TRIM_SURFACE_COMBINE            717
#define IDM_TRIM_SURFACE_CUT                718
#define IDM_TRIM_CORNER                     719
#define IDM_TRIM_CHAMFER_SUB                720
#define IDM_TRIM_CHAMFER_OO                 721
#define IDM_TRIM_CHAMFER_CORNER             722
#define IDM_TRIM_CHAMFER_ALL                723
#define IDM_TRIM_ROUNDOUT_SUB               724
#define IDM_TRIM_ROUNDOUT_OO                725
#define IDM_TRIM_ROUNDOUT_CORNER            726
#define IDM_TRIM_ROUNDOUT_ALL               727
#define IDM_TRIM_ROUNDIN_SUB                728
#define IDM_TRIM_ROUNDIN_OO                 729
#define IDM_TRIM_ROUNDIN_CORNER             730
#define IDM_TRIM_ROUNDIN_ALL                731
#define IDM_TRIM_TRANSFORM_SUB              732
#define IDM_TRIM_TRANSFORM_INVERT           733
#define IDM_TRIM_TRANSFORM_ZIGZAG           734
```

```
#define IDM_TRIM_TRANSFORM_CIRCLE               735
#define IDM_TRIM_TRANSFORM_ARC                  736
#define IDM_TRIM_TRANSFORM_SECTOR               737
#define IDM_TRIM_TRANSFORM_SEGMENT              738
#define IDM_TRIM_TRANSFORM_ELLIPSE              739
#define IDM_TRIM_TRANSFORM_EARC                 740
#define IDM_TRIM_TRANSFORM_ESECTOR              741
#define IDM_TRIM_TRANSFORM_ESEGMENT             742
#define IDM_TRIM_TRANSFORM_CURVE                743
#define IDM_TRIM_TRANSFORM_SURFACE              744
#define IDM_TRIM_CLIP_SUB                       745
#define IDM_TRIM_CLIP_CREATE_INSIDE             746
#define IDM_TRIM_CLIP_CREATE_OUTSIDE            747
#define IDM_TRIM_CLIP_RESOLVE                   748
#define IDM_TRIM_LAST                   749

#define IDM_DIM_FIRST                   800
#define IDM_DIM_TEXT_SUB                        800
#define IDM_DIM_TEXT_STANDARD                   801
#define IDM_DIM_TEXT_FRAME                      802
#define IDM_DIM_TEXT_REFERENCE                  803
#define IDM_DIM_TEXT_FLAT                       804
#define IDM_DIM_LINE_SUB                        805
#define IDM_DIM_LINE_LINE                       806
#define IDM_DIM_LINE_ARC                        807
#define IDM_DIM_SUB1                            808
#define IDM_DIM_LENGTH_P                        809
#define IDM_DIM_LENGTH_OBJ                      810
#define IDM_DIM_DISTANCE_P                      811
#define IDM_DIM_DISTANCE_OBJ                    812
#define IDM_DIM_RADIUS_P                        813
#define IDM_DIM_RADIUS_OBJ                      814
#define IDM_DIM_DIAMETER_P                      815
#define IDM_DIM_DIAMETER_OBJ                    816
#define IDM_DIM_ANGLE_P                         817
#define IDM_DIM_ANGLE_LL                        818
#define IDM_DIM_ANGLE_OBJ                       819
#define IDM_DIM_ARCLENGTH_P                     820
#define IDM_DIM_ARCLENGTH_OBJ                   821
#define IDM_DIM_COORDINATE                      822
#define IDM_DIM_SUB2                            823
#define IDM_DIM_AREA                            824
#define IDM_DIM_PERIMETER                       825
#define IDM_DIM_PARAMETER                       826
#define IDM_DIM_EDIT_LINE_SUB                   827
#define IDM_DIM_EDIT_LINE_ANGLE                 828
#define IDM_DIM_EDIT_LINE_POSITION              829
#define IDM_DIM_EDIT_NUM_SUB                    830
#define IDM_DIM_EDIT_NUM_ANGLE                  831
#define IDM_DIM_EDIT_NUM_POSITION               832
#define IDM_DIM_EDIT_NUM_UPDATE                 833
#define IDM_DIM_FONT                            834
#define IDM_DIM_LAST                    835

#define IDM_LIBRARY_FIRST               900
#define IDM_LIBRARY_BLOCK_SUB                   900
#define IDM_LIBRARY_BLOCK_INSERT                901
#define IDM_LIBRARY_BLOCK_READ                  902
#define IDM_LIBRARY_BLOCK_READ_FRAME            903
#define IDM_LIBRARY_BLOCK_EDIT                  904
```

```
#define IDM_LIBRARY_POSNO_SUB                   905
#define IDM_LIBRARY_POSNO_ASSIGN                906
#define IDM_LIBRARY_POSNO_DELETE                907
#define IDM_LIBRARY_POSNO_REORDER               908
#define IDM_LIBRARY_LIST                        909
#define IDM_LIBRARY_CLEAN                       910
#define IDM_LIBRARY_CONVERT                     911
#define IDM_LIBRARY_SPLIT                       912
#define IDM_LIBRARY_REPLACE_BLOCK               913
#define IDM_LIBRARY_REPLACE_LIBRARY             914
#define IDM_LIBRARY_TVL                         915
#define IDM_LIBRARY_LAST                916

#define IDM_EXTRA_FIRST                 1000
#define IDM_EXTRA_KEY_F10                       1000
#define IDM_EXTRA_KEY_F11                       1001
#define IDM_EXTRA_KEY_F12                       1002
#define IDM_EXTRA_SEL_SUB                       1003
#define IDM_EXTRA_SEL_SET                       1004
#define IDM_EXTRA_SEL_CLEAR                     1005
#define IDM_EXTRA_SEL_INVERT                    1006
#define IDM_EXTRA_KEY_F6_SHIFT                  1007
#define IDM_EXTRA_KEY_F8                        1008
#define IDM_EXTRA_KEY_ESC                       1009
#define IDM_EXTRA_KEY_ESC_SHIFT                 1010
#define IDM_EXTRA_KEY_F7                        1011
#define IDM_EXTRA_KEY_F5                        1012
#define IDM_EXTRA_KEY_F5_SHIFT                  1013
#define IDM_EXTRA_SNAP_SUB                      1014
#define IDM_EXTRA_SNAP_GLOBAL                   1015
#define IDM_EXTRA_SNAP_CENTER                   1016
#define IDM_EXTRA_SNAP_QUADRANT                 1017
#define IDM_EXTRA_SNAP_EDGE                     1018
#define IDM_EXTRA_SNAP_CORNER                   1019
#define IDM_EXTRA_SNAP_INTERSECTION             1020
#define IDM_EXTRA_SNAP_GEO                      1021
#define IDM_EXTRA_SNAP_MARK                     1022
#define IDM_EXTRA_SNAP_OTHERS                   1023
#define IDM_EXTRA_SNAP_RELATIVE                 1024
#define IDM_EXTRA_SNAP_RADIUS                   1025
#define IDM_EXTRA_KEY_SUB                       1026
#define IDM_EXTRA_KEY_SPACE                     1027
#define IDM_EXTRA_KEY_LBUTTON                   1028
#define IDM_EXTRA_KEY_MBUTTON                   1029
#define IDM_EXTRA_KEY_RBUTTON                   1030
#define IDM_EXTRA_KEY_ALT                       1031
#define IDM_EXTRA_KEY_LEFT                      1032
#define IDM_EXTRA_KEY_RIGHT                     1033
#define IDM_EXTRA_KEY_UP                        1034
#define IDM_EXTRA_KEY_DOWN                      1035
#define IDM_EXTRA_KEY_HOME                      1036
#define IDM_EXTRA_KEY_END                       1037
#define IDM_EXTRA_KEY_PGUP                      1038
#define IDM_EXTRA_KEY_PGDN                      1039
#define IDM_EXTRA_KEY_EDIT_STEP                 1040
#define IDM_EXTRA_LAST                  1041

#define IDM_HELP_FIRST                  1100
#define IDM_HELP_INDEX                          1100
#define IDM_HELP_COMMAND                        1101
```

```
#define IDM_HELP_CLICK                          1102
#define IDM_HELP_ON_HELP                        1103
#define IDM_HELP_REGISTER                       1104
#define IDM_HELP_QA                             1105
#define IDM_HELP_TOSONEWS                       1106
#define IDM_HELP_PUZZLE                         1107
#define IDM_HELP_ABOUT                          1108
#define IDM_HELP_LAST                   1109

#define IDM_TABLET_FIRST                1200
#define IDM_TABLET_SUB                          1200
#define IDM_TABLET_BUTTON                       1201
#define IDM_TABLET_CALIBRATE                    1202
#define IDM_TABLET_ORIGIN                       1203
#define IDM_TABLET_LOAD                         1204
#define IDM_TABLET_SAVE_AS                      1205
#define IDM_TABLET_INFO                         1206
#define IDM_TABLET_OPTIONS                      1207
#define IDM_TABLET_EDIT                         1208
#define IDM_TABLET_DELETE                       1209
#define IDM_TABLET_LAST                 1210

#define IDM_EXTERN_FIRST        1300
#define IDM_EXTERN_MODULE_MIN           IDM_EXTERN_FIRST
#define IDM_EXTERN_MODULE_MAX           ( IDM_EXTERN_MODULE_MIN + TVG_MODULE_MAX *
TVG_MODULE_MENU_MAX )
#define IDM_EXTERN_IMPORT_MIN           IDM_EXTERN_MODULE_MAX
#define IDM_EXTERN_IMPORT_MAX           ( IDM_EXTERN_MODULE_MAX + TVG_FILTER_MAX )
#define IDM_EXTERN_EXPORT_MIN           IDM_EXTERN_IMPORT_MAX
#define IDM_EXTERN_EXPORT_MAX           ( IDM_EXTERN_EXPORT_MIN + TVG_FILTER_MAX )
#define IDM_EXTERN_LAST                 IDM_EXTERN_EXPORT_MAX

#define IDM_LAST_ENTRY          IDM_EXTERN_LAST

#define IDM_INTERN_FIRST        IDM_LAST_ENTRY

#define IDM_INTERN_BLOCK_INSERT_MIN    IDM_INTERN_FIRST
#define IDM_INTERN_BLOCK_INSERT_MAX    ( IDM_INTERN_BLOCK_INSERT_MIN +
TVG_BLOCKLIST_MAX )
#define IDM_INTERN_BLOCK_EDIT_MIN      IDM_INTERN_BLOCK_INSERT_MAX
#define IDM_INTERN_BLOCK_EDIT_MAX      ( IDM_INTERN_BLOCK_EDIT_MIN +
TVG_BLOCKLIST_MAX )

#define IDM_INTERN_FILE_OPEN_MIN        IDM_INTERN_BLOCK_EDIT_MAX
#define IDM_INTERN_FILE_OPEN_MAX        ( IDM_INTERN_FILE_OPEN_MIN +
TVG_FILE_MAX )

#define IDM_INTERN_SYSTEM_SELECT_MIN    IDM_INTERN_FILE_OPEN_MAX
#define IDM_INTERN_SYSTEM_SELECT_MAX    ( IDM_INTERN_SYSTEM_SELECT_MIN +
TVG_SYSTEM_MAX + 1 )
#define IDM_INTERN_SYSTEM_EDIT_MIN      IDM_INTERN_SYSTEM_SELECT_MAX
#define IDM_INTERN_SYSTEM_EDIT_MAX      ( IDM_INTERN_SYSTEM_EDIT_MIN +
TVG_SYSTEM_MAX + 1 )

#define IDM_INTERN_HATCH_SELECT_MIN     IDM_INTERN_SYSTEM_EDIT_MAX
#define IDM_INTERN_HATCH_SELECT_MAX     ( IDM_INTERN_HATCH_SELECT_MIN +
TVG_HATCH_MAX + 1 )
#define IDM_INTERN_HATCH_EDIT_MIN       IDM_INTERN_HATCH_SELECT_MAX
#define IDM_INTERN_HATCH_EDIT_MAX       ( IDM_INTERN_HATCH_EDIT_MIN +
TVG_HATCH_MAX + 1 )
```

```
#define IDM_INTERN_LAYER_SELECT_MIN     IDM_INTERN_HATCH_EDIT_MAX
#define IDM_INTERN_LAYER_SELECT_MAX     ( IDM_INTERN_LAYER_SELECT_MIN +
TVG_LAYER_MAX + 1 )
#define IDM_INTERN_LAYER_EDIT_MIN       IDM_INTERN_LAYER_SELECT_MAX
#define IDM_INTERN_LAYER_EDIT_MAX       ( IDM_INTERN_LAYER_EDIT_MIN +
TVG_LAYER_MAX + 1 )

#define IDM_INTERN_PEN_SELECT_MIN       IDM_INTERN_LAYER_EDIT_MAX
#define IDM_INTERN_PEN_SELECT_MAX       ( IDM_INTERN_PEN_SELECT_MIN + TVG_PEN_MAX
+ 1 )
#define IDM_INTERN_PEN_EDIT_MIN         IDM_INTERN_PEN_SELECT_MAX
#define IDM_INTERN_PEN_EDIT_MAX         ( IDM_INTERN_PEN_EDIT_MIN + TVG_PEN_MAX +
1 )

#define IDM_INTERN_ZOOM_AREA_MIN        IDM_INTERN_PEN_EDIT_MAX
#define IDM_INTERN_ZOOM_AREA_MAX        ( IDM_INTERN_ZOOM_AREA_MIN +
TVG_WINDOW_MAX + 1 )
#define IDM_INTERN_ZOOM_OBJECTS_MIN     IDM_INTERN_ZOOM_AREA_MAX
#define IDM_INTERN_ZOOM_OBJECTS_MAX     ( IDM_INTERN_ZOOM_OBJECTS_MIN +
TVG_WINDOW_MAX + 1 )
#define IDM_INTERN_ZOOM_PAGE_MIN        IDM_INTERN_ZOOM_OBJECTS_MAX
#define IDM_INTERN_ZOOM_PAGE_MAX        ( IDM_INTERN_ZOOM_PAGE_MIN +
TVG_WINDOW_MAX + 1 )
#define IDM_INTERN_ZOOM_ORIGINAL_MIN    IDM_INTERN_ZOOM_PAGE_MAX
#define IDM_INTERN_ZOOM_ORIGINAL_MAX    ( IDM_INTERN_ZOOM_ORIGINAL_MIN +
TVG_WINDOW_MAX + 1 )
#define IDM_INTERN_ZOOM_FACTOR_MIN      IDM_INTERN_ZOOM_ORIGINAL_MAX
#define IDM_INTERN_ZOOM_FACTOR_MAX      ( IDM_INTERN_ZOOM_FACTOR_MIN +
TVG_WINDOW_MAX + 1 )
#define IDM_INTERN_ZOOM_UNDO_MIN        IDM_INTERN_ZOOM_FACTOR_MAX
#define IDM_INTERN_ZOOM_UNDO_MAX        ( IDM_INTERN_ZOOM_UNDO_MIN +
TVG_WINDOW_MAX + 1 )

#define IDM_INTERN_LAST         IDM_INTERN_ZOOM_UNDO_MAX
```

If one of these values is passed to TosoCommandInternal, the corresponding command will be executed. This has the same effect as the user selecting the command from the menu.
If the command ID passed has the form IDM_???_SUB, this will result in a popup menu appearing that contains the corresponding submenu entries.

Point Types (Constants)

The point types listed in this section are usually used for the declaration of external commands.

Syntax

```
#define POINT_NO                0
```
"No Function"
```
#define POINT_COMMAND           1
```
"Select Command"
```
#define POINT_HELP              2
```
"Select Screen Element or Command"

These point types must not be used by external commands!

Syntax

```
#define POINT_ANY               3
```
"Enter Point"
```
#define POINT_ANY1              4
```
"Enter Point 1"
```
#define POINT_ANY2              5
```
"Enter Point 2"
```
#define POINT_START             6
```
"Enter Start-Point"
```
#define POINT_START1            7
```
"Enter Start-Point of Leg 1"
```
#define POINT_START2            8
```
"Enter Start-Point of Leg 2"
```
#define POINT_END               9
```
"Enter End-Point"
```
#define POINT_END1              10
```
"Enter End-Point of Leg 1"
```
#define POINT_END2              11
```
"Enter End-Point of Leg 2"
```
#define POINT_REFERENCE         12
```
"Enter Reference Point"
```
#define POINT_DESTINATION       13
```
"Enter Target Point"
```
#define POINT_CORNER1           14
```
"Enter Corner 1"
```
#define POINT_CORNER2           15
```
"Enter Corner 2"
```
#define POINT_CORNER3           16
```
"Enter Corner 3"
```
#define POINT_CORNER4           17
```
"Enter Corner 4"
```
#define POINT_CIRCLE1           18
```
"Enter Point 1 on Circle"
```
#define POINT_CIRCLE2           19
```
"Enter Point 2 on Circle"
```
#define POINT_CIRCLE3           20
```
"Enter Point 3 on Circle"

```
#define POINT_ELLIPSE1          21
```
"Enter Point 1 on Ellipse"
```
#define POINT_ELLIPSE2          22
```
"Enter Point 2 on Ellipse"
```
#define POINT_FRAME             23
```
"Enter Point on Frame"
```
#define POINT_CENTER            24
```
"Enter Center"
```
#define POINT_RADIUS            25
```
"Enter Radius"
```
#define POINT_ANGLE             26
```
"Enter Angle"
```
#define POINT_ANGLE1            27
```
"Enter Start-Angle"
```
#define POINT_ANGLE2            28
```
"Enter End-Angle"
```
#define POINT_LINE_RELATIVE     29
```
"Enter Relative Line"
```
#define POINT_PIVOT1            30
```
"Enter Pivot 1"
```
#define POINT_PIVOT2            31
```
"Enter Pivot 2"
```
#define POINT_CURVE_END         32
```
"Enter End-Point of Curve"
```
#define POINT_ARC               33
```
"Enter Point on Arc"
```
#define POINT_ARC_END1          34
```
"Enter End-Point of Arc"
```
#define POINT_ARC_END2          35
```
"Enter End-Point of Arc"
```
#define POINT_ARC_RELATIVE      36
```
"Enter Relative Circular Arc"
```
#define POINT_POSITION          37
```
"Enter Position"
```
#define POINT_SECTION1          38
```
"Section: Enter Corner 1"
```
#define POINT_SECTION2          39
```
"Section: Enter Corner 2"
```
#define POINT_DIM_LINE1         40
```
"Enter Dimension Line Orientation"
```
#define POINT_DIM_LINE2         41
```
"Enter Dimension Line Position"
```
#define POINT_DIM_LINE3         42
```
"Enter Dimension Line End-Point"
```
#define POINT_DIM_NUM1          43
```
"Enter Dimension Position"
```
#define POINT_DIM_NUM2          44
```
"Enter Dimension Orientation"

Standard point entry. No identification is performed, a currently active grid will have effect, snapping is possible.


Contents

Syntax

```
#define POINT_PL                45
```
"Identify Reference Line"
```
#define POINT_PL1               46
```
"Identify Reference Line 1"
```
#define POINT_PL2               47
```
"Identify Reference Line 2"
```
#define POINT_PL3               48
```
"Identify Reference Line 3"
```
#define POINT_PC                49
```
"Identify Reference Circle"
```
#define POINT_PA                50
```
"Identify Reference Circle Part"
```
#define POINT_PE                51
```
"Identify Reference Ellipse"
```
#define POINT_PCE               52
```
"Identify Reference Circle / Ellipse"
```
#define POINT_PCE1              53
```
"Identify Reference Circle / Ellipse 1"
```
#define POINT_PCE2              54
```
"Identify Reference Circle / Ellipse 2"
```
#define POINT_PLC               55
```
"Identify Reference Object"
```
#define POINT_PLC1              56
```
"Identify Reference Object 1"
```
#define POINT_PLC2              57
```
"Identify Reference Object 2"
```
#define POINT_PLC3              58
```
"Identify Reference Object 3"
```
#define POINT_PLE               59
```
"Identify Reference Object"
```
#define POINT_PLCEA             60
```
"Identify Reference Object"

Object-independent identification. If a user clicks near to a unit, it will be temporarily resolved to standard objects, the nearest of which will be memorized. Such an identified standard object can later be retrieved by means of TosoInputGetIdentData. During identification, a currently active grid will have no effect, snapping is disabled.
The IDs are assembled of characters indicating what type of standard objects will be accepted. The characters are:

`L` Accepts lines (normal and geometry)
`C` Accepts circles (normal and geometry, circular arcs will be passed as full circles)
`E` Accepts ellipses (normal and geometry, elliptical arcs will be passed as full ellipses).
`A` Accepts arcs of circles and/or ellipses, depending on whether `C` or `E` is stated before.

For example, the point type `POINT_PLC` accepts lines and circles. If an arc, sector or segment is identified, it is resolved into a full circle and up to 2 lines, and the closest of those standard objects is passed on.


```
#define POINT_ID_ANY            61
```
"Identify Any Object" - this allows to identify objects / instances that are frozen!
```
#define POINT_ID_OBJECT         62
```
"Identify Object"

```
#define POINT_ID_INSTANCE       63
```
"Identify Instance"
```
#define POINT_ID_TRIM           64
```
"Identify Object to be Trimmed"
```
#define POINT_ID_TRIM1          65
```
"Identify Object 1 to be Trimmed"
```
#define POINT_ID_TRIM2          66
```
"Identify Object 2 to be Trimmed"
```
#define POINT_ID_LINE           67
```
"Identify Line"
```
#define POINT_ID_LINE1          68
```
"Identify Line 1"
```
#define POINT_ID_LINE2          69
```
"Identify Line 2"
```
#define POINT_ID_CIRCLE         70
```
"Identify Circle / Circle Part"
```
#define POINT_ID_CIRCLE1        71
```
"Identify Circle / Circle Part 1"
```
#define POINT_ID_CIRCLE2        72
```
"Identify Circle / Circle Part 2"
```
#define POINT_ID_ARC            73
```
"Identify Circle Part"
```
#define POINT_ID_ELLIPSE        74
```
"Identify Ellipse"
```
#define POINT_ID_EARC           75
```
"Identify Ellipse Part"
```
#define POINT_ID_CURVE          76
```
"Identify Curve"
```
#define POINT_ID_SURFACE        77
```
"Identify Surface"
```
#define POINT_ID_CURVE_SURFACE  78
```
"Identify Curve / Surface"
```
#define POINT_ID_FILLED1        79
```
"Identify Areal Object 1"
```
#define POINT_ID_FILLED2        80
```
"Identify Areal Object 2"
```
#define POINT_ID_TEXT           81
```
"Identify Text"
```
#define POINT_ID_DIMENSION1     82
```
"Identify Dimension Line"
```
#define POINT_ID_DIMENSION2     83
```
"Identify Dimension"
```
#define POINT_ID_POINT          84
```
"Identify Point"

Object-dependent identification of a single unit. If a user clicks near to a unit of a suitable type, this object will be marked by setting its identification count to the current identification count. Such an identified object can later be retrieved by means of either TosoInputGetIdentObject or TosoInputGetIdentAddress. During identification, a currently active grid will have no effect, snapping is disabled.

Syntax

```
#define POINT_ID_MULTITEXT1     85
```
"Choose Texts"

```
#define POINT_ID_MULTITEXT2     86
```
"Choose Standard Texts"
```
#define POINT_ID_MULTIDIM1      87
```
"Choose Dimensions"
```
#define POINT_ID_MULTIDIM2      88
```
"Choose Dimension Texts"
```
#define POINT_ID_MULTIINST      89
```
"Choose Instances"
```
#define POINT_ID_MULTIOBJ       90
```
"Choose Objects"
```
#define POINT_ID_MULTIALL       91
```
"Choose Any Objects" - this allows to identify objects / instances that are frozen!
```
#define POINT_ID_POINTS         92
```
"Choose Points"

Object-dependent identification multiple units. If a user identifies a unit by either clicking near to it or by using any other identification method, the unit will be marked by setting the FLAG_USE flag. All objects identified during a multi-unit-identification can later be retrieved by means of TosoEnumerateIdent or they can directly be modified by means of TosoEditIdentMatrix. During identification, a currently active grid will have no effect, snapping is disabled.

Important! As units identified during a multi-unit-identification are marked by settings a flag, only *one* multi-unit-identification is possible during a command's input phase! If a second multi-unit-selection is required, this has to be realized by an additional, previously performed permanent selection (where the flag FLAG_SELECT will be set).

Overview (Macros)

The macros in this section will be listed in form of standard procedure prototypes to show the types of parameters they expect and return. If you are interested in the implementation of these macros, you will have to look inside the header file `TOSO40.H`.

Contents

Syntax

```
int round( double a );
```
Rounding of a *double* value into an *int* value.

```
double fixed( FIXED a );
```
Conversion of a *FIXED* real value (as used by some Win32 procedures) into a *double* value.

```
void lstrclr( LPSTR a );
```
Clearing of a string.

```
double mat_x( MATRIX* m, double x, double y );
double mat_y( MATRIX* m, double x, double y );
```
Multiplication of a coordinate pair (*x*,*y*) with a matrix pointed to by *m*.

```
double mat_dx( MATRIX* m, double x, double y );
double mat_dy( MATRIX* m, double x, double y );
```
Multiplication of a vector (*x*,*y*) with a matrix pointed to by *m*.

```
double mat_sx( MATRIX* m, double x );
double mat_sy( MATRIX* m, double y );
```
Multiplication of a horizontal (*x*) or vertical (*y*) vector with a matrix pointed to by *m*.

```
int db_alignsize( int s );
```
Calculation of a data block's aligned size (multiple of eight) based on the real size *s*.

```
int db_textsize( LPSTR t );
```
Calculation of the aligned size (multiple of eight) of the string *t*.

```
int DB_SIZE_POINT( void );
```
Default size of a data block of type DB_TYPE_POINT (inlcuding one point structure).

```
int DB_SIZE_CONSTANT( void );
```
Default size of a data block of type DB_TYPE_DOUBLE, subtype DB_ALL_CONSTANT.

```
int DB_SIZE_ARC( void );
```
Default size of a data block of type DB_TYPE_DOUBLE, subtype DB_ALL_ARC.

```
int DB_SIZE_CURVE( void );
```
Default size of a data block of type DB_TYPE_DOUBLE, subtype DB_ALL_CURVE.

```
int DB_SIZE_TEXT( void );
```
Default size of a data block of type DB_TYPE_TEXT (without any text).

```
int DB_SIZE_END( void );
```
Default size of a data block of type DB_TYPE_NATIVE, subtype DB_END.

```
int DB_SIZE_ANY( void );
```
Default size of any data block of type (without data).

```
BLOCK_PTR db_offset( UNIT_OBJECT_PTR b, OFFSET x );
```
Calculates a data block pointer with the offset *x* relative to the data area of the object pointed to by *b*.

```
BLOCK_PTR db_first( UNIT_OBJECT_PTR b );
```
Calculates the pointer to the first data block of the object pointed to by *b*.

```
BLOCK_PTR db_next( BLOCK_PTR b );
```

Calculates the pointer to the data block that lies behind the data block pointed to by *b*.

```
OFFSET db_difference( UNIT_OBJECT_PTR b, BLOCK_PTR p );
```
Calculates the offset of the data block pointed to by *p* relative to the data area of the object pointed to by *b*.

```
DPOINT db_point( BLOCK_PTR b );
```
Acesses the first point entry of the data block (type DB_TYPE_POINT) pointed to by *b*. This macro may be used on both sides of an assignment!

```
double db_constant( BLOCK_PTR b );
```
Acesses the first double entry of the data block (type DB_TYPE_DOUBLE) pointed to by *b*. This macro may be used on both sides of an assignment!

```
double db_orient( BLOCK_PTR b );
```
Acesses the first double entry of the data block (type DB_TYPE_DOUBLE) pointed to by *b*. This macro may be used on both sides of an assignment!

```
double db_curve( BLOCK_PTR b );
```
Acesses the second double entry of the data block (type DB_TYPE_DOUBLE) pointed to by *b*. This macro may be used on both sides of an assignment!

```
LPSTR db_attribute_name( BLOCK_PTR b );
```
Acesses the attribute name of the data block (type DB_TYPE_ATTRIB) pointed to by *b*. This macro may be used on both sides of an assignment!

```
LPSTR db_attribute_text( BLOCK_PTR b );
```
Acesses the attribute text of the data block (type DB_TYPE_ATTRIB) pointed to by *b*. This macro may be used on both sides of an assignment!

The following macros can be used to initialize data block pointers for all complex object types. These macros rely on the following, specific variable names which may not be evident to you:

```
BLOCK_PTR                b1, b2, b3, b4, b5, b6, b7, b8;
BLOCK_TEXT_PTR           a110, b110, c110, d110, e110, f110, g110;
BLOCK_DIMLINE_PTR        b220;
BLOCK_DIMLARGE_PTR       b225;
BLOCK_DIMSMALL_PTR       b230;
BLOCK_TEXTSTANDARD_PTR   b235;
BLOCK_TEXTFRAME_PTR      b236;
BLOCK_TEXTREFERENCE_PTR  b237;
BLOCK_CLIPSURFACE_PTR    b242;
BLOCK_BITMAPREF_PTR      b243;
BLOCK_T003_PTR           t003;
BLOCK_T004_PTR           t004;
BLOCK_T005_PTR           t005;
BLOCK_T006_PTR           t006;
BLOCK_T007_PTR           t007;
BLOCK_T008_PTR           t008;
```

If you want to use the macros without modification, those variables have to be defined at the time the macros is used. This can be done using the macro `DB_PTR_DECLARATION` . If you would prefer to use different variable names, you can use these macros as examples for your own macros.

Contents

Syntax

```
void db_ptr_dline( UNIT_OBJECT_PTR p )
```
Initializes the standard data block pointers for an object of type OBJ_DLINE, pointed to by *p*.

```
void db_ptr_darc( UNIT_OBJECT_PTR p )
```

Initializes the standard data block pointers for an object of type OBJ_DARC, pointed to by *p*.

`void db_ptr_ddistance( UNIT_OBJECT_PTR p )`
Initializes the standard data block pointers for an object of type OBJ_DDISTANCE, pointed to by *p*.

`void db_ptr_dradius( UNIT_OBJECT_PTR p )`
Initializes the standard data block pointers for an object of type OBJ_DRADIUS, pointed to by *p*.

`void db_ptr_dangle( UNIT_OBJECT_PTR p )`
Initializes the standard data block pointers for an object of type OBJ_DANGLE, pointed to by *p*.

`void db_ptr_darclength( UNIT_OBJECT_PTR p )`
Initializes the standard data block pointers for an object of type OBJ_DARCLENGTH, pointed to by *p*.

`void db_ptr_dcoordinate( UNIT_OBJECT_PTR p )`
Initializes the standard data block pointers for an object of type OBJ_DCOORDINATE, pointed to by *p*.

`void db_ptr_darea( UNIT_OBJECT_PTR p )`
Initializes the standard data block pointers for an object of type OBJ_DAREA, pointed to by *p*.

`void db_ptr_textstandard( UNIT_OBJECT_PTR p )`
Initializes the standard data block pointers for an object of type OBJ_TEXTSTANDARD, pointed to by *p*.

`void db_ptr_textframe( UNIT_OBJECT_PTR p )`
Initializes the standard data block pointers for an object of type OBJ_TEXTFRAME, pointed to by *p*.

`void db_ptr_textreference( UNIT_OBJECT_PTR p )`
Initializes the standard data block pointers for an object of type OBJ_TEXTREFERENCE, pointed to by *p*.

`void db_ptr_comment( UNIT_OBJECT_PTR p )`
Initializes the standard data block pointers for an object of type OBJ_COMMENT, pointed to by *p*.

Note: All macros working with structure sizes are defined with type *int* or *long* instead of *size_t*, *DWORD* or *UINT*, as the serving application will always use *int* variables. The authors do not understand the use of unsigned values of any type. To our opinion, those unsigned values are useless relicts of the good old eight-bit-time when each bit was precious...