"Contents"

Contents of EFLIB help file

Units: *EFDEF EFKERNEL EFMSG EFCMAN EFADT EFBASIC EFMEMORY EFELEM EFPLUG EFSTRING EFAPP EFIO EFSTREAM EFTEXT EFFILE EFFILTER EFLIST EFKEYBRD EFMOUSE EFDEVICE EFARRAY EFTREE EFDATA EFTABLE EFMATH EFVIEWS EFGUI EFCONDRV EFCLINE EFCONV EFMEXP EFPATRN EFREG EFRES EFSCREEN EFSEARCH EFSORT EFSTYLES EFSYSTEM EFTEST EFTIME*

absolute — EFKEYBRD

EFKEYBRD.absolute

ADTSelector — EFADT

Const EFADT.ADTSelector: pADTSelector = NIL;

ADT component selector. This is an instance of *tADTSelector* or a derived class. This instance supply ADT's with their default components, for example key plugs and elements.

AllChars — EFDEF

Const EFDEF.AllChars: tChars = [#0..#255] ;

Complete character set: all characters in the ASCII set.

Application — EFAPP

Const EFAPP.Application: pApplication = NIL;

Application engine. Refer to *tApplication* for details.

Arccos EFMATH

function EFMATH.Arccos(X: tBaseReal): tBaseReal;

Returns the arc cosine for the given number expressed in radians.

Arcsin EFMATH

function EFMATH.Arcsin(X: tBaseReal): tBaseReal;

Returns the arc sine for the given number expressed in radians.

ArctanXY EFMATH

function EFMATH.ArctanXY(X, Y: tBaseReal): tBaseReal;

Returns the arc tangent with quadrant check (X is the denominator or the real axis value and Y is the numerator or the imaginary axis value). The angle is adjusted to be within [-Pi, Pi].

Assert EFKERNEL

procedure EFKERNEL.Assert(Condition: boolean; Identity: word);

Asserts the condition or reports an error. Use the built-in tObject assertion method in classes. This procedure is provided for your main program.

BinaryStringNumber EFSTRING

function EFSTRING.BinaryStringNumber(Data: string): longint;

Converts the specified binary number (stored as 1 and 0 in a common Pascal string) to a longint value, e.g. '11111111' -> 255.

Black EFVIEWS

Const EFVIEWS.Black = 0;

Blink EFVIEWS

Const EFVIEWS.Blink = 128;

Blue EFVIEWS

Const EFVIEWS.Blue = 1;

Brown EFVIEWS

Const EFVIEWS.Brown = 6;

Caster EFMATH

Const EFMATH.Caster: pCaster = NIL;

This is a instanciation of the *tCaster* class (or any other class derived from this class). Whenever a tMathObject descendant require an explicit or implicit type-casting, it calls the Caster instance's Cast method. Replace this instance, if you add new mathematical objects to EFLIB and want type-casting capabilities.

Ceiling EFMATH

---

function EFMATH.Ceiling(X: tBaseReal): tBaseReal;

Returns the next highest integer, ie. the cloest integer larger than or equal to the specified real number.

cfLock — EFSTYLES

Const EFSTYLES.cfLock = 01;

cfManager — EFSTYLES

Const EFSTYLES.cfManager = 04;

cfModal — EFSTYLES

Const EFSTYLES.cfModal = 03;

cfProtection — EFSTYLES

Const EFSTYLES.cfProtection = 02;

CIm — EFMATH

Const EFMATH.CIm: pComplex = NIL;

The complex number (0, 1) <=> i. This is a predefined constant.

Classes — EFKERNEL

Const EFKERNEL.Classes: pClassManager = NIL;

This is the class hierarchy manager. This hierarchy is used for identification of instances and relation checks of classes. The manager also maintains information for stream storage.

Combinations — EFMATH

function EFMATH.Combinations(Maximum, Items: word): tBaseReal;

Returns the number of possible combinations of the specified number of items in a set of the specified maximum number of items, ie. the number of valid subsets with a certain number of items.

CompareVariable — EFKERNEL

function EFKERNEL.CompareVariable(var Data1; Size1: word; var Data2; Size2: word): shortint;

Returns 1, 0 or -1 according to if the first variable (Data1, Size1) is bigger, equal or smaller then the second variable (Data2, Size2).

COne — EFMATH

Const EFMATH.COne: pComplex = NIL;

The complex number (1, 0) <=> 1. This is a predefined constant.

ConsoleSelector — EFCONDRV

Const EFCONDRV.ConsoleSelector: pConsoleSelector = NIL;

Console component selector. This is an instance of *tConsoleSelector* or a derived class. This instance supply consoles with their default components, for example colors and fonts.

Cosh — EFMATH

function EFMATH.Cosh(X: tBaseReal): tBaseReal;
Returns the hyperbolic cosine of the given angle.

CRC32Table EFFILTER

Const EFFILTER.CRC32Table: array = ($00000000, $77073096, $ee0e612c, $990951ba, $076dc419, $706af48f, $e963a535, $9e6495a3, $0edb8832, $79dcb8a4, $e0d5e91e, $97d2d988, $09b64c2b, $7eb17cbd, $e7b82d07, $90bf1d91, $1db71064, $6ab020f2, $f3b97148, $84be41de, $1adad47d, $6ddde4eb, $f4d4b551, $83d385c7, $136c9856, $646ba8c0, $fd62f97a, $8a65c9ec, $14015c4f, $63066cd9, $fa0f3d63, $8d080df5, $3b6e20c8, $4c69105e, $d56041e4, $a2677172, $3c03e4d1, $4b04d447, $d20d85fd, $a50ab56b, $35b5a8fa, $42b2986c, $dbbbc9d6, $acbcf940, $32d86ce3, $45df5c75, $dcd60dcf, $abd13d59, $26d930ac, $51de003a, $c8d75180, $bfd06116, $21b4f4b5, $56b3c423, $cfba9599, $b8bda50f, $2802b89e, $5f058808, $c60cd9b2, $b10be924, $2f6f7c87, $58684c11, $c1611dab, $b6662d3d, $76dc4190, $01db7106, $98d220bc, $efd5102a, $71b18589, $06b6b51f, $9fbfe4a5, $e8b8d433, $7807c9a2, $0f00f934, $9609a88e, $e10e9818, $7f6a0dbb, $086d3d2d, $91646c97, $e6635c01, $6b6b51f4, $1c6c6162, $856530d8, $f262004e, $6c0695ed, $1b01a57b, $8208f4c1, $f50fc457, $65b0d9c6, $12b7e950, $8bbeb8ea, $fcb9887c, $62dd1ddf, $15da2d49, $8cd37cf3, $fbd44c65, $4db26158, $3ab551ce, $a3bc0074, $d4bb30e2, $4adfa541, $3dd895d7, $a4d1c46d, $d3d6f4fb, $4369e96a, $346ed9fc, $ad678846, $da60b8d0, $44042d73, $33031de5, $aa0a4c5f, $dd0d7cc9, $5005713c, $270241aa, $be0b1010, $c90c2086, $5768b525, $206f85b3, $b966d409, $ce61e49f, $5edef90e, $29d9c998, $b0d09822, $c7d7a8b4, $59b33d17, $2eb40d81, $b7bd5c3b, $c0ba6cad, $edb88320, $9abfb3b6, $03b6e20c, $74b1d29a, $ead54739, $9dd277af, $04db2615, $73dc1683, $e3630b12, $94643b84, $0d6d6a3e, $7a6a5aa8, $e40ecf0b, $9309ff9d, $0a00ae27, $7d079eb1, $f00f9344, $8708a3d2, $1e01f268, $6906c2fe, $f762575d, $806567cb, $196c3671, $6e6b06e7, $fed41b76, $89d32be0, $10da7a5a, $67dd4acc, $f9b9df6f, $8ebeeff9, $17b7be43, $60b08ed5, $d6d6a3e8, $a1d1937e, $38d8c2c4, $4fdff252, $d1bb67f1, $a6bc5767, $3fb506dd,

*$48b2364b, $d80d2bda, $af0a1b4c, $36034af6,
$41047a60, $df60efc3, $a867df55, $316e8eef, $4669be79,
$cb61b38c, $bc66831a, $256fd2a0, $5268e236, $cc0c7795,
$bb0b4703, $220216b9, $5505262f, $c5ba3bbe, $b2bd0b28,
$2bb45a92, $5cb36a04, $c2d7ffa7, $b5d0cf31, $2cd99e8b,
$5bdeae1d, $9b64c2b0, $ec63f226, $756aa39c, $026d930a,
$9c0906a9, $eb0e363f, $72076785, $05005713,
$95bf4a82, $e2b87a14, $7bb12bae, $0cb61b38, $92d28e9b,
$e5d5be0d, $7cdcefb7, $0bdbdf21, $86d3d2d4, $f1d4e242,
$68ddb3f8, $1fda836e, $81be16cd, $f6b9265b, $6fb077e1,
$18b74777, $88085ae6, $ff0f6a70, $66063bca, $11010b5c,
$8f659eff, $f862ae69, $616bffd3, $166ccf45, $a00ae278,
$d70dd2ee, $4e048354, $3903b3c2, $a7672661,
$d06016f7, $4969474d, $3e6e77db, $aed16a4a,
$d9d65adc, $40df0b66, $37d83bf0, $a9bcae53, $debb9ec5,
$47b2cf7f, $30b5ffe9, $bdbdf21c, $cabac28a, $53b39330,
$24b4a3a6, $bad03605, $cdd70693, $54de5729,
$23d967bf, $b3667a2e, $c4614ab8, $5d681b02, $2a6f2b94,
$b40bbe37, $c30c8ea1, $5a05df1b, $2d02ef8d);*

CRC 32 table; constants used for CRC 32 calculation when the compiler directive CRCTABLES is defined. Refer to FLAGS.INC for further information.

CreateRandomElement — EFTEST

function EFTEST.CreateRandomElement(Size: word): pGenericElement; far;

Creates a random tGenericElement instance using the current random seed. This function is called from example programs.

CreateRandomString — EFTEST

function EFTEST.CreateRandomString(Length: word): pString; far;

Creates a random tString instance with the specified length.

Cyan — EFVIEWS

Const EFVIEWS.Cyan = 3;

DarkGray — EFVIEWS

Const EFVIEWS.DarkGray = 8;

DefaultDelimeters — EFSTRING

Const EFSTRING.DefaultDelimeters: pToken = NIL;

Default text separator sequences (chained *tToken* instances). The Expand and Split methods in tToken uses these delimeters to decide where strings can be splitted. This instance is preset to the delimeters blank space and *Endl*.

DeleteFile — EFSTREAM

function EFSTREAM.DeleteFile(Filename: string): boolean;

Deletes a file and returns TRUE if deletion succeeded.

DirectoryLevel — EFSTREAM

function EFSTREAM.DirectoryLevel(WhatDirectory: string): integer;

Returns the tree level of specified directory, ie. the number of directories that are between this directory and the root directory in the drive.

DirectoryWithBackslash — EFSTREAM

function EFSTREAM.DirectoryWithBackslash(WhatDirectory: string): string;

Returns the directory ended with a backslash character.

DosMajorVersion — EFKERNEL

function EFKERNEL.DosMajorVersion: byte;

Returns the major DOS version number, eg. 6 for version 6.2.

EFADT

Unit EFADT (in EFLIB.TEX help file)

EFLIB | Extended Function Library (C) Johan Larsson, 1992 - 1998 All rights reserved. E-mail to jola@ts.umu.se.

RELEASE 3 PROTOTYPE 5.1 [Last updated 970804]

Contents: ABSTRACT DATA TYPE interface object. This unit contains tADT, tOrderedADT and tLinearADT - the three base classes for all data structures in EFLIB. It also contains some ADT plug classes: modules that plugs into ADTs and help them doing certain tasks, for instance comparing elements or sorting.

PROTOTYPE 6 WILL PROVIDE AN ENTIRELY REDESIGNED ADT ENGINE WITHOUT ANY BACKWARD COMPATIBLITY. USE WITH CAUTION.

Revision changes: 970804 [J]: Prototype 5.1 release.

Authors: [J] Johan Larsson, author of EFLIB. http://www.ts.umu.se/~jola/, jola@ts.umu.se.

| THE USE OF EFLIB REQUIRES ACCEPTANCE OF THE LICENSE AGREEMENT | ENCLOSED IN

EVERY DISTRIBUTABLE PACKAGE. REFER TO LICENSE.TXT | OR THE EFLIB WEB SITE FOR FURTHER INFORATION. EFLIB IS PROTECTED | BY SWEDISH COPYRIGHT LAW AND INTERNATIONAL TREARTY PROVISIONS. | YOU AGREE THAT YOUR USE OF EFLIB IS SUBJECT TO THESE LAWS, | WHICH PROHIBIT UNAUTHORISED COPYING OR DUPLICATION OF EFLIB. | | THIS HEADER FILE MAY NEVER BE REMOVED. YOU ARE PERMITTED TO | ADD INFORMATION UNDER "REVISION CHANGES", "AUTHORS" AND | "CONTENTS".

See also: http://ftp.sunet.se/EFLIB/ http://ftp.sunet.se/EFLIB/Legal/License/

Constants: *ADTSelector*

Types: *pIterator pLinearIterator pADT tADT pLinearADT tLinearADT pOrderedADT tOrderedADT pADTStream tADTStream tIterator tLinearIterator pADTPlug tADTPlug pKeyPlug tKeyPlug pBoundedKeyPlug tBoundedKeyPlug pReversedKeyPlug tReversedKeyPlug pSortPlug tSortPlug pQuickSortPlug tQuickSortPlug pADTSelector tADTSelector*

EFAPP

Unit EFAPP (in EFLIB.TEX help file)

EFLIB | Extended Function Library (C) Johan Larsson, 1992 - 1998 All rights reserved. E-mail to jola@ts.umu.se.

RELEASE 3 PROTOTYPE 5.1 [Last updated 970603]

Contents: APPLICATION ENGINE. This unit contains the main component for applications. *tApplication* provides the main event loop - the origin of all events in your application. tApplication is the communication link between devices (keyboard, mouse, ...) and components.

Revision changes: 970603 [J]: Prototype 5.1 release.

Authors: [J] Johan Larsson, author of EFLIB. http://www.ts.umu.se/~jola/, jola@ts.umu.se.

| THE USE OF EFLIB REQUIRES ACCEPTANCE OF THE LICENSE AGREEMENT | ENCLOSED IN

EVERY DISTRIBUTABLE PACKAGE. REFER TO LICENSE.TXT | OR THE EFLIB WEB SITE FOR FURTHER INFORATION. EFLIB IS PROTECTED | BY SWEDISH COPYRIGHT LAW AND INTERNATIONAL TREARTY PROVISIONS. | YOU AGREE THAT YOUR USE OF EFLIB IS SUBJECT TO THESE LAWS, | WHICH PROHIBIT UNAUTHORISED COPYING OR DUPLICATION OF EFLIB. | | THIS HEADER FILE MAY NEVER BE REMOVED. YOU ARE PERMITTED TO | ADD INFORMATION UNDER "REVISION CHANGES", "AUTHORS" AND | "CONTENTS".

See also: http://ftp.sunet.se/EFLIB/ http://ftp.sunet.se/EFLIB/Legal/License/

Constants: *Application*

Types: *pApplication tApplication*

EFARRAY

Unit EFARRAY (in EFLIB.TEX help file)

EFLIB | Extended Function Library (C) Johan Larsson, 1992 - 1998 All rights reserved. E-mail to jola@ts.umu.se.

RELEASE 3 PROTOTYPE 5.1 [Last updated 970627]

Contents: ARRAY ADT'S with polymorphism. This unit contains a variety of array classes: tArray, tBoundedArray and tVirtualArray. Arrays are fast linear data structures, and are sometimes used interally by other ADT's.

Revision changes: 970627 [J]: Prototype 5.1 release.

Authors: [J] Johan Larsson, author of EFLIB. http://www.ts.umu.se/~jola/, jola@ts.umu.se.

| THE USE OF EFLIB REQUIRES ACCEPTANCE OF THE LICENSE AGREEMENT | ENCLOSED IN EVERY DISTRIBUTABLE PACKAGE. REFER TO LICENSE.TXT | OR THE EFLIB WEB SITE FOR FURTHER INFORATION. EFLIB IS PROTECTED | BY SWEDISH COPYRIGHT LAW AND INTERNATIONAL TREARTY PROVISIONS. | YOU

See also: http://ftp.sunet.se/EFLIB/ http://ftp.sunet.se/EFLIB/Legal/License/

Types: *pSparceIterator pArray tArray pBoundedArray tBoundedArray pVirtualArray tVirtualArray tSparceIterator*

EFBASIC

Unit EFBASIC (in EFLIB.TEX help file)

EFLIB | Extended Function Library (C) Johan Larsson, 1992 - 1998 All rights reserved. E-mail to jola@ts.umu.se.

RELEASE 3 PROTOTYPE 5.1 [Last updated 970701]

Contents: BASIC ROUTINES. Miscellaneous components such as tSet, tCharacters, tCoordinate and tField. tTime and tTimer are preliminary located in this unit, though they are moved to a separate unit in the current working model of PROTOTYPE 6.

Revision changes: 970701 [J]: Prototype 5.1 release.

Authors: [J] Johan Larsson, author of EFLIB. http://www.ts.umu.se/~jola/, jola@ts.umu.se.

INFORMATION UNDER "REVISION CHANGES", "AUTHORS" AND | "CONTENTS".

See also: http://ftp.sunet.se/EFLIB/ http://ftp.sunet.se/EFLIB/Legal/License/

Types: *pSet tSet pCharacters tCharacters pField pCoordinates tCoordinates tField pTime tTime pTimer tTimer*

Functions and Procedures: *SearchVariable SwapVariable SummarizeData*

EFCLINE

Unit EFCLINE (in EFLIB.TEX help file)

EFLIB | Extended Function Library (C) Johan Larsson, 1992 - 1998 All rights reserved. E-mail to jola@ts.umu.se.

RELEASE 3 PROTOTYPE 5.1 [Last updated 970628]

Contents: COMMAND LINE TOOLS. This unit contains classes that helps you with the managing of arguments that are passed to the program as command line parameters.

Revision changes: 970628 [J]: Prototype 5.1 release.

Authors: [J] Johan Larsson, author of EFLIB. http://www.ts.umu.se/~jola/, jola@ts.umu.se.

| THE USE OF EFLIB REQUIRES ACCEPTANCE OF THE LICENSE AGREEMENT | ENCLOSED IN EVERY DISTRIBUTABLE PACKAGE. REFER TO LICENSE.TXT | OR THE EFLIB WEB SITE FOR FURTHER INFORATION. EFLIB IS PROTECTED | BY SWEDISH COPYRIGHT LAW AND INTERNATIONAL TREARTY PROVISIONS. | YOU AGREE THAT YOUR USE OF EFLIB IS SUBJECT TO THESE LAWS, | WHICH PROHIBIT UNAUTHORISED COPYING OR DUPLICATION OF EFLIB. | | THIS HEADER FILE MAY NEVER BE REMOVED. YOU ARE PERMITTED TO | ADD INFORMATION UNDER "REVISION CHANGES", "AUTHORS" AND | "CONTENTS".

See also: http://ftp.sunet.se/EFLIB/ http://ftp.sunet.se/EFLIB/Legal/License/

Types: *pCommandLine tCommandLine*

EFCMAN

Unit EFCMAN (in EFLIB.TEX help file)

EFLIB | Extended Function Library (C) Johan Larsson, 1992 - 1998 All rights reserved. E-mail to jola@ts.umu.se.

RELEASE 3 PROTOTYPE 5.1 [Last updated 970701]

Contents: COMPONENT-MANAGER ARCHITECTURE. This unit contains the component definition (*tComponent*), the manager subclasses (*tManager* and *tCommunicator*) and the primtive message classes.

THIS UNIT IS CURRENTLY EVOLVING. PROTOTYPE 6 WILL PROVIDE A DIFFERENT SOLUTION AND A GUI BUILT ON THESE CLASSES.

Revision changes: 970701 [J]: Prototype 5.1 release.

Authors: [J] Johan Larsson, author of EFLIB. http://www.ts.umu.se/~jola/, jola@ts.umu.se.

| THE USE OF EFLIB REQUIRES ACCEPTANCE OF THE LICENSE AGREEMENT | ENCLOSED IN EVERY DISTRIBUTABLE PACKAGE. REFER TO LICENSE.TXT | OR THE EFLIB WEB SITE FOR FURTHER INFORATION. EFLIB IS PROTECTED | BY SWEDISH COPYRIGHT LAW AND INTERNATIONAL TREARTY PROVISIONS. | YOU AGREE THAT YOUR USE OF EFLIB IS SUBJECT TO THESE LAWS, | WHICH PROHIBIT UNAUTHORISED COPYING OR DUPLICATION OF EFLIB. | | THIS HEADER FILE MAY NEVER BE REMOVED. YOU ARE PERMITTED TO | ADD INFORMATION UNDER "REVISION CHANGES", "AUTHORS" AND | "CONTENTS".

See also: http://ftp.sunet.se/EFLIB/ http://ftp.sunet.se/EFLIB/Legal/License/

Types: *pComponent pMessage tMessage pServiceMessage tServiceMessage pManager tComponent tManager pCommunicator tCommunicator*

Unit EFCONDRV (in EFLIB.TEX help file)

RELEASE 3 PROTOTYPE 5.1 [Last updated 970607]

Contents: CONSOLE DRIVERS AND BASIC GUI COMPONENTS. This unit contains the core of EFLIB′s graphical user interface: the console drivers (tConsole), the color class (tColor) and the font class (tFont).

THIS UNIT DOES NOT COMPILE. IT IS AN INTERFACE OUTLINE FOR PROTOTYPE 6.

Revision changes: 970607 [J]: Prototype 5.1 release.

Authors: [J] Johan Larsson, author of EFLIB. http://www.ts.umu.se/∼jola/, jola@ts.umu.se.

See also: http://ftp.sunet.se/EFLIB/ http://ftp.sunet.se/EFLIB/Legal/License/

Constants: *Screen ConsoleSelector*

Types: *pColor tColor pConsole pFont tFont pLineStyle tLineStyle pFillStyle tFillStyle tConsole pBoundedConsole tBoundedConsole pCRT tCRT pVirtualConsole tVirtualConsole pImage tImage pConsoleSelector tConsoleSelector*

EFCONV

Unit EFCONV (in EFLIB.TEX help file)

EFLIB | Extended Function Library (C) Johan Larsson, 1992 - 1998 All rights reserved. E-mail to jola@ts.umu.se.

RELEASE 3 PROTOTYPE 5.1 [Last updated 970624]

Contents: CONVERSION TOOLS. This unit contains the class *tConverter* that can replace expressions in streams with other expression. tConverter is a very flexible file conversion tool.

THIS UNIT DOES NOT COMPILE. IT IS AN INTERFACE OUTLINE FOR PROTOTYPE 6.

Revision changes: 970624 [J]: Prototype 5.1 release.

Authors: [J] Johan Larsson, author of EFLIB. http://www.ts.umu.se/~jola/, jola@ts.umu.se.

| THE USE OF EFLIB REQUIRES ACCEPTANCE OF THE LICENSE AGREEMENT | ENCLOSED IN EVERY DISTRIBUTABLE PACKAGE. REFER TO LICENSE.TXT | OR THE EFLIB WEB SITE FOR FURTHER INFORATION. EFLIB IS PROTECTED | BY SWEDISH COPYRIGHT LAW AND INTERNATIONAL TREARTY PROVISIONS. | YOU AGREE THAT YOUR USE OF EFLIB IS SUBJECT TO THESE LAWS, | WHICH PROHIBIT UNAUTHORISED COPYING OR DUPLICATION OF EFLIB. | | THIS HEADER FILE MAY NEVER BE REMOVED. YOU ARE PERMITTED TO | ADD INFORMATION UNDER "REVISION CHANGES", "AUTHORS" AND | "CONTENTS".

See also: http://ftp.sunet.se/EFLIB/ http://ftp.sunet.se/EFLIB/Legal/License/

Types: *pConverter tConverter*

EFDATA

Unit EFDATA (in EFLIB.TEX help file)

EFLIB | Extended Function Library (C) Johan Larsson, 1992 - 1998 All rights reserved. E-mail to jola@ts.umu.se.

RELEASE 3 PROTOTYPE 5.1 [Last updated 970616]

Contents: ABSTRACT DATA TYPES with polymorphism. This unit contains some classic data structures: stacks and queues. These data structures are composites of linear data structures, but are themself descendants of tOrderedADT. Classes: tComposteADT, tStack, tQueue, tCircularQueue and tPriorityQueue.

Revision changes: 970616 [J]: Prototype 5.1 release.

Authors: [J] Johan Larsson, author of EFLIB. http://www.ts.umu.se/~jola/, jola@ts.umu.se.

| THE USE OF EFLIB REQUIRES ACCEPTANCE OF THE LICENSE AGREEMENT | ENCLOSED IN EVERY DISTRIBUTABLE PACKAGE. REFER TO LICENSE.TXT | OR THE EFLIB WEB SITE FOR FURTHER INFORATION. EFLIB IS PROTECTED | BY SWEDISH COPYRIGHT LAW AND INTERNATIONAL TREARTY PROVISIONS. | YOU AGREE THAT YOUR USE OF EFLIB IS SUBJECT TO THESE LAWS, | WHICH PROHIBIT UNAUTHORISED COPYING OR DUPLICATION OF EFLIB. | | THIS HEADER FILE MAY NEVER BE REMOVED. YOU ARE PERMITTED TO | ADD INFORMATION UNDER "REVISION CHANGES", "AUTHORS" AND | "CONTENTS".

See also: http://ftp.sunet.se/EFLIB/ http://ftp.sunet.se/EFLIB/Legal/License/

Types: *pCompositeADT tCompositeADT pStack tStack pQueue tQueue pCircularQueue tCircularQueue pPriorityQueue tPriorityQueue*

EFDEF

Unit EFDEF (in EFLIB.TEX help file)

EFLIB | Extended Function Library (C) Johan Larsson, 1992 - 1998 All rights reserved. E-mail to jola@ts.umu.se.

RELEASE 3 PROTOTYPE 5.1 [Last updated 970606]

Contents: CONSTANTS AND TYPE DEFINITIONS. The EFDEF unit contains global settings for the library.

Revision changes: 970606 [J]: Prototype 5.1 release.

Authors: [J] Johan Larsson, author of EFLIB. http://www.ts.umu.se/~jola/, jola@ts.umu.se.

See also: http://ftp.sunet.se/EFLIB/ http://ftp.sunet.se/EFLIB/Legal/License/

Constants: *StandardChars AllChars TextChars NumericChars FilenameChars Null EOL sFocused sClose sTouch sUpdate sProtect sUnprotect sLock sUnlock Error_VMT Error_General Error_Abstract Error_Forbidden Error_Assertion Error_Allocation Error_Resource Error_Damaged Error_Compatibility Error_Selector Error_PlugNotFound Error_Extension Error_Registration Error_Identity Error_Message Error_Manager Error_Stream Error_StreamStorage Error_InvalidIndex Error_Access Error_Address Error_CopyIn Error_CopyOut Error_OutOfMemory Error_FileNotFound Error_InvalidFilename Error_File Error_Stream_Mode Error_Stream_Read Error_Stream_Write Error_Stream_Buffer Error_Filter_Base Error_ADT_Compatibility Error_ADT_InvalidElement Error_ADT_ReadOnly Error_ADT_Capacity Error_GUI_Forbidden Error_GUI_Boundaries Error_GUI_InvalidComponent Error_GUI_ForbiddenComponent Error_GUI_Memory Error_Math_Arithmetics Error_Math_Syntax Error_Math_Typecast*

Types: *pByteArray tByteArray pCharArray tCharArray pHugeArray tHugeArray pChars tChars pBit tBit tSortOrder tSortMethod tSearchMethod tTreeOrder pProcessPhase tProcessPhase pAlignemnt tAlignment pDirection tDirection pPresentationMethod tPresentationMethod pBorder tBorder*

EFDEVICE

Unit EFDEVICE (in EFLIB.TEX help file)

EFLIB | Extended Function Library (C) Johan Larsson, 1992 - 1998 All rights reserved. E-mail to jola@ts.umu.se.

RELEASE 3 PROTOTYPE 5.1 [Last updated 970607]

Contents: ABSTRACT DEVICE CLASS AND SOUND DEVICE. This unit supply the basic device class, tDevice, and the descendant tSoundDevice.

Revision changes: 970607 [J]: Prototype 5.1 release.

Authors: [J] Johan Larsson, author of EFLIB. http://www.ts.umu.se/~jola/, jola@ts.umu.se.

| THE USE OF EFLIB REQUIRES ACCEPTANCE OF THE LICENSE AGREEMENT | ENCLOSED IN EVERY DISTRIBUTABLE PACKAGE. REFER TO LICENSE.TXT | OR THE EFLIB WEB SITE FOR FURTHER INFORATION. EFLIB IS PROTECTED | BY SWEDISH COPYRIGHT LAW AND INTERNATIONAL TREARTY PROVISIONS. | YOU AGREE THAT YOUR USE OF EFLIB IS SUBJECT TO THESE LAWS, | WHICH PROHIBIT UNAUTHORISED COPYING OR DUPLICATION OF EFLIB. | | THIS HEADER FILE MAY NEVER BE REMOVED. YOU ARE PERMITTED TO | ADD INFORMATION UNDER "REVISION CHANGES", "AUTHORS" AND | "CONTENTS".

See also: http://ftp.sunet.se/EFLIB/ http://ftp.sunet.se/EFLIB/Legal/License/

Constants: *Speaker*

Types: *pInstallMessage tInstallMessage pDevice tDevice pSoundDevice tSoundDevice*

Unit EFELEM (in EFLIB.TEX help file)

RELEASE 3 PROTOTYPE 5.1 [Last updated 970628]

Contents: ELEMENT CLASSES for data structures. This unit contains a set of elements that can be used in EFLIB′s data structures. All elements descend from the tElement class.

PROTOTYPE 6 WILL PROVIDE AN ENTIRELY REDESIGNED ELEMENT HANDLING BASED ON TOBJECT CLASSES, EXTENDED IN TELEMENT; NO BACKWARD COMPATIBLITY IS ENSURED.

Revision changes: 970628 [J]: Prototype 5.1 release.

Authors: [J] Johan Larsson, author of EFLIB. http://www.ts.umu.se/∼jola/, jola@ts.umu.se.

See also: http://ftp.sunet.se/EFLIB/ http://ftp.sunet.se/EFLIB/Legal/License/

Types: *pElement tElement pGenericElement tGenericElement pStaticElement tStaticElement pIdentityElement tIdentityElement pCarrierElement tCarrierElement pStreamElement tStreamElement*

EFFILE

Unit EFFILE (in EFLIB.TEX help file)

EFLIB | Extended Function Library (C) Johan Larsson, 1992 - 1998 All rights reserved. E-mail to jola@ts.umu.se.

RELEASE 3 PROTOTYPE 5.1 [Last updated 970627]

Contents: FILE HANDLING. This unit contains the buffered file streams and basic file handling routines.

Revision changes: 970627 [J]: Prototype 5.1 release.

Authors: [J] Johan Larsson, author of EFLIB. http://www.ts.umu.se/~jola/, jola@ts.umu.se.

| THE USE OF EFLIB REQUIRES ACCEPTANCE OF THE LICENSE AGREEMENT | ENCLOSED IN EVERY DISTRIBUTABLE PACKAGE. REFER TO LICENSE.TXT | OR THE EFLIB WEB SITE FOR FURTHER INFORATION. EFLIB IS PROTECTED | BY SWEDISH COPYRIGHT LAW AND INTERNATIONAL TREARTY PROVISIONS. | YOU AGREE THAT YOUR USE OF EFLIB IS SUBJECT TO THESE LAWS, | WHICH PROHIBIT UNAUTHORISED COPYING OR DUPLICATION OF EFLIB. | | THIS HEADER FILE MAY NEVER BE REMOVED. YOU ARE PERMITTED TO | ADD INFORMATION UNDER "REVISION CHANGES", "AUTHORS" AND | "CONTENTS".

See also: http://ftp.sunet.se/EFLIB/ http://ftp.sunet.se/EFLIB/Legal/License/

Types: *pFile tFile pIFile tIFile pOFile tOFile pTemporaryFile tTemporaryFile*

EFFILTER

Unit EFFILTER (in EFLIB.TEX help file)

EFLIB | Extended Function Library (C) Johan Larsson, 1992 - 1998 All rights reserved. E-mail to jola@ts.umu.se.

RELEASE 3 PROTOTYPE 5.1 [Last updated 970630]

Contents: FILTER CLASSES. This unit contains classes that filter information in streams: buffer filter, text filter, duplicator filter, encryption, validation codes (CRC16 and CRC32) and compression.

PROTOTYPE 6 WILL PROVIDE AN ENTIRELY REDESIGNED STREAM AND FILTER ENGINE. STREAMS ARE SPLITTED INTO TWO CLASSES: INPUT STREAMS AND OUTPUT STREAMS. NO BACKWARD COMPATIBILITY IS PLANNED.

Revision changes: 970630 [J]: Prototype 5.1 release.

Authors: [J] Johan Larsson, author of EFLIB. http://www.ts.umu.se/~jola/, jola@ts.umu.se.

See also: http://ftp.sunet.se/EFLIB/ http://ftp.sunet.se/EFLIB/Legal/License/

Constants: *CRC32Table*

Types: *pFilter tFilter pBufferFilter tBufferFilter pTextFilter tTextFilter pDuplicateFilter tDuplicateFilter pEncryptFilter tEncryptFilter pSequentialFilter tSequentialFilter pCRC16Filter tCRC16Filter pCRC32Filter tCRC32Filter pCompressFilter tCompressFilter*

EFGUI

Unit EFGUI (in EFLIB.TEX help file)

EFLIB | Extended Function Library (C) Johan Larsson, 1992 - 1998 All rights reserved. E-mail to jola@ts.umu.se.

RELEASE 3 PROTOTYPE 5.1 [Last updated 970630]

Contents: GRAPHICAL USER INTERFACE. This unit contains the base classes that makes EFLIBs user interface, for example tFonts and tColors.

NOT AVAILABLE. CROSS-PLATFORM GUI PLANNED FOR PROTOTYPE 6, SCHEDULED FOR RELEASE IN APRIL 30, 1998.

Revision changes: 970630 [J]: Prototype 5.1 release.

Authors: [J] Johan Larsson, author of EFLIB. http://www.ts.umu.se/~jola/, jola@ts.umu.se.

| THE USE OF EFLIB REQUIRES ACCEPTANCE OF THE LICENSE AGREEMENT | ENCLOSED IN EVERY DISTRIBUTABLE PACKAGE. REFER TO LICENSE.TXT | OR THE EFLIB WEB SITE FOR FURTHER INFORATION. EFLIB IS PROTECTED | BY SWEDISH COPYRIGHT LAW AND INTERNATIONAL TREARTY PROVISIONS. | YOU AGREE THAT YOUR USE OF EFLIB IS SUBJECT TO THESE LAWS, | WHICH PROHIBIT UNAUTHORISED COPYING OR DUPLICATION OF EFLIB. | | THIS HEADER FILE MAY NEVER BE REMOVED. YOU ARE PERMITTED TO | ADD INFORMATION UNDER "REVISION CHANGES", "AUTHORS" AND | "CONTENTS".

See also: http://ftp.sunet.se/EFLIB/ http://ftp.sunet.se/EFLIB/Legal/License/

EFIO

Unit EFIO (in EFLIB.TEX help file)

EFLIB | Extended Function Library (C) Johan Larsson, 1992 - 1998 All rights reserved. E-mail to jola@ts.umu.se.

RELEASE 3 PROTOTYPE 5.1 [Last updated 970605]

Contents: FILE AND DIRECTORY HANDLING. This unit contains classes that manages files and directories.

THESE CLASSES WILL REPLACED WITH OPERATING SYSTEM PORTABLE CLASSES IN

PROTOTYPE 6. THESE CLASSES ARE EXCLUSIVELY WRITTEN FOR THE BP7 STANDARD.

Revision changes: 970605 [J]: Prototype 5.1 release.

Authors: [J] Johan Larsson, author of EFLIB. http://www.ts.umu.se/~jola/, jola@ts.umu.se.

| THE USE OF EFLIB REQUIRES ACCEPTANCE OF THE LICENSE AGREEMENT | ENCLOSED IN EVERY DISTRIBUTABLE PACKAGE. REFER TO LICENSE.TXT | OR THE EFLIB WEB SITE FOR FURTHER INFORATION. EFLIB IS PROTECTED | BY SWEDISH COPYRIGHT LAW AND INTERNATIONAL TREARTY PROVISIONS. | YOU AGREE THAT YOUR USE OF EFLIB IS SUBJECT TO THESE LAWS, | WHICH PROHIBIT UNAUTHORISED COPYING OR DUPLICATION OF EFLIB. | | THIS HEADER FILE MAY NEVER BE REMOVED. YOU ARE PERMITTED TO | ADD INFORMATION UNDER "REVISION CHANGES", "AUTHORS" AND | "CONTENTS".

See also: http://ftp.sunet.se/EFLIB/ http://ftp.sunet.se/EFLIB/Legal/License/

Types: *pDirectory tDirectory pFilename tFilename pPath tPath*

EFKERNEL

Unit EFKERNEL (in EFLIB.TEX help file)

EFLIB | Extended Function Library (C) Johan Larsson, 1992 - 1998 All rights reserved. E-mail to jola@ts.umu.se.

RELEASE 3 PROTOTYPE 5.1 [Last updated 970628]

Contents: BASE CLASSES AND KERNEL COMPONENTS. This unit contains tObject, the abstract parent class and tStream, the stream engine, both vital for the library. EFKERNEL also contains the setup class and other kernel components including tClassManager (the class hierarchy).

EFKERNEL HAS BEEN REDESIGNED STARTING IN THE CURRENT WORKING MODEL OF PROTOTYPE 6. THE NEW DESIGN FOCUSES ON

DESIGN PATTERNS AND META CLASSES. SOME CLASSES ARE MOVED TO NEW UNITS (EFUTIL AND EFERROR), AND ABSTRACT (STUB) CLASSES AS PROVIDED FOR HIGHER EXTENSIBILITY. A TOOLKIT PATTERN IS USED TO CONFIGURE EFLIB PROTOTYPE 6 DURING RUN-TIME.

| THE NEW KERNEL WITH NOT BE COMPATIBLE WITH PROTOTYPE 5.1.

Revision changes: 970628 [J]: Prototype 5.1 release.

Authors: [J] Johan Larsson, author of EFLIB. http://www.ts.umu.se/~jola/, jola@ts.umu.se.

See also: http://ftp.sunet.se/EFLIB/ http://ftp.sunet.se/EFLIB/Legal/License/

Constants: *KernelInstalled Setup Classes ErrorHandler Interceptor*

Types: *pStream pClassIdentity pObject tObject pHandle tHandle tAccess tStream pContainer tContainer pCarrier tCarrier pClassManager tClassManager tClassIdentity pClassRegister tClassRegister pCarrierRegister tCarrierRegister pError pErrorHandler tErrorHandler tError pErrorMessage tErrorMessage pSetup tSetup*

Functions and Procedures: *InstallKernel RemoveKernel HeapErrorHandler RuntimeErrorHandler CompareVariable DosMajorVersion Assert Maximum Minimum IsInteger*

Unit EFKEYBRD (in EFLIB.TEX help file)

EFLIB | Extended Function Library (C) Johan Larsson, 1992 - 1998 All rights reserved. E-mail to jola@ts.umu.se.

RELEASE 3 PROTOTYPE 5.1 [Last updated 970626]

Contents: KEYBOARD DEVICE HANDLER. This unit contains scan codes for the keyboard and device handler object for they keyboard.

THESE CLASSES RELY ON MS-DOS OPERATING SYSTEM CALLS. A NEW DEVICE LAYER WILL BE INTRODUCED IN PROTOTYPE 6 FOR HIGHER PORTABILITY AND EXTENSIBILITY.

Revision changes: 970626 [J]: Prototype 5.1 release.

Authors: [J] Johan Larsson, author of EFLIB. http://www.ts.umu.se/~jola/, jola@ts.umu.se.

See also: http://ftp.sunet.se/EFLIB/ http://ftp.sunet.se/EFLIB/Legal/License/

Constants: *KBD_F1 KBD_ShiftF1 KBD_CtrlF1 KBD_AltF1 KBD_F2 KBD_ShiftF2 KBD_CtrlF2 KBD_AltF2 KBD_F3 KBD_ShiftF3 KBD_CtrlF3 KBD_AltF3 KBD_F4 KBD_ShiftF4 KBD_CtrlF4 KBD_AltF4 KBD_F5 KBD_ShiftF5 KBD_CtrlF5 KBD_AltF5 KBD_F6 KBD_ShiftF6 KBD_CtrlF6 KBD_AltF6 KBD_F7 KBD_ShiftF7 KBD_CtrlF7 KBD_AltF7 KBD_F8*

KBD_ShiftF8 KBD_CtrlF8 KBD_AltF8 KBD_F9 KBD_ShiftF9 KBD_CtrlF9 KBD_AltF9 KBD_F10 KBD_ShiftF10 KBD_CtrlF10 KBD_AltF10 KBD_F11 KBD_ShiftF11 KBD_CtrlF11 KBD_AltF11 KBD_F12 KBD_ShiftF12 KBD_CtrlF12 KBD_AltF12 KBD_Up KBD_ShiftUp KBD_CtrlUp KBD_AltUp KBD_Down KBD_ShiftDown KBD_CtrlDown KBD_AltDown KBD_Left KBD_ShiftLeft KBD_CtrlLeft KBD_AltLeft KBD_Right KBD_ShiftRight KBD_CtrlRight KBD_AltRight KBD_Home KBD_ShiftHome KBD_CtrlHome KBD_AltHome KBD_End KBD_Shiftend KBD_CtrlEnd KBD_Altend KBD_PgUp KBD_ShiftPgUp KBD_CtrlPgUp KBD_AltPgUp KBD_PgDn KBD_ShiftPgDn KBD_CtrlPgDn KBD_AltPgDn KBD_Ins KBD_ShiftIns KBD_CtrlIns KBD_AltIns KBD_Del KBD_ShiftDel KBD_CtrlDel KBD_AltDel KBD_Gray5 KBD_ShiftGray5 KBD_CtrlGray5 KBD_AltGray5 KBD_A KBD_ShiftA KBD_CtrlA KBD_AltA KBD_B KBD_ShiftB KBD_CtrlB KBD_AltB KBD_C KBD_ShiftC KBD_CtrlC KBD_AltC KBD_D KBD_ShiftD KBD_CtrlD KBD_AltD KBD_E KBD_ShiftE KBD_CtrlE KBD_AltE KBD_F KBD_ShiftF KBD_CtrlF KBD_AltF KBD_G KBD_ShiftG KBD_CtrlG KBD_AltG KBD_H KBD_ShiftH KBD_CtrlH KBD_AltH KBD_I KBD_ShiftI KBD_CtrlI KBD_AltI KBD_J KBD_ShiftJ KBD_CtrlJ KBD_AltJ KBD_K KBD_ShiftK KBD_CtrlK KBD_AltK KBD_L KBD_ShiftL KBD_CtrlL KBD_AltL KBD_M KBD_ShiftM KBD_CtrlM KBD_AltM KBD_N KBD_ShiftN KBD_CtrlN KBD_AltN KBD_O KBD_ShiftO KBD_CtrlO KBD_AltO KBD_P KBD_ShiftP KBD_CtrlP KBD_AltP KBD_Q KBD_ShiftQ KBD_CtrlQ KBD_AltQ KBD_R KBD_ShiftR KBD_CtrlR KBD_AltR KBD_S KBD_ShiftS KBD_CtrlS KBD_AltS KBD_T KBD_ShiftT KBD_CtrlT KBD_AltT KBD_U KBD_ShiftU KBD_CtrlU KBD_AltU KBD_V KBD_ShiftV KBD_CtrlV KBD_AltV KBD_W KBD_ShiftW KBD_CtrlW KBD_AltW KBD_X KBD_ShiftX KBD_CtrlX KBD_AltX KBD_Y KBD_ShiftY KBD_CtrlY KBD_AltY KBD_Z KBD_ShiftZ KBD_CtrlZ KBD_AltZ KBD_Num1 KBD_Alt1 KBD_Num2 KBD_Ctrl2 KBD_Alt2 KBD_Num3 KBD_Alt3 KBD_Num4 KBD_Alt4 KBD_Num5 KBD_Alt5 KBD_Num6 KBD_Ctrl6 KBD_Alt6 KBD_Num7 KBD_Alt7 KBD_Num8 KBD_Alt8 KBD_Num9 KBD_Alt9 KBD_Num0 KBD_Alt0 KBD_Space KBD_BkSp KBD_CtrlBkSp KBD_AltBkSp

KBD_TAB KBD_ShiftTAB KBD_CtrlTAB KBD_AltTAB KBD_Enter KBD_CtrlEnter KBD_AltEnter KBD_ESC KBD_AltESC KBD_Minus KBD_CtrlMinus KBD_AltMinus KBD_Plus KBD_AltPlus KBD_GrMin KBD_CtrlGrMin KBD_AltGrMin KBD_GrPls KBD_CtrlGrPls KBD_AltGrPls KBD_Star KBD_GrStar KBD_AltGrStar KeyboardBIOSSegment KeyboardInterrupt Keyboard

Types: *pKeyboard tKeyboard pKeyEvent tKeyEvent*

Variables: *KeyboardHead KeyboardTail KeyboardStatus absolute*

EFLIST

Unit EFLIST (in EFLIB.TEX help file)

EFLIB | Extended Function Library (C) Johan Larsson, 1992 - 1998 All rights reserved. E-mail to jola@ts.umu.se.

RELEASE 3 PROTOTYPE 5.1 [Last updated 970612]

Contents: ABSTRACT DATA TYPES with polymorphism. This unit contains list ADT′s: tList, tCursorList, tReversedList, tAdjustedList and tOrderedList. They all descend from tList, hence being linear data structures.

Revision changes: 970612 [J]: Prototype 5.1 release.

Authors: [J] Johan Larsson, author of EFLIB. http://www.ts.umu.se/~jola/, jola@ts.umu.se.

| THE USE OF EFLIB REQUIRES ACCEPTANCE OF THE LICENSE AGREEMENT | ENCLOSED IN EVERY DISTRIBUTABLE PACKAGE. REFER TO LICENSE.TXT | OR THE EFLIB WEB SITE FOR FURTHER INFORATION. EFLIB IS PROTECTED | BY SWEDISH COPYRIGHT LAW AND INTERNATIONAL TREARTY PROVISIONS. | YOU AGREE THAT YOUR USE OF EFLIB IS SUBJECT TO THESE LAWS, | WHICH PROHIBIT UNAUTHORISED COPYING OR DUPLICATION OF EFLIB. | | THIS HEADER FILE MAY NEVER BE REMOVED. YOU ARE PERMITTED TO | ADD INFORMATION UNDER ″REVISION CHANGES″, ″AUTHORS″ AND | ″CONTENTS″.

See also: http://ftp.sunet.se/EFLIB/
http://ftp.sunet.se/EFLIB/Legal/License/

Types: *pNodeIterator pLinkage tLinkage pList tList pCursorList tCursorList pReversedList tReversedList pAdjustedList tAdjustedList pOrderedList tOrderedList tNodeIterator*

EFMATH

Unit EFMATH (in EFLIB.TEX help file)

EFLIB | Extended Function Library (C) Johan Larsson, 1992 - 1998 All rights reserved. E-mail to jola@ts.umu.se.

RELEASE 3 PROTOTYPE 5.1 [Last updated 970616]

Contents: MATH ROUTINES. This unit contains mathematical routines for vector, matrix and complex algebra, numerical methods, standard arithmetics and expression parsing. THIS IS THE MOST EVOLVING PART OF EFLIB.

Future plans: PROTOTYPE 6 will implement a completely redesigned math unit. Most important is the new type-casting mechanisms, the modified class hierarchy with ADT element support, improved generic math arithmetics, validated vector and matrix operations, corrected polynomial class and SYMBOLIC DIFFERENTATION and EXPRESSION PARSING (see EFMEXP.PAS). Scheduled release: April 30, 1998.

Known problems: The tPolynomial class has known memory-critical problems. The tSolver class has not been tested since it is to be replaced in prototype 6.

Revision changes: 970616 [J]: Prototype 5.1 release.

Authors: [J] Johan Larsson, author of EFLIB. http://www.ts.umu.se/~jola/, jola@ts.umu.se.

| THE USE OF EFLIB REQUIRES ACCEPTANCE OF THE LICENSE AGREEMENT | ENCLOSED IN EVERY DISTRIBUTABLE PACKAGE. REFER TO LICENSE.TXT | OR THE EFLIB WEB SITE FOR FURTHER INFORATION. EFLIB IS PROTECTED | BY SWEDISH COPYRIGHT LAW AND INTERNATIONAL TREARTY PROVISIONS. | YOU AGREE THAT YOUR USE OF EFLIB IS SUBJECT

TO THESE LAWS, | WHICH PROHIBIT UNAUTHORISED COPYING OR DUPLICATION OF EFLIB. | | THIS HEADER FILE MAY NEVER BE REMOVED. YOU ARE PERMITTED TO | ADD INFORMATION UNDER "REVISION CHANGES", "AUTHORS" AND | "CONTENTS".

See also: http://ftp.sunet.se/EFLIB/ http://ftp.sunet.se/EFLIB/Legal/License/

Constants: *Caster CIm COne Primes Pi Infinity MathematicalTokens*

Types: *tBaseReal pMathObject tMathObject pCaster tCaster pInteger tInteger pRational tRational pReal tReal pVector tVector pComplex tComplex pMatrix tMatrix pPolynomial tPolynomial pMathFunction tMathFunction pExternalF tExternalF pExternalH tExternalH pSolver tSolver*

Functions and Procedures: *Power Log Tan Sinh Cosh Tanh Arcsin Arccos ArctanXY Sgn GCD Ceiling Floor Factorial Combinations IsPrime NextPrime*

EFMEMORY

Unit EFMEMORY (in EFLIB.TEX help file)

RELEASE 3 BUILD 1 [Last updated 970701]

MEMORY ENGINE. This unit handles the system memory. It contains classes for dynamic memory allocation (tAllocation).

Types: *pAllocation tAllocation pBuffer tBuffer pMemoryStream tMemoryStream pStreamHandle tStreamHandle*

Functions and Procedures: *FillWord*

Unit EFMEXP (in EFLIB.TEX help file)

EFLIB | Extended Function Library (C) Johan Larsson, 1992 - 1998 All rights reserved. E-mail to jola@ts.umu.se.

RELEASE 3 PROTOTYPE 5.1 [Last updated 970616]

Contents: MATH EXPRESSION HANDLING. This unit provides classes for handling of mathematical expressions. Expressions are converted to reversed polish notation (RPN), which also is the way they are stored in tExpression. Expressions can be associated to a collection of variables. tEvaluator is responsible for evaluation of an expression, and also maintains a table with operators.

NOT AVAILABLE IN THIS RELEASE. INTERFACE NOT SETTLED.

Revision changes: 970616 [J]: Prototype 5.1 release.

Authors: [J] Johan Larsson, author of EFLIB. http://www.ts.umu.se/~jola/, jola@ts.umu.se.

See also: http://ftp.sunet.se/EFLIB/ http://ftp.sunet.se/EFLIB/Legal/License/

Types: *pMathFunction tMathFunction pExpression tExpression pEvaluator tEvaluator pVariable tVariable pOperator tOperator*

Unit EFMOUSE (in EFLIB.TEX help file)

RELEASE 3 PROTOTYPE 5.1 [Last updated 970626]

Contents: MOUSE DEVICE. This unit contains objects that handles the mouse device. It require that a mouse driver is installed (such as MOUSE.COM) or Microsoft Windows is running.

THESE CLASSES RELY ON MS-DOS OPERATING SYSTEM CALLS. A NEW DEVICE LAYER WILL BE INTRODUCED IN PROTOTYPE 6 FOR HIGHER PORTABILITY AND EXTENSIBILITY.

Revision changes: 970626 [J]: Prototype 5.1 release.

Authors: [J] Johan Larsson, author of EFLIB. http://www.ts.umu.se/~jola/, jola@ts.umu.se.

See also: http://ftp.sunet.se/EFLIB/ http://ftp.sunet.se/EFLIB/Legal/License/

Constants: *MouseInterrupt MSE_LeftButton MSE_RightButton MSE_MiddleButton MSE_AnyButton MSE_SelectButton MSE_ConfirmButton Mouse*

Types: *pMouseDevice tMouseDevice pMouseEvent tMouseEvent*

Variables: *EventHappened EventX EventY EventButtons*

EFMSG

Unit EFMSG (in EFLIB.TEX help file)

EFLIB | Extended Function Library (C) Johan Larsson, 1992 - 1998 All rights reserved. E-mail to jola@ts.umu.se.

RELEASE 3 PROTOTYPE 5.1 [Last updated 970606]

Contents: MESSAGES. This unit contains the basic set of messages.

| NOT AVAILABLE.

Revision changes: 970606 [J]: Prototype 5.1 release.

Authors: [J] Johan Larsson, author of EFLIB. http://www.ts.umu.se/~jola/, jola@ts.umu.se.

| THE USE OF EFLIB REQUIRES ACCEPTANCE OF THE LICENSE AGREEMENT | ENCLOSED IN EVERY DISTRIBUTABLE PACKAGE. REFER TO LICENSE.TXT | OR THE EFLIB WEB SITE FOR FURTHER INFORATION. EFLIB IS PROTECTED | BY SWEDISH COPYRIGHT LAW AND INTERNATIONAL TREARTY PROVISIONS. | YOU AGREE THAT YOUR USE OF EFLIB IS SUBJECT TO THESE LAWS, | WHICH PROHIBIT UNAUTHORISED COPYING OR DUPLICATION OF EFLIB. | | THIS HEADER FILE MAY NEVER BE REMOVED. YOU ARE PERMITTED TO | ADD INFORMATION UNDER "REVISION CHANGES", "AUTHORS" AND | "CONTENTS".

See also: http://ftp.sunet.se/EFLIB/ http://ftp.sunet.se/EFLIB/Legal/License/

Types: *pFocusedEvent tFocusedEvent pLocalProcedure tLocalProcedure pLocalMessage tLocalMessage*

EFPATRN

Unit EFPATRN (in EFLIB.TEX help file)

EFLIB | Extended Function Library (C) Johan Larsson, 1992 - 1998 All rights reserved. E-mail to jola@ts.umu.se.

RELEASE 3 PROTOTYPE 5.1 [Last updated 970701]

Contents: PATTERN MATCHING. This unit provides some pattern matching mechanisms. tPattern and tPatternSensor defines pattern recognision system that can determine what class a certain pattern corresponds to.

| NOT AVAILABLE. PLANNED FOR PROTOTYPE 6; REQUIRED FOR EFMEXP.PAS.

Revision changes: 970701 [J]: Prototype 5.1 release.

Authors: [J] Johan Larsson, author of EFLIB. http://www.ts.umu.se/~jola/, jola@ts.umu.se.

See also: http://ftp.sunet.se/EFLIB/ http://ftp.sunet.se/EFLIB/Legal/License/

Types: *pPattern tPattern pPatternString tPatternString pPatternNumber tPatternNumber pWildcard tWildcard pPatternManager tPatternManager*

EFPLUG

Unit EFPLUG (in EFLIB.TEX help file)

RELEASE 3 PROTOTYPE 5.1 [Last updated 970627]

Contents: PLUG TECHNOLOGY. This unit defines the fundamental properties of all plug classes and the plug manager interface.

Future plans: These classes will be moved to the EFUTIL.PAS unit introduced in prototype 6.

Known problems: The tPolynomial class has known memory-critical problems. The tSolver class has not been tested since it is to be replaced in prototype 6.

Revision changes: 970627 [J]: Prototype 5.1 release.

Authors: [J] Johan Larsson, author of EFLIB. http://www.ts.umu.se/~jola/, jola@ts.umu.se.

See also: http://ftp.sunet.se/EFLIB/ http://ftp.sunet.se/EFLIB/Legal/License/

Types: *pPlug pPlugManager tPlugManager tPlug*

EFREG

Unit EFREG (in EFLIB.TEX help file)

RELEASE 3 PROTOTYPE 5.1 [Last updated 970627]

Contents: CLASS REGISTRATION TOOLS. This unit contains classes that manages (registers or maps) instances of other classes: tRegister and tMapper.

| NOT AVAILABLE.

Future plans: This unit will be replaced by EFUTIL and an extensive meta class engine in EFLIB prototype 6 planned for release in April 30, 1998.

Revision changes: 970627 [J]: Prototype 5.1 release.

Authors: [J] Johan Larsson, author of EFLIB. http://www.ts.umu.se/~jola/, jola@ts.umu.se.

| THE USE OF EFLIB REQUIRES ACCEPTANCE OF THE LICENSE AGREEMENT | ENCLOSED IN EVERY DISTRIBUTABLE PACKAGE. REFER TO LICENSE.TXT | OR THE EFLIB WEB SITE FOR FURTHER INFORATION. EFLIB IS PROTECTED | BY SWEDISH COPYRIGHT LAW AND INTERNATIONAL TREARTY PROVISIONS. | YOU AGREE THAT YOUR USE OF EFLIB IS SUBJECT TO THESE LAWS, | WHICH PROHIBIT UNAUTHORISED COPYING OR DUPLICATION OF EFLIB. | | THIS HEADER FILE MAY NEVER BE REMOVED. YOU ARE PERMITTED TO | ADD INFORMATION UNDER "REVISION CHANGES", "AUTHORS" AND | "CONTENTS".

See also: http://ftp.sunet.se/EFLIB/ http://ftp.sunet.se/EFLIB/Legal/License/

Types: *pHandle tHandle pRegister tRegister*

EFRES

Unit EFRES (in EFLIB.TEX help file)

EFLIB | Extended Function Library (C) Johan Larsson, 1992 - 1998 All rights reserved. E-mail to jola@ts.umu.se.

RELEASE 3 PROTOTYPE 5.1 [Last updated 970616]

Contents: RESOURCE HANDLING. This unit handles external resources with application components. An external resource is handled by tResource and maintains compressed components and their properties.

Future plans: PROTOTYPE 6 will depend more on external resources and persistent objects. Thus, this unit need to be reviewed and integrated with the new kernel design.

Known problems: Compression filter performance is not good enough.

Revision changes: 970616 [J]: Prototype 5.1 release.

Authors: [J] Johan Larsson, author of EFLIB. http://www.ts.umu.se/~jola/, jola@ts.umu.se.

| THE USE OF EFLIB REQUIRES ACCEPTANCE OF THE LICENSE AGREEMENT | ENCLOSED IN EVERY DISTRIBUTABLE PACKAGE. REFER TO LICENSE.TXT | OR THE EFLIB WEB SITE FOR FURTHER INFORATION. EFLIB IS PROTECTED | BY SWEDISH COPYRIGHT LAW AND INTERNATIONAL TREARTY PROVISIONS. | YOU AGREE THAT YOUR USE OF EFLIB IS SUBJECT TO THESE LAWS, | WHICH PROHIBIT UNAUTHORISED COPYING OR DUPLICATION OF EFLIB. | | THIS HEADER FILE MAY NEVER BE REMOVED. YOU ARE PERMITTED TO | ADD INFORMATION UNDER "REVISION CHANGES", "AUTHORS" AND | "CONTENTS".

See also: http://ftp.sunet.se/EFLIB/ http://ftp.sunet.se/EFLIB/Legal/License/

Types: *pResource tResource pStreamableFile tStreamableFile pArchive tArchive*

EFSCREEN

Unit EFSCREEN (in EFLIB.TEX help file)

EFLIB | Extended Function Library (C) Johan Larsson, 1992 - 1998 All rights reserved. E-mail to jola@ts.umu.se.

RELEASE 3 PROTOTYPE 5.1 [Last updated 970606]

Contents: SCREEN HANDLING. This unit defines the basic console drivers used by EFLIB, and supply a Console instance - the main output device.

Future plans: This unit will be replaced by the EFLIB cross-platform GUI classes.

Revision changes: 970606 [J]: Prototype 5.1 release.

Authors: [J] Johan Larsson, author of EFLIB. http://www.ts.umu.se/~jola/, jola@ts.umu.se.

See also: http://ftp.sunet.se/EFLIB/ http://ftp.sunet.se/EFLIB/Legal/License/

Types: *pScreen tScreen pBufferedScreen tBufferedScreen*

EFSEARCH

Unit EFSEARCH (in EFLIB.TEX help file)

EFLIB | Extended Function Library (C) Johan Larsson, 1992 - 1998 All rights reserved. E-mail to jola@ts.umu.se.

RELEASE 3 PROTOTYPE 5.1 [Last updated 970625]

Contents: SEARCH FILTERS. This unit contains the abstract class tMatchFilter that is the parent for tSearchFilter. tSearchFilter implements a Knuth-Morris-Pratt-search algorithm.

Revision changes: 970625 [J]: Prototype 5.1 release.

Authors: [J] Johan Larsson, author of EFLIB. http://www.ts.umu.se/~jola/, jola@ts.umu.se.

TO THESE LAWS, | WHICH PROHIBIT UNAUTHORISED COPYING OR DUPLICATION OF EFLIB. | | THIS HEADER FILE MAY NEVER BE REMOVED. YOU ARE PERMITTED TO | ADD INFORMATION UNDER "REVISION CHANGES", "AUTHORS" AND | "CONTENTS".

See also: http://ftp.sunet.se/EFLIB/ http://ftp.sunet.se/EFLIB/Legal/License/

Types: *pMatchFilter tMatchFilter pSearchFilter tSearchFilter pDelayedSearchFilter tDelayedSearchFilter pReplaceFilter tReplaceFilter*

EFSORT

Unit EFSORT (in EFLIB.TEX help file)

EFLIB | Extended Function Library (C) Johan Larsson, 1992 - 1998 All rights reserved. E-mail to jola@ts.umu.se.

RELEASE 3 PROTOTYPE 5.1 [Last updated 970627]

Contents: SORT PLUGS FOR DATA STRUCTURES. This unit provides a few additional sorting algorithms: Bubble Sort, Insertion Sort and Merge Sort.

Future plans: The revised ADT design in prototype 6 will require redesign of these classes as well.

Revision changes: 970616 [J]: Prototype 5.1 release.

Authors: [J] Johan Larsson, author of EFLIB. http://www.ts.umu.se/~jola/, jola@ts.umu.se.

| THE USE OF EFLIB REQUIRES ACCEPTANCE OF THE LICENSE AGREEMENT | ENCLOSED IN EVERY DISTRIBUTABLE PACKAGE. REFER TO LICENSE.TXT | OR THE EFLIB WEB SITE FOR FURTHER INFORATION. EFLIB IS PROTECTED | BY SWEDISH COPYRIGHT LAW AND INTERNATIONAL TREARTY PROVISIONS. | YOU AGREE THAT YOUR USE OF EFLIB IS SUBJECT TO THESE LAWS, | WHICH PROHIBIT UNAUTHORISED COPYING OR DUPLICATION OF EFLIB. | | THIS HEADER FILE MAY NEVER BE REMOVED. YOU ARE PERMITTED TO | ADD

INFORMATION UNDER "REVISION CHANGES", "AUTHORS" AND | "CONTENTS".

See also: http://ftp.sunet.se/EFLIB/ http://ftp.sunet.se/EFLIB/Legal/License/

Types: *pBubbleSortPlug tBubbleSortPlug pInsertionSortPlug tInsertionSortPlug pMergeSortPlug tMergeSortPlug*

EFSTREAM

Unit EFSTREAM (in EFLIB.TEX help file)

EFLIB | Extended Function Library (C) Johan Larsson, 1992 - 1998 All rights reserved. E-mail to jola@ts.umu.se.

RELEASE 3 PROTOTYPE 5.1 [Last updated 970623]

Contents: STREAM CLASSES. This unit contains classes that handle streamed input and output, among others tNullStream, tStandardStream, tDataStream and tFile.

PROTOTYPE 6 WILL PROVIDE AN ENTIRELY REDESIGNED STREAM AND FILTER ENGINE. STREAMS ARE SPLITTED INTO TWO CLASSES: INPUT STREAMS AND OUTPUT STREAMS. NO BACKWARD COMPATIBILITY IS PLANNED.

Revision changes: 970623 [J]: Prototype 5.1 release.

Authors: [J] Johan Larsson, author of EFLIB. http://www.ts.umu.se/~jola/, jola@ts.umu.se.

| THE USE OF EFLIB REQUIRES ACCEPTANCE OF THE LICENSE AGREEMENT | ENCLOSED IN EVERY DISTRIBUTABLE PACKAGE. REFER TO LICENSE.TXT | OR THE EFLIB WEB SITE FOR FURTHER INFORATION. EFLIB IS PROTECTED | BY SWEDISH COPYRIGHT LAW AND INTERNATIONAL TREARTY PROVISIONS. | YOU AGREE THAT YOUR USE OF EFLIB IS SUBJECT TO THESE LAWS, | WHICH PROHIBIT UNAUTHORISED COPYING OR DUPLICATION OF EFLIB. | | THIS HEADER FILE MAY NEVER BE REMOVED. YOU ARE PERMITTED TO | ADD INFORMATION UNDER "REVISION CHANGES", "AUTHORS" AND | "CONTENTS".

See also: http://ftp.sunet.se/EFLIB/
http://ftp.sunet.se/EFLIB/Legal/License/

Constants: *StdIO*

Types: *pNullStream tNullStream pStandardStream tStandardStream pDataStream tDataStream pFileStream tFileStream*

Functions and Procedures: *IsValidFile IsFile IsDirectory IsSomething ExtractFilename ExtractName ExtractExtention ExtractPath UniqueFilename DirectoryLevel DirectoryWithBackslash Path IsDiskReady LastDrive DeleteFile SizeOfFile SearchPath*

EFSTRING

Unit EFSTRING (in EFLIB.TEX help file)

EFLIB | Extended Function Library (C) Johan Larsson, 1992 - 1998 All rights reserved. E-mail to jola@ts.umu.se.

RELEASE 3 PROTOTYPE 5.1 [Last updated 970619]

Contents: STRING ROUTINES. This unit contains an extended string type, tString, and string routines.

Future plans: Integration with new kernel design. Abstract parent class.

Revision changes: 970619 [J]: Prototype 5.1 release. 970611 [J]: Endl instance added (C++ alike). 970601 [J]: Redesigned class hierarchy: token classes.

Authors: [J] Johan Larsson, author of EFLIB. http://www.ts.umu.se/~jola/, jola@ts.umu.se.

| THE USE OF EFLIB REQUIRES ACCEPTANCE OF THE LICENSE AGREEMENT | ENCLOSED IN EVERY DISTRIBUTABLE PACKAGE. REFER TO LICENSE.TXT | OR THE EFLIB WEB SITE FOR FURTHER INFORATION. EFLIB IS PROTECTED | BY SWEDISH COPYRIGHT LAW AND INTERNATIONAL TREARTY PROVISIONS. | YOU AGREE THAT YOUR USE OF EFLIB IS SUBJECT TO THESE LAWS, | WHICH PROHIBIT UNAUTHORISED COPYING OR DUPLICATION OF EFLIB. | | THIS HEADER FILE MAY NEVER BE

REMOVED. YOU ARE PERMITTED TO | ADD INFORMATION UNDER "REVISION CHANGES", "AUTHORS" AND | "CONTENTS".

See also: http://ftp.sunet.se/EFLIB/ http://ftp.sunet.se/EFLIB/Legal/License/

Constants: *Endl DefaultDelimeters UpperFilter LowerFilter*

Types: *pStringMatcher pString tString pToken tToken tStringMatcher pTranslation tTranslation*

Functions and Procedures: *BinaryStringNumber StringHexNumber StringPointer StringNumber StringValue StringIsInteger StringIsReal StringBoolean LowerCase UpperCase*

EFSTYLES

Unit EFSTYLES (in EFLIB.TEX help file)

EFLIB | Extended Function Library (C) Johan Larsson, 1992 - 1998 All rights reserved. E-mail to jola@ts.umu.se.

RELEASE 3 PROTOTYPE 5.1 [Last updated 970607]

Contents: STYLE CLASSES. This unit contains different style classes. A style is a set of properties related to visual components or screen output.

| NOT AVAILABLE. INTERFACE NOT SETTLED.

Revision changes: 970607 [J]: Prototype 5.1 release.

Authors: [J] Johan Larsson, author of EFLIB. http://www.ts.umu.se/~jola/, jola@ts.umu.se.

| THE USE OF EFLIB REQUIRES ACCEPTANCE OF THE LICENSE AGREEMENT | ENCLOSED IN EVERY DISTRIBUTABLE PACKAGE. REFER TO LICENSE.TXT | OR THE EFLIB WEB SITE FOR FURTHER INFORATION. EFLIB IS PROTECTED | BY SWEDISH COPYRIGHT LAW AND INTERNATIONAL TREARTY PROVISIONS. | YOU AGREE THAT YOUR USE OF EFLIB IS SUBJECT TO THESE LAWS, | WHICH PROHIBIT UNAUTHORISED COPYING OR DUPLICATION OF EFLIB. | | THIS HEADER FILE MAY NEVER BE REMOVED. YOU ARE PERMITTED TO | ADD

INFORMATION UNDER "REVISION CHANGES", "AUTHORS" AND | "CONTENTS".

See also: http://ftp.sunet.se/EFLIB/ http://ftp.sunet.se/EFLIB/Legal/License/

Constants: *cfLock cfProtection cfModal cfManager*

Types: *pStyle tStyle*

EFSYSTEM

Unit EFSYSTEM (in EFLIB.TEX help file)

EFLIB | Extended Function Library (C) Johan Larsson, 1992 - 1998 All rights reserved. E-mail to jola@ts.umu.se.

RELEASE 3 PROTOTYPE 5.1 [Last updated 970701]

Contents: SYSTEM ROUTINES. This unit contains the system profile (*Profile*), setting classes and the swap stream. The profile contains both user- defined settings that can be read or written to a INI-file when EFINIT is incorporated, and non-persistent status settings. The latter are run-time settings such as flags. They are specific for a certain execution.

| NOT AVAILABLE. INTERFACE NOT SETTLED.

Revision changes: 970701 [J]: Prototype 5.1 release.

Authors: [J] Johan Larsson, author of EFLIB. http://www.ts.umu.se/~jola/, jola@ts.umu.se.

| THE USE OF EFLIB REQUIRES ACCEPTANCE OF THE LICENSE AGREEMENT | ENCLOSED IN EVERY DISTRIBUTABLE PACKAGE. REFER TO LICENSE.TXT | OR THE EFLIB WEB SITE FOR FURTHER INFORATION. EFLIB IS PROTECTED | BY SWEDISH COPYRIGHT LAW AND INTERNATIONAL TREARTY PROVISIONS. | YOU AGREE THAT YOUR USE OF EFLIB IS SUBJECT TO THESE LAWS, | WHICH PROHIBIT UNAUTHORISED COPYING OR DUPLICATION OF EFLIB. | | THIS HEADER FILE MAY NEVER BE REMOVED. YOU ARE PERMITTED TO | ADD INFORMATION UNDER "REVISION CHANGES", "AUTHORS" AND | "CONTENTS".

See also: http://ftp.sunet.se/EFLIB/
http://ftp.sunet.se/EFLIB/Legal/License/

Constants: *Profile Environment Swap*

Types: *pNamedElement tNamedElement pMapping tMapping pNumberElement tNumberElement pLocalElement tLocalElement pFlag tFlag pSetting tSetting pProfile tProfile pEnvironment tEnvironment pSystemProfile tSystemProfile pClassSelector tClassSelector pComponentSelector tComponentSelector pSwapFilter tSwapFilter*

EFTABLE

Unit EFTABLE (in EFLIB.TEX help file)

RELEASE 3 PROTOTYPE 5.1 [Last updated 970627]

Contents: TABLE DATA STRUCTURES. This unit contains hash tables, that is ordered data structures that associates all elements to a virtual index value calculated from the element content.

Future plans: Hash tables will play a more important role in prototype 6 since system configurations (EFSYSTEM.PAS) will depend on them. The current interfaces and solutions are a bit sketchy and need to be refined. Possible backward compatibility.

Revision changes: 970627 [J]: Prototype 5.1 release.

Authors: [J] Johan Larsson, author of EFLIB. http://www.ts.umu.se/~jola/, jola@ts.umu.se.

See also: http://ftp.sunet.se/EFLIB/ http://ftp.sunet.se/EFLIB/Legal/License/

Types: *pHashPlug pSparceIterator pBucketIterator pHashTable tHashTable pOpenHashTable tOpenHashTable pProbeHashTable tProbeHashTable pDoubleHashTable tDoubleHashTable pBucket tBucket pBucketHashTable tBucketHashTable tBucketIterator tHashPlug pTextHashPlug tTextHashPlug*

EFTEST

Unit EFTEST (in EFLIB.TEX help file)

RELEASE 3 PROTOTYPE 5.1 [Last updated 970531]

Contents: TEST SUPPORT ROUTINES. This unit contains some functions for EFLIB testing and example programs.

Future plans: No changes planned for prototype 6 other than adjustments to the new element hierarchy (CreateRandomElement).

Revision changes: 970531 [J]: Prototype 5.1 release.

Authors: [J] Johan Larsson, author of EFLIB. http://www.ts.umu.se/~jola/, jola@ts.umu.se.

EFLIB. | | THIS HEADER FILE MAY NEVER BE REMOVED. YOU ARE PERMITTED TO | ADD INFORMATION UNDER "REVISION CHANGES", "AUTHORS" AND | "CONTENTS".

See also: http://ftp.sunet.se/EFLIB/ http://ftp.sunet.se/EFLIB/Legal/License/

Constants: *MemoryStatus*

Functions and Procedures: *CreateRandomString CreateRandomElement MemPush MemPop*

EFTEXT

Unit EFTEXT (in EFLIB.TEX help file)

EFLIB | Extended Function Library (C) Johan Larsson, 1992 - 1998 All rights reserved. E-mail to jola@ts.umu.se.

RELEASE 3 PROTOTYPE 5.1 [Last updated 970630]

Contents: TEXT HANDLING. This unit contains classes like tText and tParser, the general text interface and the text parser.

| INTERFACE NOT SETTLED.

Future plans: Classes in this unit will be redesigned and refined. The role of this unit will be very central in EFLIB since it generalizes streams and strings into multiple-row text. The text file association is hidden under a generalized abstract text resource class as today.

Revision changes: 970630 [J]: Prototype 5.1 release.

Authors: [J] Johan Larsson, author of EFLIB. http://www.ts.umu.se/~jola/, jola@ts.umu.se.

| THE USE OF EFLIB REQUIRES ACCEPTANCE OF THE LICENSE AGREEMENT | ENCLOSED IN EVERY DISTRIBUTABLE PACKAGE. REFER TO LICENSE.TXT | OR THE EFLIB WEB SITE FOR FURTHER INFORATION. EFLIB IS PROTECTED | BY SWEDISH COPYRIGHT LAW AND INTERNATIONAL TREARTY PROVISIONS. | YOU AGREE THAT YOUR USE OF EFLIB IS SUBJECT TO THESE LAWS, | WHICH PROHIBIT UNAUTHORISED COPYING OR DUPLICATION OF

EFLIB. | | THIS HEADER FILE MAY NEVER BE REMOVED. YOU ARE PERMITTED TO | ADD INFORMATION UNDER "REVISION CHANGES", "AUTHORS" AND | "CONTENTS".

See also: http://ftp.sunet.se/EFLIB/ http://ftp.sunet.se/EFLIB/Legal/License/

Constants: *Setup_TextBuffer*

Types: *pTextResource tTextResource pTextFile tTextFile pText tText pHyperText tHyperText pParser tParser*

EFTIME

Unit EFTIME (in EFLIB.TEX help file)

EFLIB | Extended Function Library (C) Johan Larsson, 1992 - 1998 All rights reserved. E-mail to jola@ts.umu.se.

RELEASE 3 PROTOTYPE 5.1 [Last updated 970630]

Contents: TIME AND DATE HANDLING. This unit contains tTiming with descendants. These classes provide ways of dealing with time and date. tTimer implements a timer device with millisecond precision.

| NOT AVAILABLE.

Future plans: This unit will replace EFBASIC date and time classes starting with prototype 6.

Revision changes: 970630 [J]: Prototype 5.1 release.

Authors: [J] Johan Larsson, author of EFLIB. http://www.ts.umu.se/~jola/, jola@ts.umu.se.

| THE USE OF EFLIB REQUIRES ACCEPTANCE OF THE LICENSE AGREEMENT | ENCLOSED IN EVERY DISTRIBUTABLE PACKAGE. REFER TO LICENSE.TXT | OR THE EFLIB WEB SITE FOR FURTHER INFORATION. EFLIB IS PROTECTED | BY SWEDISH COPYRIGHT LAW AND INTERNATIONAL TREARTY PROVISIONS. | YOU AGREE THAT YOUR USE OF EFLIB IS SUBJECT TO THESE LAWS, | WHICH PROHIBIT UNAUTHORISED COPYING OR DUPLICATION OF EFLIB. | | THIS HEADER FILE MAY NEVER BE REMOVED. YOU ARE PERMITTED TO | ADD

INFORMATION UNDER "REVISION CHANGES", "AUTHORS" AND | "CONTENTS".

See also: http://ftp.sunet.se/EFLIB/ http://ftp.sunet.se/EFLIB/Legal/License/

Types: *pTiming tTiming pTime tTime pDate tDate pTimeDate tTimeDate pTimer tTimer*

EFTREE

Unit EFTREE (in EFLIB.TEX help file)

EFLIB | Extended Function Library (C) Johan Larsson, 1992 - 1998 All rights reserved. E-mail to jola@ts.umu.se.

RELEASE 3 PROTOTYPE 5.1 [Last updated 970630]

Contents: TREE DATA STRUCTURES. Trees are ordered data structure where each node can connect to more than one immediate successor (children node). EFLIB provides you with a complete set of tree ADT's.

| FROZEN UNTIL ADT KERNEL SETTLED FOR PROTOTYPE 6.

Future plans: The tree ADT's are very important for EFLIB. They will probably descend from a tGraph class which is extended into tTree and further extended into specialized trees like binary trees and AVL trees.

Revision changes: 970630 [J]: Prototype 5.1 release.

Authors: [J] Johan Larsson, author of EFLIB. http://www.ts.umu.se/~jola/, jola@ts.umu.se.

| THE USE OF EFLIB REQUIRES ACCEPTANCE OF THE LICENSE AGREEMENT | ENCLOSED IN EVERY DISTRIBUTABLE PACKAGE. REFER TO LICENSE.TXT | OR THE EFLIB WEB SITE FOR FURTHER INFORATION. EFLIB IS PROTECTED | BY SWEDISH COPYRIGHT LAW AND INTERNATIONAL TREARTY PROVISIONS. | YOU AGREE THAT YOUR USE OF EFLIB IS SUBJECT TO THESE LAWS, | WHICH PROHIBIT UNAUTHORISED COPYING OR DUPLICATION OF EFLIB. | | THIS HEADER FILE MAY NEVER BE REMOVED. YOU ARE PERMITTED TO | ADD

INFORMATION UNDER "REVISION CHANGES", "AUTHORS" AND | "CONTENTS".

See also: http://ftp.sunet.se/EFLIB/ http://ftp.sunet.se/EFLIB/Legal/License/

Constants: *Setup_DefaultTreeOrder*

Types: *pTreeLinkage tTreeLinkage pAVLLinkage tAVLLinkage pTree tTree pBinaryTree tBinaryTree pAVLTree tAVLTree pSplayTree tSplayTree pTreeIterator tTreeIterator pPathPlug tPathPlug*

EFVIEWS

Unit EFVIEWS (in EFLIB.TEX help file)

RELEASE 3 PROTOTYPE 5.1 [Last updated 970605]

Contents: VIEWS. This unit contains the abstract view classes. All GUI components must descend from either tView or tGroup. The latter enables the component to own other components. An example on a view is the clickable button, on a group the window with header, border and close button.

| NOT AVAILABLE.

Future plans: The design of the GUI as introduced in prototype 6 may lead to changes in the view-group relation and the assocation to the component-manager architecture.

Revision changes: 970605 [J]: Prototype 5.1 release.

Authors: [J] Johan Larsson, author of EFLIB. http://www.ts.umu.se/~jola/, jola@ts.umu.se.

TO THESE LAWS, | WHICH PROHIBIT UNAUTHORISED COPYING OR DUPLICATION OF EFLIB. | | THIS HEADER FILE MAY NEVER BE REMOVED. YOU ARE PERMITTED TO | ADD INFORMATION UNDER "REVISION CHANGES", "AUTHORS" AND | "CONTENTS".

See also: http://ftp.sunet.se/EFLIB/ http://ftp.sunet.se/EFLIB/Legal/License/

Constants: *Black Brown LightRed Blue LightGray LightMagenta Green DarkGray Yellow Cyan LightBlue White Red LightGreen Magenta LightCyan Blink*

Types: *pView tView pArrangement tArrangement pGroup tGroup tOldView*

Endl — EFSTRING

Const EFSTRING.Endl: pString = NIL;

This is a string with a line feed sequence (CR/LF). Endl can be inserted in other *tString* instances.

Environment — EFSYSTEM

Const EFSYSTEM.Environment: pEnvironment = NIL;

This is a shortcut address to the "Environment" profile in the *Profile* instance in EFSETUP. Use this instance when want to access the environment settings.

EOL — EFDEF

Const EFDEF.EOL: array = #13 + #10;

End of line sequence used by text streams.

ErrorHandler EFKERNEL

Const EFKERNEL.ErrorHandler: pErrorHandler = NIL;

Error manager: this is a pointer to a dynamic instance of *tErrorHandler*. The error handler handles program errors.

Error_Abstract EFDEF

Const EFDEF.Error_Abstract = 003;

Call to an abstract method (fatal).

Error_Access EFDEF

Const EFDEF.Error_Access = 101;

Forbidden memory access.

Error_Address EFDEF

Const EFDEF.Error_Address = 102;

Wrong memory address.

Error_ADT_Capacity EFDEF

Const EFDEF.Error_ADT_Capacity = 304;

Out of capacity (fatal).

Error_ADT_Compatibility EFDEF

Const EFDEF.Error_ADT_Compatibility = 301;

Invalid operation due to ADT incompatibility.

Error_ADT_InvalidElement EFDEF

Const EFDEF.Error_ADT_InvalidElement = 302;

Element invalid.

Error_ADT_ReadOnly EFDEF

Const EFDEF.Error_ADT_ReadOnly = 303;

Write access not permitted.

Error_Allocation EFDEF

Const EFDEF.Error_Allocation = 006;

Instance not allocated (fatal).

Error_Assertion EFDEF

Const EFDEF.Error_Assertion = 005;

Could not assert a condition (fatal).

Error_Compatibility EFDEF

Const EFDEF.Error_Compatibility = 009;

Incompatible components (fatal).

Error_CopyIn EFDEF

Const EFDEF.Error_CopyIn = 103;

CopyIn operation failed.

Error_CopyOut EFDEF

Const EFDEF.Error_CopyOut = 104;

CopyOut operation failed.

Error_Damaged EFDEF

Const EFDEF.Error_Damaged = 008;

Component is damaged (fatal).

Error_Extension EFDEF

Const EFDEF.Error_Extension = 012;

Extension failure.

Error_File EFDEF

Const EFDEF.Error_File = 203;

File operation failed (fatal).

Error_FileNotFound EFDEF

Const EFDEF.Error_FileNotFound = 201;

File not found (fatal).

Error_Filter_Base EFDEF

Const EFDEF.Error_Filter_Base = 214;

Base stream not ready.

Error_Forbidden EFDEF

Const EFDEF.Error_Forbidden = 004;

Forbidden operation requested (fatal).

Error_General EFDEF

Const EFDEF.Error_General = 002;

General run-time error (fatal).

Error_GUI_Boundaries EFDEF

Const EFDEF.Error_GUI_Boundaries = 402;

Invalid boundaries; outside limits.

Error_GUI_Forbidden EFDEF

Const EFDEF.Error_GUI_Forbidden = 401;

Forbidden GUI operation requested.

Error_GUI_ForbiddenComponent EFDEF

Const EFDEF.Error_GUI_ForbiddenComponent = 404;

Forbidden access to GUI component requested.

Error_GUI_InvalidComponent EFDEF

Const EFDEF.Error_GUI_InvalidComponent = 403;

Invalid GUI component.

Error_GUI_Memory EFDEF

Const EFDEF.Error_GUI_Memory = 405;

Insufficient memory for GUI operation.

Error_Identity EFDEF

Const EFDEF.Error_Identity = 014;

Identity not registered or damaged database.

Error_InvalidFilename EFDEF

Const EFDEF.Error_InvalidFilename = 202;

Invalid filename (fatal).

Error_InvalidIndex EFDEF

Const EFDEF.Error_InvalidIndex = 040;

Invalid index value or reference.

Error_Manager EFDEF

Const EFDEF.Error_Manager = 021;

Manager failure (fatal).

Error_Math_Arithmetics EFDEF

Const EFDEF.Error_Math_Arithmetics = 801;

Arithmetic failure; undefined operation.

Error_Math_Syntax EFDEF

Const EFDEF.Error_Math_Syntax = 802;

Invalid expression syntax.

Error_Math_Typecast EFDEF

Const EFDEF.Error_Math_Typecast = 803;

Type-casting not permitted.

Error_Message EFDEF

Const EFDEF.Error_Message = 020;

Unrecognized message or invalid reciever.

Error_OutOfMemory EFDEF

Const EFDEF.Error_OutOfMemory = 105;

Out of memory (fatal).

Error_PlugNotFound EFDEF

Const EFDEF.Error_PlugNotFound = 011;

Plug not found or disabled (fatal).

Error_Registration EFDEF

Const EFDEF.Error_Registration = 013;

Registration failure or ambiguous identities (fatal).

Error_Resource EFDEF

Const EFDEF.Error_Resource = 007;

Invalid resource instance (fatal).

Error_Selector EFDEF

Const EFDEF.Error_Selector = 010;

Component selector could not select a class (fatal).

Error_Stream EFDEF

Const EFDEF.Error_Stream = 030;

Fatal stream access failure.

Error_StreamStorage EFDEF

Const EFDEF.Error_StreamStorage = 032;

Stream storage not supported (fatal).

Error_Stream_Buffer EFDEF

Const EFDEF.Error_Stream_Buffer = 213;

Buffer not operational.

Error_Stream_Mode EFDEF

Const EFDEF.Error_Stream_Mode = 210;

Access not permitted: invalid mode.

Error_Stream_Read EFDEF

Const EFDEF.Error_Stream_Read = 211;

Stream read error.

Error_Stream_Write EFDEF

Const EFDEF.Error_Stream_Write = 212;

Stream write error.

Error_VMT EFDEF

Const EFDEF.Error_VMT = 001;

Object not initialized (VMT failure, fatal).

EventButtons EFMOUSE

See EventX

EventHappened EFMOUSE

EFMOUSE.EventHappened: boolean;

Mouse event handler flag (TRUE if event happened)

EventX EFMOUSE

EFMOUSE.EventX,

EventY,

EventButtons: word;

Event handler status variables.

EventY EFMOUSE

See EventX

ExtractExtention EFSTREAM

---

function EFSTREAM.ExtractExtention(WhatFullname: string): string;

Returns the three letter extention of a file.

ExtractFilename EFSTREAM

---

function EFSTREAM.ExtractFilename(WhatFullname: string): string;

Returns a filename without path prefix.

ExtractName EFSTREAM

---

function EFSTREAM.ExtractName(WhatFullname: string): string;

Returns the name and path of a file, but without extention.

ExtractPath EFSTREAM

---

function EFSTREAM.ExtractPath(WhatFullname: string): string;

Returns the path prefix for a full filename.

Factorial EFMATH

---

function EFMATH.Factorial(Number: word): tBaseReal;

Returns a factorial number n! there n is a positive integer. 0! is by definition equal to 1.

FilenameChars EFDEF

Const EFDEF.FilenameChars: tChars = ['!', '#'..')', '-', '.', '0'..'9', '@'..'Z', '^'..'{', '{'..#255] ;

Flename character set: the valid characters for MS-DOS filenames.

FillWord EFMEMORY

---

procedure EFMEMORY.FillWord(var Destination; Count, Data: word);

Fills a variable with word-sized data. The contents of the specified Data is copied Count times to Destination.

Floor EFMATH

---

function EFMATH.Floor(X: tBaseReal): tBaseReal;

Returns the next lowest integer, ie. the closest integer smaller than or equal to the specified real number.

GCD EFMATH

---

function EFMATH.GCD(X, Y: longint): longint;

Returns the greatest common divisor (GCD) of two arbitrary integers.

Green EFVIEWS

Const EFVIEWS.Green = 2;

HeapErrorHandler EFKERNEL

function EFKERNEL.HeapErrorHandler(Size: word): integer;

Heap error handler that gets called whenever the heap manager cannot complete an allocation request.

Infinity EFMATH

Const EFMATH.Infinity: tBaseReal = 9.9999999999E+37;

InstallKernel EFKERNEL

procedure EFKERNEL.InstallKernel;

Attempts to install the kernel engine. If the kernel engine can't be installed, the program have to terminate. If the kernel engine has already been installed, nothing will be done.

Interceptor EFKERNEL

Const EFKERNEL.Interceptor: pClassRegister = NIL;

This is the component interceptor. It destructs all instances that have been registered (using the tClassQueue.Put) before the program terminates.

IsDirectory EFSTREAM

function EFSTREAM.IsDirectory(What: string): boolean;

Returns TRUE if directory exists and is accessable.

IsDiskReady — EFSTREAM

function EFSTREAM.IsDiskReady(Drive: byte): boolean;

Returnes TRUE if drive (0..255; current, A, B, C, etc.) is ready for access.

IsFile — EFSTREAM

function EFSTREAM.IsFile(What: string): boolean;

Returns TRUE if file exists and is accessable.

IsInteger — EFKERNEL

function EFKERNEL.IsInteger(RealVariable: real): boolean;

Returns TRUE if the specified variable is an integer (ie. doesn't contain any decimals).

IsPrime — EFMATH

function EFMATH.IsPrime(What: longint): boolean;

Returns TRUE if the specified integer is a prime number. Uses an optimized algorithm.

IsSomething — EFSTREAM

function EFSTREAM.IsSomething(What: string): boolean;

Returns TRUE if specified filename corresponds to an existing file or a valid directory.

IsValidFile EFSTREAM

---

function EFSTREAM.IsValidFile(What: string): boolean;

Returns TRUE if the specified string is valid filename, ie. a possible name of a valid file. If the string contains a path, the path must lead to an existing directory.

KBD_A EFKEYBRD

Const EFKEYBRD.KBD_A = $1E41;

KBD_Alt0 EFKEYBRD

Const EFKEYBRD.KBD_Alt0 = $8100;

KBD_Alt1 EFKEYBRD

Const EFKEYBRD.KBD_Alt1 = $7800;

KBD_Alt2 EFKEYBRD

Const EFKEYBRD.KBD_Alt2 = $7900;

KBD_Alt3 EFKEYBRD

Const EFKEYBRD.KBD_Alt3 = $7A00;

KBD_Alt4 EFKEYBRD

Const EFKEYBRD.KBD_Alt4 = $7B00;

KBD_Alt5 EFKEYBRD

Const EFKEYBRD.KBD_Alt5 = $7C00;

KBD_Alt6 EFKEYBRD

Const EFKEYBRD.KBD_Alt6 = $7D00;

KBD_Alt7 EFKEYBRD

Const EFKEYBRD.KBD_Alt7 = $7E00;

KBD_Alt8 EFKEYBRD

Const EFKEYBRD.KBD_Alt8 = $7F00;

KBD_Alt9 EFKEYBRD

Const EFKEYBRD.KBD_Alt9 = $8000;

KBD_AltA EFKEYBRD

Const EFKEYBRD.KBD_AltA = $1E00;

KBD_AltB EFKEYBRD

Const EFKEYBRD.KBD_AltB = $3000;

KBD_AltBkSp EFKEYBRD

Const EFKEYBRD.KBD_AltBkSp = $0E00;

KBD_AltC EFKEYBRD

Const EFKEYBRD.KBD_AltC = $2E00;

KBD_AltD EFKEYBRD

Const EFKEYBRD.KBD_AltD = $2000;

KBD_AltDel EFKEYBRD

Const EFKEYBRD.KBD_AltDel = $A300;

KBD_AltDown EFKEYBRD

Const EFKEYBRD.KBD_AltDown = $A000;

KBD_AltE EFKEYBRD

Const EFKEYBRD.KBD_AltE = $1200;

KBD_Altend EFKEYBRD

Const EFKEYBRD.KBD_Altend = $9F00;

KBD_AltEnter EFKEYBRD

Const EFKEYBRD.KBD_AltEnter = $1C00;

KBD_AltESC EFKEYBRD

Const EFKEYBRD.KBD_AltESC = $0100;

KBD_AltF EFKEYBRD

Const EFKEYBRD.KBD_AltF = $2100;

KBD_AltF1 EFKEYBRD

Const EFKEYBRD.KBD_AltF1 = $6800;

KBD_AltF10 EFKEYBRD

Const EFKEYBRD.KBD_AltF10 = $7100;

KBD_AltF11 EFKEYBRD

Const EFKEYBRD.KBD_AltF11 = $8B00;

KBD_AltF12 EFKEYBRD

Const EFKEYBRD.KBD_AltF12 = $8C00;

KBD_AltF2 EFKEYBRD

Const EFKEYBRD.KBD_AltF2 = $6900;

KBD_AltF3 EFKEYBRD

Const EFKEYBRD.KBD_AltF3 = $6A00;

KBD_AltF4 EFKEYBRD

Const EFKEYBRD.KBD_AltF4 = $6B00;

KBD_AltF5 EFKEYBRD

Const EFKEYBRD.KBD_AltF5 = $6C00;

KBD_AltF6 EFKEYBRD

Const EFKEYBRD.KBD_AltF6 = $6D00;

KBD_AltF7 EFKEYBRD

Const EFKEYBRD.KBD_AltF7 = $6E00;

KBD_AltF8 EFKEYBRD

Const EFKEYBRD.KBD_AltF8 = $6F00;

KBD_AltF9 EFKEYBRD

Const EFKEYBRD.KBD_AltF9 = $7000;

KBD_AltG EFKEYBRD

Const EFKEYBRD.KBD_AltG = $2200;

KBD_AltGray5 EFKEYBRD

Const EFKEYBRD.KBD_AltGray5 = $9C00;

KBD_AltGrMin EFKEYBRD

Const EFKEYBRD.KBD_AltGrMin = $4A00;

KBD_AltGrPls EFKEYBRD

Const EFKEYBRD.KBD_AltGrPls = $4E00;

KBD_AltGrStar EFKEYBRD

Const EFKEYBRD.KBD_AltGrStar = $3700;

Miscellaneous keyboard constants *
****

KBD_AltH EFKEYBRD

Const EFKEYBRD.KBD_AltH = $2300;

KBD_AltHome EFKEYBRD

Const EFKEYBRD.KBD_AltHome = $9700;

KBD_AltI EFKEYBRD

Const EFKEYBRD.KBD_AltI = $1700;

KBD_AltIns EFKEYBRD

Const EFKEYBRD.KBD_AltIns = $A200;

KBD_AltJ EFKEYBRD

Const EFKEYBRD.KBD_AltJ = $2400;

KBD_AltK EFKEYBRD

Const EFKEYBRD.KBD_AltK = $2500;

KBD_AltL EFKEYBRD

Const EFKEYBRD.KBD_AltL = $2600;

KBD_AltLeft EFKEYBRD

Const EFKEYBRD.KBD_AltLeft = $9B00;

KBD_AltM EFKEYBRD

Const EFKEYBRD.KBD_AltM = $3200;

KBD_AltMinus EFKEYBRD

Const EFKEYBRD.KBD_AltMinus = $8200;

KBD_AltN EFKEYBRD

Const EFKEYBRD.KBD_AltN = $3100;

KBD_AltO EFKEYBRD

Const EFKEYBRD.KBD_AltO = $1800;

KBD_AltP EFKEYBRD

Const EFKEYBRD.KBD_AltP = $1900;

KBD_AltPgDn EFKEYBRD

Const EFKEYBRD.KBD_AltPgDn = $A100;

KBD_AltPgUp EFKEYBRD

Const EFKEYBRD.KBD_AltPgUp = $9900;

KBD_AltPlus EFKEYBRD

Const EFKEYBRD.KBD_AltPlus = $8300;

KBD_AltQ EFKEYBRD

Const EFKEYBRD.KBD_AltQ = $1000;

KBD_AltR EFKEYBRD

Const EFKEYBRD.KBD_AltR = $1300;

KBD_AltRight EFKEYBRD

Const EFKEYBRD.KBD_AltRight = $9D00;

KBD_AltS EFKEYBRD

Const EFKEYBRD.KBD_AltS = $1F00;

KBD_AltT EFKEYBRD

Const EFKEYBRD.KBD_AltT = $1400;

KBD_AltTAB EFKEYBRD

Const EFKEYBRD.KBD_AltTAB = $A500;

KBD_AltU EFKEYBRD

Const EFKEYBRD.KBD_AltU = $1600;

KBD_AltUp EFKEYBRD

Const EFKEYBRD.KBD_AltUp = $9800;

KBD_AltV EFKEYBRD

Const EFKEYBRD.KBD_AltV = $2F00;

KBD_AltW EFKEYBRD

Const EFKEYBRD.KBD_AltW = $1100;

KBD_AltX EFKEYBRD

Const EFKEYBRD.KBD_AltX = $2D00;

KBD_AltY EFKEYBRD

Const EFKEYBRD.KBD_AltY = $1500;

KBD_AltZ EFKEYBRD

Const EFKEYBRD.KBD_AltZ = $2C00;

KBD_B EFKEYBRD

Const EFKEYBRD.KBD_B = $3042;

KBD_BkSp EFKEYBRD

Const EFKEYBRD.KBD_BkSp = $0E08;

KBD_C EFKEYBRD

Const EFKEYBRD.KBD_C = $2E43;

KBD_Ctrl2 EFKEYBRD

Const EFKEYBRD.KBD_Ctrl2 = $0300;

KBD_Ctrl6 EFKEYBRD

Const EFKEYBRD.KBD_Ctrl6 = $071E;

KBD_CtrlA EFKEYBRD

Const EFKEYBRD.KBD_CtrlA = $1E01;

KBD_CtrlB EFKEYBRD

Const EFKEYBRD.KBD_CtrlB = $3002;

KBD_CtrlBkSp EFKEYBRD

Const EFKEYBRD.KBD_CtrlBkSp = $0E7F;

KBD_CtrlC EFKEYBRD

Const EFKEYBRD.KBD_CtrlC = $2E03;

KBD_CtrlD EFKEYBRD

Const EFKEYBRD.KBD_CtrlD = $2004;

KBD_CtrlDel EFKEYBRD

Const EFKEYBRD.KBD_CtrlDel = $9300;

KBD_CtrlDown EFKEYBRD

Const EFKEYBRD.KBD_CtrlDown = $9100;

KBD_CtrlE EFKEYBRD

Const EFKEYBRD.KBD_CtrlE = $1205;

KBD_CtrlEnd EFKEYBRD

Const EFKEYBRD.KBD_CtrlEnd = $7500;

KBD_CtrlEnter EFKEYBRD

Const EFKEYBRD.KBD_CtrlEnter = $1C0A;

KBD_CtrlF EFKEYBRD

Const EFKEYBRD.KBD_CtrlF = $2106;

KBD_CtrlF1 EFKEYBRD

Const EFKEYBRD.KBD_CtrlF1 = $5E00;

KBD_CtrlF10 EFKEYBRD

Const EFKEYBRD.KBD_CtrlF10 = $6700;

KBD_CtrlF11 EFKEYBRD

Const EFKEYBRD.KBD_CtrlF11 = $8900;

KBD_CtrlF12 EFKEYBRD

Const EFKEYBRD.KBD_CtrlF12 = $8A00;

KBD_CtrlF2 EFKEYBRD

Const EFKEYBRD.KBD_CtrlF2 = $5F00;

KBD_CtrlF3 EFKEYBRD

Const EFKEYBRD.KBD_CtrlF3 = $6000;

KBD_CtrlF4 EFKEYBRD

Const EFKEYBRD.KBD_CtrlF4 = $6100;

KBD_CtrlF5 EFKEYBRD

Const EFKEYBRD.KBD_CtrlF5 = $6200;

KBD_CtrlF6 EFKEYBRD

Const EFKEYBRD.KBD_CtrlF6 = $6300;

KBD_CtrlF7 EFKEYBRD

Const EFKEYBRD.KBD_CtrlF7 = $6400;

KBD_CtrlF8 EFKEYBRD

Const EFKEYBRD.KBD_CtrlF8 = $6500;

KBD_CtrlF9 EFKEYBRD

Const EFKEYBRD.KBD_CtrlF9 = $6600;

KBD_CtrlG EFKEYBRD

Const EFKEYBRD.KBD_CtrlG = $2207;

KBD_CtrlGray5 EFKEYBRD

Const EFKEYBRD.KBD_CtrlGray5 = $8F00;

KBD_CtrlGrMin EFKEYBRD

Const EFKEYBRD.KBD_CtrlGrMin = $8E00;

KBD_CtrlGrPls EFKEYBRD

Const EFKEYBRD.KBD_CtrlGrPls = $9000;

KBD_CtrlH EFKEYBRD

Const EFKEYBRD.KBD_CtrlH = $2308;

KBD_CtrlHome EFKEYBRD

Const EFKEYBRD.KBD_CtrlHome = $7700;

KBD_CtrlI EFKEYBRD

Const EFKEYBRD.KBD_CtrlI = $1709;

KBD_CtrlIns EFKEYBRD

Const EFKEYBRD.KBD_CtrlIns = $9200;

KBD_CtrlJ EFKEYBRD

Const EFKEYBRD.KBD_CtrlJ = $240A;

KBD_CtrlK EFKEYBRD

Const EFKEYBRD.KBD_CtrlK = $250B;

KBD_CtrlL EFKEYBRD

Const EFKEYBRD.KBD_CtrlL = $260C;

KBD_CtrlLeft EFKEYBRD

Const EFKEYBRD.KBD_CtrlLeft = $7300;

KBD_CtrlM EFKEYBRD

Const EFKEYBRD.KBD_CtrlM = $320D;

KBD_CtrlMinus EFKEYBRD

Const EFKEYBRD.KBD_CtrlMinus = $0C1F;

KBD_CtrlN EFKEYBRD

Const EFKEYBRD.KBD_CtrlN = $310E;

KBD_CtrlO EFKEYBRD

Const EFKEYBRD.KBD_CtrlO = $180F;

KBD_CtrlP EFKEYBRD

Const EFKEYBRD.KBD_CtrlP = $1910;

KBD_CtrlPgDn EFKEYBRD

Const EFKEYBRD.KBD_CtrlPgDn = $7600;

KBD_CtrlPgUp EFKEYBRD

Const EFKEYBRD.KBD_CtrlPgUp = $8400;

KBD_CtrlQ EFKEYBRD

Const EFKEYBRD.KBD_CtrlQ = $1011;

KBD_CtrlR EFKEYBRD

Const EFKEYBRD.KBD_CtrlR = $1312;

KBD_CtrlRight EFKEYBRD

Const EFKEYBRD.KBD_CtrlRight = $7400;

KBD_CtrlS EFKEYBRD

Const EFKEYBRD.KBD_CtrlS = $1F13;

KBD_CtrlT EFKEYBRD

Const EFKEYBRD.KBD_CtrlT = $1414;

KBD_CtrlTAB EFKEYBRD

Const EFKEYBRD.KBD_CtrlTAB = $9400;

KBD_CtrlU EFKEYBRD

Const EFKEYBRD.KBD_CtrlU = $1615;

KBD_CtrlUp EFKEYBRD

Const EFKEYBRD.KBD_CtrlUp = $8D00;

KBD_CtrlV EFKEYBRD

Const EFKEYBRD.KBD_CtrlV = $2F16;

KBD_CtrlW EFKEYBRD

Const EFKEYBRD.KBD_CtrlW = $1117;

KBD_CtrlX EFKEYBRD

Const EFKEYBRD.KBD_CtrlX = $2D18;

KBD_CtrlY EFKEYBRD

Const EFKEYBRD.KBD_CtrlY = $1519;

KBD_CtrlZ EFKEYBRD

Const EFKEYBRD.KBD_CtrlZ = $2C1A;

KBD_D EFKEYBRD

Const EFKEYBRD.KBD_D = $2044;

KBD_Del EFKEYBRD

Const EFKEYBRD.KBD_Del = $5300;

KBD_Down EFKEYBRD

Const EFKEYBRD.KBD_Down = $5000;

KBD_E EFKEYBRD

Const EFKEYBRD.KBD_E = $1245;

KBD_End EFKEYBRD

Const EFKEYBRD.KBD_End = $4F00;

KBD_Enter EFKEYBRD

Const EFKEYBRD.KBD_Enter = $1C0D;

KBD_ESC EFKEYBRD

Const EFKEYBRD.KBD_ESC = $011B;

KBD_F EFKEYBRD

Const EFKEYBRD.KBD_F = $2146;

KBD_F1 EFKEYBRD

Const EFKEYBRD.KBD_F1 = $3B00;

KBD_F10 EFKEYBRD

Const EFKEYBRD.KBD_F10 = $4400;

KBD_F11 EFKEYBRD

Const EFKEYBRD.KBD_F11 = $8500;

KBD_F12 EFKEYBRD

Const EFKEYBRD.KBD_F12 = $8600;

KBD_F2 EFKEYBRD

Const EFKEYBRD.KBD_F2 = $3C00;

KBD_F3 EFKEYBRD

Const EFKEYBRD.KBD_F3 = $3D00;

KBD_F4 EFKEYBRD

Const EFKEYBRD.KBD_F4 = $3E00;

KBD_F5 EFKEYBRD

Const EFKEYBRD.KBD_F5 = $3F00;

KBD_F6 EFKEYBRD

Const EFKEYBRD.KBD_F6 = $4000;

KBD_F7 EFKEYBRD

Const EFKEYBRD.KBD_F7 = $4100;

KBD_F8 EFKEYBRD

Const EFKEYBRD.KBD_F8 = $4200;

KBD_F9 EFKEYBRD

Const EFKEYBRD.KBD_F9 = $4300;

KBD_G EFKEYBRD

Const EFKEYBRD.KBD_G = $2247;

KBD_Gray5 EFKEYBRD

Const EFKEYBRD.KBD_Gray5 = $4C00;

KBD_GrMin EFKEYBRD

Const EFKEYBRD.KBD_GrMin = $4A2D;

KBD_GrPls EFKEYBRD

Const EFKEYBRD.KBD_GrPls = $4E2B;

KBD_GrStar EFKEYBRD

Const EFKEYBRD.KBD_GrStar = $372A;

KBD_H EFKEYBRD

Const EFKEYBRD.KBD_H = $2348;

KBD_Home EFKEYBRD

Const EFKEYBRD.KBD_Home = $4700;

KBD_I EFKEYBRD

Const EFKEYBRD.KBD_I = $1749;

KBD_Ins EFKEYBRD

Const EFKEYBRD.KBD_Ins = $5200;

KBD_J EFKEYBRD

Const EFKEYBRD.KBD_J = $244A;

KBD_K EFKEYBRD

Const EFKEYBRD.KBD_K = $254B;

KBD_L EFKEYBRD

Const EFKEYBRD.KBD_L = $264C;

KBD_Left EFKEYBRD

Const EFKEYBRD.KBD_Left = $4B00;

KBD_M EFKEYBRD

Const EFKEYBRD.KBD_M = $324D;

KBD_Minus EFKEYBRD

Const EFKEYBRD.KBD_Minus = $0C2D;

KBD_N EFKEYBRD

Const EFKEYBRD.KBD_N = $314E;

KBD_Num0 EFKEYBRD

Const EFKEYBRD.KBD_Num0 = $0B30;

KBD_Num1 EFKEYBRD

Const EFKEYBRD.KBD_Num1 = $0231;

KBD_Num2 EFKEYBRD

Const EFKEYBRD.KBD_Num2 = $0332;

KBD_Num3 EFKEYBRD

Const EFKEYBRD.KBD_Num3 = $0433;

KBD_Num4 EFKEYBRD

Const EFKEYBRD.KBD_Num4 = $0534;

KBD_Num5 EFKEYBRD

Const EFKEYBRD.KBD_Num5 = $0635;

KBD_Num6 EFKEYBRD

Const EFKEYBRD.KBD_Num6 = $0736;

KBD_Num7 EFKEYBRD

Const EFKEYBRD.KBD_Num7 = $0837;

KBD_Num8 EFKEYBRD

Const EFKEYBRD.KBD_Num8 = $0938;

KBD_Num9 EFKEYBRD

Const EFKEYBRD.KBD_Num9 = $0A39;

KBD_O EFKEYBRD

Const EFKEYBRD.KBD_O = $184F;

KBD_P EFKEYBRD

Const EFKEYBRD.KBD_P = $1950;

KBD_PgDn EFKEYBRD

Const EFKEYBRD.KBD_PgDn = $5100;

KBD_PgUp EFKEYBRD

Const EFKEYBRD.KBD_PgUp = $4900;

KBD_Plus EFKEYBRD

Const EFKEYBRD.KBD_Plus = $0D2B;

KBD_Q EFKEYBRD

Const EFKEYBRD.KBD_Q = $1051;

KBD_R EFKEYBRD

Const EFKEYBRD.KBD_R = $1352;

KBD_Right EFKEYBRD

Const EFKEYBRD.KBD_Right = $4D00;

KBD_S EFKEYBRD

Const EFKEYBRD.KBD_S = $1F53;

KBD_ShiftA EFKEYBRD

Const EFKEYBRD.KBD_ShiftA = $1E61;

KBD_ShiftB EFKEYBRD

Const EFKEYBRD.KBD_ShiftB = $3062;

KBD_ShiftC EFKEYBRD

Const EFKEYBRD.KBD_ShiftC = $2E63;

KBD_ShiftD EFKEYBRD

Const EFKEYBRD.KBD_ShiftD = $2064;

KBD_ShiftDel EFKEYBRD

Const EFKEYBRD.KBD_ShiftDel = $532E;

KBD_ShiftDown EFKEYBRD

Const EFKEYBRD.KBD_ShiftDown = $5032;

KBD_ShiftE EFKEYBRD

Const EFKEYBRD.KBD_ShiftE = $1265;

KBD_Shiftend EFKEYBRD

Const EFKEYBRD.KBD_Shiftend = $4F31;

KBD_ShiftF EFKEYBRD

Const EFKEYBRD.KBD_ShiftF = $2166;

KBD_ShiftF1 EFKEYBRD

Const EFKEYBRD.KBD_ShiftF1 = $5400;

KBD_ShiftF10 EFKEYBRD

Const EFKEYBRD.KBD_ShiftF10 = $5D00;

KBD_ShiftF11 EFKEYBRD

Const EFKEYBRD.KBD_ShiftF11 = $8700;

KBD_ShiftF12 EFKEYBRD

Const EFKEYBRD.KBD_ShiftF12 = $8800;

KBD_ShiftF2 EFKEYBRD

Const EFKEYBRD.KBD_ShiftF2 = $5500;

KBD_ShiftF3 EFKEYBRD

Const EFKEYBRD.KBD_ShiftF3 = $5600;

KBD_ShiftF4 EFKEYBRD

Const EFKEYBRD.KBD_ShiftF4 = $5700;

KBD_ShiftF5 EFKEYBRD

Const EFKEYBRD.KBD_ShiftF5 = $5800;

KBD_ShiftF6 EFKEYBRD

Const EFKEYBRD.KBD_ShiftF6 = $5900;

KBD_ShiftF7 EFKEYBRD

Const EFKEYBRD.KBD_ShiftF7 = $5A00;

KBD_ShiftF8 EFKEYBRD

Const EFKEYBRD.KBD_ShiftF8 = $5B00;

KBD_ShiftF9 EFKEYBRD

Const EFKEYBRD.KBD_ShiftF9 = $5C00;

KBD_ShiftG EFKEYBRD

Const EFKEYBRD.KBD_ShiftG = $2267;

KBD_ShiftGray5 EFKEYBRD

Const EFKEYBRD.KBD_ShiftGray5 = $4C35;

KBD_ShiftH EFKEYBRD

Const EFKEYBRD.KBD_ShiftH = $2368;

KBD_ShiftHome EFKEYBRD

Const EFKEYBRD.KBD_ShiftHome = $4737;

KBD_ShiftI EFKEYBRD

Const EFKEYBRD.KBD_ShiftI = $1769;

KBD_ShiftIns EFKEYBRD

Const EFKEYBRD.KBD_ShiftIns = $5230;

KBD_ShiftJ EFKEYBRD

Const EFKEYBRD.KBD_ShiftJ = $246A;

KBD_ShiftK EFKEYBRD

Const EFKEYBRD.KBD_ShiftK = $256B;

KBD_ShiftL EFKEYBRD

Const EFKEYBRD.KBD_ShiftL = $266C;

KBD_ShiftLeft EFKEYBRD

Const EFKEYBRD.KBD_ShiftLeft = $4B34;

KBD_ShiftM EFKEYBRD

Const EFKEYBRD.KBD_ShiftM = $326D;

KBD_ShiftN EFKEYBRD

Const EFKEYBRD.KBD_ShiftN = $316E;

KBD_ShiftO EFKEYBRD

Const EFKEYBRD.KBD_ShiftO = $184F;

KBD_ShiftP EFKEYBRD

Const EFKEYBRD.KBD_ShiftP = $1970;

KBD_ShiftPgDn EFKEYBRD

Const EFKEYBRD.KBD_ShiftPgDn = $5133;

KBD_ShiftPgUp EFKEYBRD

Const EFKEYBRD.KBD_ShiftPgUp = $4939;

KBD_ShiftQ EFKEYBRD

Const EFKEYBRD.KBD_ShiftQ = $1071;

KBD_ShiftR EFKEYBRD

Const EFKEYBRD.KBD_ShiftR = $1372;

KBD_ShiftRight EFKEYBRD

Const EFKEYBRD.KBD_ShiftRight = $4D36;

KBD_ShiftS EFKEYBRD

Const EFKEYBRD.KBD_ShiftS = $1F73;

KBD_ShiftT EFKEYBRD

Const EFKEYBRD.KBD_ShiftT = $1474;

KBD_ShiftTAB EFKEYBRD

Const EFKEYBRD.KBD_ShiftTAB = $0F00;

KBD_ShiftU EFKEYBRD

Const EFKEYBRD.KBD_ShiftU = $1675;

KBD_ShiftUp EFKEYBRD

Const EFKEYBRD.KBD_ShiftUp = $4838;

KBD_ShiftV EFKEYBRD

Const EFKEYBRD.KBD_ShiftV = $2F76;

KBD_ShiftW EFKEYBRD

Const EFKEYBRD.KBD_ShiftW = $1177;

KBD_ShiftX EFKEYBRD

Const EFKEYBRD.KBD_ShiftX = $2D78;

KBD_ShiftY EFKEYBRD

Const EFKEYBRD.KBD_ShiftY = $1579;

KBD_ShiftZ EFKEYBRD

Const EFKEYBRD.KBD_ShiftZ = $2C7A;

KBD_Space EFKEYBRD

Const EFKEYBRD.KBD_Space = $3920;

KBD_Star EFKEYBRD

Const EFKEYBRD.KBD_Star = $092A;

KBD_T EFKEYBRD

Const EFKEYBRD.KBD_T = $1454;

KBD_TAB EFKEYBRD

Const EFKEYBRD.KBD_TAB = $0F09;

KBD_U EFKEYBRD

Const EFKEYBRD.KBD_U = $1655;

KBD_Up EFKEYBRD

Const EFKEYBRD.KBD_Up = $4800;

KBD_V EFKEYBRD

Const EFKEYBRD.KBD_V = $2F56;

KBD_W EFKEYBRD

Const EFKEYBRD.KBD_W = $1157;

KBD_X EFKEYBRD

Const EFKEYBRD.KBD_X = $2D58;

KBD_Y EFKEYBRD

Const EFKEYBRD.KBD_Y = $1559;

KBD_Z EFKEYBRD

Const EFKEYBRD.KBD_Z = $2C5A;

KernelInstalled EFKERNEL

Const EFKERNEL.KernelInstalled: boolean = FALSE;

This variable is set to TRUE when the kernel is installed, that is, kernel classes have been registered and initialized.

Keyboard EFKEYBRD

Const EFKEYBRD.Keyboard: pKeyboard = NIL;

Instance of the keyboard device object. This instance provides access to the keyboard. It can be replaced with an extended keyboard device. * Keyboard variables * **

KeyboardBIOSSegment EFKEYBRD

Const EFKEYBRD.KeyboardBIOSSegment = $0040;

Memory segment for keyboard BIOS.

KeyboardHead EFKEYBRD

EFKEYBRD.KeyboardHead: word absolute KeyboardBIOSSegment: $001A;

Memory offset for the keyboard buffer head.

KeyboardInterrupt EFKEYBRD

Const EFKEYBRD.KeyboardInterrupt = $16;

Keyboard BIOS interrupt (16h as default).

KeyboardStatus EFKEYBRD

EFKEYBRD.KeyboardStatus: set of (RightShift, LeftShift, Ctrl, Alt, ScrollLock, NumLock, CapsLock, Insert)

KeyboardTail EFKEYBRD

EFKEYBRD.KeyboardTail: word absolute KeyboardBIOSSegment: $001C;

Memory offset for the keyboard buffer tail.

LastDrive EFSTREAM

function EFSTREAM.LastDrive: char;

Returns the last drive letter that can be accessed at this time.

LightBlue EFVIEWS

Const EFVIEWS.LightBlue = 9;

LightCyan EFVIEWS

Const EFVIEWS.LightCyan = 11;

LightGray EFVIEWS

Const EFVIEWS.LightGray = 7;

LightGreen EFVIEWS

Const EFVIEWS.LightGreen = 10;

LightMagenta EFVIEWS

Const EFVIEWS.LightMagenta = 13;

LightRed EFVIEWS

Const EFVIEWS.LightRed = 12;

Log EFMATH

function EFMATH.Log(Number: tBaseReal; Base: tBaseReal): tBaseReal;

Returns the n-logarithm of the specified number with n equal to the specified base.

LowerCase EFSTRING

function EFSTRING.LowerCase(Data: string): string;

Convert a string to lower case characters using the *LowerFilter* instance in *EFSTRING*.

LowerFilter EFSTRING

Const EFSTRING.LowerFilter: pTranslation = NIL;

Upper case translation table used for case conversion of strings. See #tTranslation.

Magenta EFVIEWS

Const EFVIEWS.Magenta = 5;

MathematicalTokens EFMATH

Const EFMATH.MathematicalTokens = ['+', '-', '', '/', '(', ')', '^'] ;*

Maximum EFKERNEL

function EFKERNEL.Maximum(Number1, Number2: longint): longint;

Returns the biggest value of the two specified numbers.

MemoryStatus EFTEST

Const EFTEST.MemoryStatus: pStack = NIL;

This stack handles memory status. Used by the MemPop and MemPush routines.

MemPop EFTEST

procedure EFTEST.MemPop; far;

Asserts that no memory is lost since last MemPush command. See *MemPush* for more information.

MemPush EFTEST

procedure EFTEST.MemPush; far;

Registers the current memory status. Whenever a MemPop command is requested, the free memory is compared with the available memory when MemPush was last called. If memory has been lost, this is written to standard I/O.

Minimum EFKERNEL

function EFKERNEL.Minimum(Number1, Number2: longint): longint;

Returns the smallest value of the two specified numbers.

Mouse EFMOUSE

Const EFMOUSE.Mouse: pMouse = NIL;

Instance of the mouse device object. This instance provides access to the mouse. It can be replaced with an extended mouse device.

MouseInterrupt EFMOUSE

Const EFMOUSE.MouseInterrupt = $33;

Mouse driver interrupt.

MSE_AnyButton EFMOUSE

Const EFMOUSE.MSE_AnyButton = 7;

Any button bit (0, 1 and 2)

MSE_ConfirmButton EFMOUSE

Const EFMOUSE.MSE_ConfirmButton: byte = MSE_LeftButton;

Button for double clicking: "select-and-confirm"

MSE_LeftButton EFMOUSE

Const EFMOUSE.MSE_LeftButton = 1;

Left button bit (0)

MSE_MiddleButton EFMOUSE

Const EFMOUSE.MSE_MiddleButton = 4;

Middle button bit (2)

MSE_RightButton EFMOUSE

Const EFMOUSE.MSE_RightButton = 2;

Right button bit (1)

MSE_SelectButton EFMOUSE

Const EFMOUSE.MSE_SelectButton: byte = MSE_LeftButton;

Button for "selection without confirming"

NextPrime EFMATH

function EFMATH.NextPrime(Number: longint): longint;

Returns the nearest prime number after the specified integer.

Null EFDEF

Const EFDEF.Null: char = #0;

Null character for null-terminated strings.

NumericChars EFDEF

Const EFDEF.NumericChars: tChars = ['0'..'9', ',', '.'] ;

Numberical character set: all characters that is valid for numbers (without operators).

pAdjustedList EFLIST

EFLIST.pAdjustedList = ^tAdjustedList;

pADT EFADT

EFADT.pADT = ^tADT;

pADTPlug EFADT

EFADT.pADTPlug = ^tADTPlug;

pADTSelector EFADT

EFADT.pADTSelector = ^tADTSelector;

pADTStream EFADT

EFADT.pADTStream = ^tADTStream;

pAlignemnt EFDEF

EFDEF.pAlignemnt = ^tAlignment;

pAllocation EFMEMORY

EFMEMORY.pAllocation = ^tAllocation;

pApplication EFAPP

EFAPP.pApplication = ^tApplication;

pArchive EFRES

EFRES.pArchive = ^tArchive;

pArrangement EFVIEWS

EFVIEWS.pArrangement = ^tArrangement;

pArray EFARRAY

EFARRAY.pArray = ^tArray;

Path EFSTREAM

function EFSTREAM.Path: string;
Returns the current directory, e.g. "c:\mydir".

pAVLLinkage EFTREE

EFTREE.pAVLLinkage = ^tAVLLinkage;

pAVLTree EFTREE

EFTREE.pAVLTree = ^tAVLTree;

pBinaryTree EFTREE

EFTREE.pBinaryTree = ^tBinaryTree;

pBit EFDEF

EFDEF.pBit = ^tBit;

pBorder EFDEF

EFDEF.pBorder = ^tBorder;

pBoundedArray EFARRAY

EFARRAY.pBoundedArray = ^tBoundedArray;

pBoundedConsole EFCONDRV

EFCONDRV.pBoundedConsole = ^tBoundedConsole;

pBoundedKeyPlug EFADT

EFADT.pBoundedKeyPlug = ^tBoundedKeyPlug;

pBubbleSortPlug EFSORT

EFSORT.pBubbleSortPlug = ^tBubbleSortPlug;

pBucket EFTABLE

EFTABLE.pBucket = ^tBucket;

pBucketHashTable EFTABLE

EFTABLE.pBucketHashTable = ^tBucketHashTable;

pBucketIterator EFTABLE

EFTABLE.pBucketIterator = ^tBucketIterator;

pBuffer EFMEMORY

EFMEMORY.pBuffer = ^tBuffer;

pBufferedScreen EFSCREEN

EFSCREEN.pBufferedScreen = ^tBufferedScreen;

pBufferFilter EFFILTER

EFFILTER.pBufferFilter = ^tBufferFilter;

pByteArray EFDEF

EFDEF.pByteArray = ^tByteArray;

pCarrier EFKERNEL

EFKERNEL.pCarrier = ^tCarrier;

pCarrierElement EFELEM

EFELEM.pCarrierElement = ^tCarrierElement;

pCarrierRegister EFKERNEL

EFKERNEL.pCarrierRegister = ^tCarrierRegister;

pCaster EFMATH

EFMATH.pCaster = ^tCaster;

pCharacters EFBASIC

EFBASIC.pCharacters = ^tCharacters;

pCharArray EFDEF

EFDEF.pCharArray = ^tCharArray;

pChars EFDEF

EFDEF.pChars = ^tChars;

pCircularQueue EFDATA

EFDATA.pCircularQueue = ^tCircularQueue;

pClassIdentity EFKERNEL

EFKERNEL.pClassIdentity = ^tClassIdentity;

pClassManager EFKERNEL

EFKERNEL.pClassManager = ^tClassManager;

pClassRegister EFKERNEL

EFKERNEL.pClassRegister = ^tClassRegister;

pClassSelector EFSYSTEM

EFSYSTEM.pClassSelector = ^tClassSelector;

pColor EFCONDRV

EFCONDRV.pColor = ^tColor;

pCommandLine EFCLINE

EFCLINE.pCommandLine = ^tCommandLine;

pCommunicator EFCMAN

EFCMAN.pCommunicator = ^tCommunicator;

pComplex EFMATH

EFMATH.pComplex = ^tComplex;

pComponent EFCMAN

EFCMAN.pComponent = ^tComponent;

pComponentSelector EFSYSTEM

EFSYSTEM.pComponentSelector = ^tComponentSelector;

pCompositeADT EFDATA

EFDATA.pCompositeADT = ^tCompositeADT;

pCompressFilter EFFILTER

EFFILTER.pCompressFilter = ^tCompressFilter;

pConsole EFCONDRV

EFCONDRV.pConsole = ^tConsole;

pConsoleSelector EFCONDRV

EFCONDRV.pConsoleSelector = ^tConsoleSelector;

pContainer EFKERNEL

EFKERNEL.pContainer = ^tContainer;

pConverter EFCONV

EFCONV.pConverter = ^tConverter;

pCoordinates EFBASIC

EFBASIC.pCoordinates = ^tCoordinates;

pCRC16Filter EFFILTER

EFFILTER.pCRC16Filter = ^tCRC16Filter;

pCRC32Filter EFFILTER

EFFILTER.pCRC32Filter = ^tCRC32Filter;

pCRT EFCONDRV

EFCONDRV.pCRT = ^tCRT;

pCursorList EFLIST

EFLIST.pCursorList = ^tCursorList;

pDataStream EFSTREAM

EFSTREAM.pDataStream = ^tDataStream;

pDate EFTIME

EFTIME.pDate = ^tDate;

pDelayedSearchFilter EFSEARCH

EFSEARCH.pDelayedSearchFilter = ^tDelayedSearchFilter;

pDevice EFDEVICE

EFDEVICE.pDevice = ^tDevice;

pDirection EFDEF

EFDEF.pDirection = ^tDirection;

pDirectory EFIO

EFIO.pDirectory = ^tDirectory;

pDoubleHashTable EFTABLE

EFTABLE.pDoubleHashTable = ^tDoubleHashTable;

pDuplicateFilter EFFILTER

EFFILTER.pDuplicateFilter = ^tDuplicateFilter;

pElement EFELEM

EFELEM.pElement = ^tElement;

pEncryptFilter EFFILTER

EFFILTER.pEncryptFilter = ^tEncryptFilter;

pEnvironment EFSYSTEM

EFSYSTEM.pEnvironment = ^tEnvironment;

pError EFKERNEL

EFKERNEL.pError = ^tError;

pErrorHandler EFKERNEL

EFKERNEL.pErrorHandler = ^tErrorHandler;

pErrorMessage EFKERNEL

EFKERNEL.pErrorMessage = ^tErrorMessage;

pEvaluator EFMEXP

EFMEXP.pEvaluator = ^tEvaluator;

pExpression EFMEXP

EFMEXP.pExpression = ^tExpression;

pExternalF EFMATH

EFMATH.pExternalF = ^tExternalF;

pExternalH EFMATH

EFMATH.pExternalH = ^tExternalF;

pField EFBASIC

EFBASIC.pField = ^tField;

pFile EFFILE

EFFILE.pFile = ^tFile;

pFilename EFIO

EFIO.pFilename = ^tFilename;

pFileStream EFSTREAM

EFSTREAM.pFileStream = ^tFileStream;

pFillStyle EFCONDRV

EFCONDRV.pFillStyle = ^tFillStyle;

pFilter EFFILTER

EFFILTER.pFilter = ^tFilter;

pFlag EFSYSTEM

EFSYSTEM.pFlag = ^tFlag;

pFocusedEvent EFMSG

EFMSG.pFocusedEvent = ^tFocusedEvent;

pFont EFCONDRV

EFCONDRV.pFont = ^tFont;

pGenericElement EFELEM

EFELEM.pGenericElement = ^tGenericElement;

pGroup EFVIEWS

EFVIEWS.pGroup = ^tGroup;

pHandle EFKERNEL

EFKERNEL.pHandle = ^tHandle;

EFREG.pHandle = ^tHandle;

pHashPlug EFTABLE

EFTABLE.pHashPlug = ^tHashPlug;

pHashTable EFTABLE

EFTABLE.pHashTable = ^tHashTable;

pHugeArray EFDEF

EFDEF.pHugeArray = ^tHugeArray;

pHyperText EFTEXT

EFTEXT.pHyperText = ^tHyperText;

Pi EFMATH

*Const EFMATH.Pi: tBaseReal =
3.14159265358979323846264338;*

pIdentityElement EFELEM

EFELEM.pIdentityElement = ^tIdentityElement;

pIFile EFFILE

EFFILE.pIFile = ^tIFile;

pImage EFCONDRV

EFCONDRV.pImage = ^tImage;

pInsertionSortPlug EFSORT

EFSORT.pInsertionSortPlug = ^tInsertionSortPlug;

pInstallMessage EFDEVICE

EFDEVICE.pInstallMessage = ^tInstallMessage;

pInteger EFMATH

EFMATH.pInteger = ^tInteger;

pIterator EFADT

EFADT.pIterator = ^tIterator;

pKeyboard EFKEYBRD

EFKEYBRD.pKeyboard = ^tKeyboard;

pKeyEvent EFKEYBRD

EFKEYBRD.pKeyEvent = ^tKeyEvent;

pKeyPlug EFADT

EFADT.pKeyPlug = ^tKeyPlug;

pLinearADT EFADT

EFADT.pLinearADT = ^tLinearADT;

pLinearIterator EFADT

EFADT.pLinearIterator = ^tLinearIterator;

Element iterator class that provides a fast mechanism of walking through elements in an arbitrary data structure.

pLineStyle EFCONDRV

EFCONDRV.pLineStyle = ^tLineStyle;

pLinkage EFLIST

EFLIST.pLinkage = ^tLinkage;

pList EFLIST

EFLIST.pList = ^tList;

pLocalElement EFSYSTEM

EFSYSTEM.pLocalElement = ^tLocalElement;

pLocalMessage EFMSG

EFMSG.pLocalMessage = ^tLocalMessage;

pLocalProcedure EFMSG

EFMSG.pLocalProcedure = ^tLocalProcedure;

pManager EFCMAN

EFCMAN.pManager = ^tManager;

pMapping EFSYSTEM

EFSYSTEM.pMapping = ^tMapping;

pMatchFilter EFSEARCH

EFSEARCH.pMatchFilter = ^tMatchFilter;

pMathFunction EFMATH

EFMATH.pMathFunction = ^tMathFunction;

EFMEXP.pMathFunction = ^tMathFunction;

pMathObject EFMATH

EFMATH.pMathObject = ^tMathObject;

pMatrix EFMATH

EFMATH.pMatrix = ^tMatrix;

pMemoryStream EFMEMORY

EFMEMORY.pMemoryStream = ^tMemoryStream;

pMergeSortPlug EFSORT

EFSORT.pMergeSortPlug = ^tMergeSortPlug;

pMessage EFCMAN

EFCMAN.pMessage = ^tMessage;

pMouseDevice EFMOUSE

EFMOUSE.pMouseDevice = ^tMouseDevice;

pMouseEvent EFMOUSE

EFMOUSE.pMouseEvent = ^tMouseEvent;

pNamedElement EFSYSTEM

EFSYSTEM.pNamedElement = ^tNamedElement;

pNodeIterator EFLIST

EFLIST.pNodeIterator = ^tNodeIterator;

Element iterator class that provides a fast mechanism of walking through elements in doubly linked data structures.

pNullStream EFSTREAM

EFSTREAM.pNullStream = ^tNullStream;

pNumberElement EFSYSTEM

EFSYSTEM.pNumberElement = ^tNumberElement;

pObject EFKERNEL

EFKERNEL.pObject = ^tObject;

pOFile EFFILE

EFFILE.pOFile = ^tOFile;

pOpenHashTable EFTABLE

EFTABLE.pOpenHashTable = ^tOpenHashTable;

pOperator EFMEXP

EFMEXP.pOperator = ^tOperator;

pOrderedADT EFADT

EFADT.pOrderedADT = ^tOrderedADT;

pOrderedList EFLIST

EFLIST.pOrderedList = ^tOrderedList;

Power EFMATH

function EFMATH.Power(Base, Exponent: tBaseReal): tBaseReal;

Returns the power of a number, ie. Base ^ Exponent, for any base and exponent, if the result is within valid range. Zero is returned if a range error occurrs.

pParser EFTEXT

EFTEXT.pParser = ^tParser;

pPath EFIO

EFIO.pPath = ^tPath;

pPathPlug EFTREE

EFTREE.pPathPlug = ^tPathPlug;

pPattern EFPATRN

EFPATRN.pPattern = ^tPattern;

pPatternManager EFPATRN

EFPATRN.pPatternManager = ^tPatternManager;

pPatternNumber EFPATRN

EFPATRN.pPatternNumber = ^tPatternNumber;

pPatternString EFPATRN

EFPATRN.pPatternString = ^tPatternString;

pPlug EFPLUG

EFPLUG.pPlug = ^tPlug;

pPlugManager EFPLUG

EFPLUG.pPlugManager = ^tPlugManager;

pPolynomial EFMATH

EFMATH.pPolynomial = ^tPolynomial;

pPresentationMethod EFDEF

EFDEF.pPresentationMethod = ^tPresentationMethod;

pPriorityQueue EFDATA

EFDATA.pPriorityQueue = ^tPriorityQueue;

pProbeHashTable EFTABLE

EFTABLE.pProbeHashTable = ^tProbeHashTable;

pProcessPhase EFDEF

EFDEF.pProcessPhase = ^tProcessPhase;

pProfile EFSYSTEM

EFSYSTEM.pProfile = ^tProfile;

pQueue EFDATA

EFDATA.pQueue = ^tQueue;

pQuickSortPlug EFADT

EFADT.pQuickSortPlug = ^tQuickSortPlug;

pRational EFMATH

EFMATH.pRational = ^tRational;

pReal EFMATH

EFMATH.pReal = ^tReal;

pRegister EFREG

EFREG.pRegister = ^tRegister;

pReplaceFilter EFSEARCH

EFSEARCH.pReplaceFilter = ^tReplaceFilter;

pResource EFRES

EFRES.pResource = ^tResource;

pReversedKeyPlug EFADT

EFADT.pReversedKeyPlug = ^tReversedKeyPlug;

pReversedList EFLIST

EFLIST.pReversedList = ^tReversedList;

Primes EFMATH

Const EFMATH.Primes: array = (2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 51, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97);

The first 26 prime numbers (primes below 100).

Profile EFSYSTEM

Const EFSYSTEM.Profile: pSystemProfile = NIL;

System profile. This instance contains a register of settings and status flags. The settings are organized in sub-profiles such as "Environment" and "System".

pScreen EFSCREEN

EFSCREEN.pScreen = ^tScreen;

pSearchFilter EFSEARCH

EFSEARCH.pSearchFilter = ^tSearchFilter;

pSequentialFilter EFFILTER

EFFILTER.pSequentialFilter = ^tSequentialFilter;

pServiceMessage EFCMAN

EFCMAN.pServiceMessage = ^tServiceMessage;

pSet EFBASIC

EFBASIC.pSet = ^tSet;

pSetting EFSYSTEM

EFSYSTEM.pSetting = ^tSetting;

pSetup EFKERNEL

EFKERNEL.pSetup = ^tSetup;

pSolver EFMATH

EFMATH.pSolver = ^tSolver;

pSortPlug EFADT

EFADT.pSortPlug = ^tSortPlug;

pSoundDevice EFDEVICE

EFDEVICE.pSoundDevice = ^tSoundDevice;

pSparceIterator EFARRAY

EFARRAY.pSparceIterator = ^tSparceIterator;

EFTABLE.pSparceIterator = ^tSparceIterator;

pSplayTree EFTREE

EFTREE.pSplayTree = ^tSplayTree;

pStack EFDATA

EFDATA.pStack = ^tStack;

pStandardStream EFSTREAM

EFSTREAM.pStandardStream = ^tStandardStream;

pStaticElement EFELEM

EFELEM.pStaticElement = ^tStaticElement;

pStream EFKERNEL

EFKERNEL.pStream = ^tStream;

pStreamableFile EFRES

EFRES.pStreamableFile = ^tStreamableFile;

pStreamElement EFELEM

EFELEM.pStreamElement = ^tStreamElement;

pStreamHandle EFMEMORY

EFMEMORY.pStreamHandle = ^tStreamHandle;

pString EFSTRING

EFSTRING.pString = ^tString;

pStringMatcher EFSTRING

EFSTRING.pStringMatcher = ^tStringMatcher;

pStyle EFSTYLES

EFSTYLES.pStyle = ^tStyle;

pSwapFilter EFSYSTEM

EFSYSTEM.pSwapFilter = ^tSwapFilter;

pSystemProfile EFSYSTEM

EFSYSTEM.pSystemProfile = ^tSystemProfile;

pTemporaryFile EFFILE

EFFILE.pTemporaryFile = ^tTemporaryFile;

pText EFTEXT

EFTEXT.pText = ^tText;

pTextFile EFTEXT

EFTEXT.pTextFile = ^tTextFile;

pTextFilter EFFILTER

EFFILTER.pTextFilter = ^tTextFilter;

pTextHashPlug EFTABLE

EFTABLE.pTextHashPlug = ^tTextHashPlug;

pTextResource EFTEXT

EFTEXT.pTextResource = ^tTextResource;

pTime EFBASIC

EFBASIC.pTime = ^tTime;

EFTIME.pTime = ^tTime;

pTimeDate EFTIME

EFTIME.pTimeDate = ^tTimeDate;

pTimer EFTIME

EFTIME.pTimer = ^tTimer;

EFBASIC.pTimer = ^tTimer;

pTiming EFTIME

EFTIME.pTiming = ^tTiming;

pToken EFSTRING

EFSTRING.pToken = ^tToken;

pTranslation EFSTRING

EFSTRING.pTranslation = ^tTranslation;

pTree EFTREE

EFTREE.pTree = ^tTree;

pTreeIterator EFTREE

EFTREE.pTreeIterator = ^tTreeIterator;

pTreeLinkage EFTREE

EFTREE.pTreeLinkage = ^tTreeLinkage;

pVariable EFMEXP

EFMEXP.pVariable = ^tVariable;

pVector EFMATH

EFMATH.pVector = ^tVector;

pView EFVIEWS

EFVIEWS.pView = ^tView;

pVirtualArray EFARRAY

EFARRAY.pVirtualArray = ^tVirtualArray;

pVirtualConsole EFCONDRV

EFCONDRV.pVirtualConsole = ^tVirtualConsole;

pWildcard EFPATRN

EFPATRN.pWildcard = ^tWildcard;

Red EFVIEWS

Const EFVIEWS.Red = 4;

RemoveKernel EFKERNEL

procedure EFKERNEL.RemoveKernel;

Removes the kernel before the program terminates. This procedure calls tInterceptor.

RuntimeErrorHandler EFKERNEL

procedure EFKERNEL.RuntimeErrorHandler;

Run-time error handler that directs errors to the error handler. This procedure is called whenever the program terminates. It separates normal terminations from abnormal terminations.

sClose EFDEF

Const EFDEF.sClose: word = 5;

This service forces the receiver to destruct immediately.

Screen EFCONDRV

Const EFCONDRV.Screen: pConsole = NIL;

Screen driver. All visual output is passed to this *tConsole* instance. You can set this pointer to some other instance if you want to associate visual output to some other device.

SearchPath EFSTREAM

function EFSTREAM.SearchPath(Filename: string): PathStr;

Searches the MS-DOS path for specified file and returns the full filename with path if it is found.

SearchVariable EFBASIC

function EFBASIC.SearchVariable(var SearchString: string; var Data; Size: word): word;

Returns the position of a string inside a variable (Data, Size). The positions must have the value (1 .. Size). If the string doesn't occur, zero is returned.

Setup EFKERNEL

Const EFKERNEL.Setup: pSetup = NIL;

Kernel setup. This instance defines the basic settings for EFLIB, for example default error messages and national case translations.

Setup_DefaultTreeOrder EFTREE

Const EFTREE.Setup_DefaultTreeOrder: tTreeOrder = InOrder;

Setup_TextBuffer EFTEXT

Const EFTEXT.Setup_TextBuffer: word = 2048;

Default text file buffer for the internal *tFile* associated to the *tTextFile* class. The default value is 2K bytes.

sFocused EFDEF

Const EFDEF.sFocused: word = 0;

This service tells the receiver that it has been focused.

Sgn EFMATH

function EFMATH.Sgn(X: tBaseReal): shortint;

Returns the sign of a number (+1 or -1).

Sinh EFMATH

function EFMATH.Sinh(X: tBaseReal): tBaseReal;

Returns the hyperbolic sine of the given angle.

SizeOfFile EFSTREAM

function EFSTREAM.SizeOfFile(Filename: string): longint;

Returns the size of a specified file in bytes or -1 if the filename doesn′t correspond to an existing file.

sLock EFDEF

Const EFDEF.sLock: word = 60;

This services makes the receiver component locked. See tComponent.

Speaker EFDEVICE

Const EFDEVICE.Speaker: pSoundDevice = NIL;

The speaker device. This device handles the internal speaker. It is an instance of *tSoundDevice*, but it can be customized.

sProtect EFDEF

Const EFDEF.sProtect: word = 50;

This services makes the receiver component protected. Protection can be released with the *sUnprotect* service. See tComponent.

StandardChars EFDEF

Const EFDEF.StandardChars: tChars = [#32..#255] ;

Standard character set: all characters except the control characters (ASCII 32-255).

StdIO EFSTREAM

Const EFSTREAM.StdIO: pStream = NIL;

Standard I/O device. This stream is the standard input and ouput device, that is, user input and text outputs should be directed to StdIO. By default, StdIO is a *tStandardStream* instance - but StdIO can be any stream of your choice.

sTouch EFDEF

Const EFDEF.sTouch: word = 10;

This service touchs the receiver so it knows that its status have changed in some way. sTouch is sent to views when their properties have changed. The receiver must check for Z-orders, etc.

StringBoolean EFSTRING

function EFSTRING.StringBoolean(Truth: boolean): string;

Converts the specified boolean variable into a string (either 'TRUE' or 'FALSE').

StringHexNumber EFSTRING

---

function EFSTRING.StringHexNumber(Number: longint): string;

Converts a integer-type variable to a hexadecimal string, e.g. 10 –> 'A'.

StringIsInteger EFSTRING

---

function EFSTRING.StringIsInteger(Data: string): boolean;

Returns TRUE if the specified string is a valid integer (such as '10000000') without performing any range checks.

StringIsReal EFSTRING

---

function EFSTRING.StringIsReal(Data: string): boolean;

Returns TRUE if the specified string is a valid real number.

StringNumber EFSTRING

---

function EFSTRING.StringNumber(Number: real; Width, Decimals: byte): string;

Converts a real type number to a string, with specified width and number of decimals. StringNumber (5.12, 2, 1) corresponds to 5.12:1:2.

StringPointer EFSTRING

function EFSTRING.StringPointer(Address: pointer): string;

Converts a memory address to a hexadecimal string representation, e.g. SEGB800 –> '$B800:0000'.

StringValue EFSTRING

function EFSTRING.StringValue(Data: string): real;

Converts a string into a real number, e.g. '12' –> 12. Returns zero if the string does not contain a valid real number.

SummarizeData EFBASIC

function EFBASIC.SummarizeData(var Data; Size: word): longint;

Returns the summarized data in the specified variable (Data, Size). All bytes are added together into one single number.

sUnlock EFDEF

Const EFDEF.sUnlock: word = 61;

This services unlocks the receiver component. See tComponent.

sUnprotect EFDEF

Const EFDEF.sUnprotect: word = 51;

This services makes the receiver component unprotected (default state). See tComponent.

sUpdate EFDEF

Const EFDEF.sUpdate: word = 11;

This service forces the receiver to update itself even if it considers itself unchanged. In views, the sUpdate service results in an unconditional redraw.

Swap EFSYSTEM

Const EFSYSTEM.Swap: pSwapFilter = NIL;

This is the swap stream, that is a stream supplied by *EFSYSTEM* for large data transfers. Whenever you use the swap file, you must start by requesting the storage you need. After you are done, that storage must be released from the swap file. See *tSwapFilter* for more information.

SwapVariable EFBASIC

procedure EFBASIC.SwapVariable(var Data1, Data2; Size: word);

Swaps the contents of two variables of equal size.

tAccess EFKERNEL

EFKERNEL.tAccess = (PermitRead, PermitWrite, PermitFull);

This type declares the access mode for streams. Streams can be in three modes: write-only (PermitRead), read-only (PermitWrite) or full access (PermitFull).

tAdjustedList EFLIST

EFLIST.tAdjustedList = object(tReversedList)

This class implements a self-organizing list, a list that improves future performance based on current usage. The self-organizing list is basically a reversed linked list, but the methods *Get* and *Locate* are overridden. After the default operations are performed, each of these methods moves the accessed element to the front of the list, hence giving it faster access time.

Fields and Methods:

public
constructor *Initialize...*
constructor *Resemble...*
constructor *Duplicate...*
function *Get...* virtual;
function *Locate...* virtual;
constructor *StreamLoad...*

Methods

Duplicate

constructor tAdjustedList.Duplicate(ADT: pADT);

Initializes a duplicate ADT instance (with equal elements).

Overrides tReversedList.Duplicate.

Overrides tList.Duplicate.

Get

function tAdjustedList.Get(Index: word): pElement; virtual;

Gets the specified element instance.

Overrides tList.Get.

Overrides tLinearADT.Get.

Initialize

constructor tAdjustedList.Initialize(SizeOfElements: word);

Initializes an instance of this ADT class.

Overrides tReversedList.Initialize.

Overrides tList.Initialize.

Overrides tLinearADT.Initialize.

Overrides tADT.Initialize.

Overrides tObject.Initialize.

Locate *function tAdjustedList.Locate(MatchElement: pElement): word; virtual;*

Searchs for element and returns found element or zero.

Overrides tLinearADT.Locate.

Resemble *constructor tAdjustedList.Resemble(ADT: pADT);*

Initializes an ADT that resembles the specified ADT.

Overrides tReversedList.Resemble.

Overrides tList.Resemble.

StreamLoad *constructor tAdjustedList.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

Overrides tReversedList.StreamLoad.

Overrides tList.StreamLoad.

Overrides tADT.StreamLoad.

Overrides tObject.StreamLoad.

tADT EFADT

EFADT.tADT = object(tObject)

This is the base class for all data structures. It is abstract and cannot be instantiated. All ADT′s descend from this class (and either *tLinearADT* or *tOrderedADT*).

An abstract data type (ADT) is a data type that does not have any design or implementation information associated with it. An abstract data type is used without regard for how its logical domain and methods are designed or implemented. It has a polymorphic interface.

Common behavior for tADT descendants:

(1) An ADT must construct an element iterator when the method *CreateIterator* is called.

(2) An ADT must construct an element when the method *CreateElement* is called.

(3) Elements can be transferred between ADT′s using the methods *CopyIn* and *CopyOut*. Ordered ADT′s then use the default element order.

(4) An ADT is classified as homogeneous if all elements are equally large (or of the same type).

(3) and (4) are called component selectors and are provided for high extendibility.

The following methods in tADT must be overridden: *Clear*, *Put*, *Elements*, *StreamLoad*.

You may also want to override: *Capacity*, *IsAllocated*, *IsReadOnly*, *StreamStore*, *StreamImport* and *StreamExport*.

Fields and Methods:

public
fDefaultSize: word;
fPlugs: *pPlugManager*;
constructor *Initialize*;
destructor *Intercept*; virtual;
procedure *SetDefaultSize...* virtual;
procedure *InstallPlug...* virtual;
procedure *Clear*; virtual;
procedure *Store...* virtual;
procedure *Put...* virtual;
function *Compare...* virtual;
procedure *CopyIn...* virtual;
procedure *CopyOut...* virtual;
function *CreateIterator...* virtual;
function *CreateElement...* virtual;
constructor *StreamLoad...*
procedure *StreamStore...* virtual;
procedure *StreamLoadProperties...* virtual;
procedure *StreamStoreProperties...* virtual;
procedure *StreamBuild...* virtual;
procedure *StreamImport...* virtual;
procedure *StreamExport...* virtual;
function *Elements...* virtual;
function *Capacity...* virtual;
function *DefaultSize...* virtual;
function *PackageSize...* virtual;
function *IsAllocated...* virtual;
function *IsReadOnly...* virtual;
function *IsEmpty...* virtual;
function *IsHomogeneous...* virtual;
function *IsFree...* virtual;
function *IsInside...* virtual;

function *IsEqual...* virtual;
function *IsCompatible...* virtual;

Descendants:

tLinearADT
  EFARRAY.tArray
    EFARRAY.tBoundedArray
      EFARRAY.tVirtualArray
  EFLIST.tList
    EFLIST.tCursorList
    EFLIST.tReversedList
      EFLIST.tAdjustedList
      EFLIST.tOrderedList
tOrderedADT
  EFDATA.tCompositeADT
    EFDATA.tQueue
      EFDATA.tCircularQueue
      EFDATA.tPriorityQueue
    EFDATA.tStack
    EFTABLE.tHashTable
      EFTABLE.tBucketHashTable
      EFTABLE.tOpenHashTable
        EFTABLE.tDoubleHashTable
        EFTABLE.tProbeHashTable
  EFTREE.tBinaryTree
    EFTREE.tAVLTree
    EFTREE.tSplayTree
  EFTREE.tTree

Fields

fDefaultSize *tADT.fDefaultSize: word;*

Default size of elements (in bytes).

fPlugs *tADT.fPlugs: pPlugManager;*

Optional ADT plugs that replace or support new operations.

Methods

Capacity *function tADT.Capacity: word; virtual;*

Returns the storage capacity in number of elements.

Clear *procedure tADT.Clear; virtual;*

Clears or removes all elements inside data type.

Compare *function tADT.Compare(A, B: pElement): shortint; virtual;*

Compares elements. Uses fKeyPlug if it's assigned.

CopyIn *procedure tADT.CopyIn(Source: pADT); virtual;*

Copies all elements from the specified ADT into this ADT.

CopyOut *procedure tADT.CopyOut(Target: pADT); virtual;*

Copies all elements from this ADT into the specified ADT.

CreateElement *function tADT.CreateElement: pElement; virtual;*

Creates an instance of the element class.

CreateIterator *function tADT.CreateIterator: pIterator; virtual;*

Creates an instance of the iterator class.

DefaultSize *function tADT.DefaultSize: word; virtual;*

Returns the default element size (in bytes).

Elements *function tADT.Elements: word; virtual;*

Returns the number of elements that currently are used.

Initialize *constructor tADT.Initialize;*

Initializes an (abstract) ADT instance.

Overrides tObject.Initialize.

InstallPlug *procedure tADT.InstallPlug(Plug: pPlug); virtual;*

Installs a new plug for this ADT.

Intercept *destructor tADT.Intercept; virtual;*

Intercepts an ADT and its key plug.

Overrides tObject.Intercept.

IsAllocated *function tADT.IsAllocated: boolean; virtual;*

Is this ADT allocated, that is ready for use?

IsCompatible *function tADT.IsCompatible(Instance: pObject): boolean; virtual;*

Are ADT's compatible, that is has compatible elements.

Overrides tObject.IsCompatible.

IsEmpty *function tADT.IsEmpty: boolean; virtual;*

Is this ADT type empty, that is does not have any elements?

IsEqual *function tADT.IsEqual(Instance: pObject): boolean; virtual;*

Are these ADT′s equal, that is contain the same elements?

Overrides tObject.IsEqual.

IsFree *function tADT.IsFree: boolean; virtual;*

Has this ADT storage for at least another element?

IsHomogeneous *function tADT.IsHomogeneous: boolean; virtual;*

Is this ADT homogeneous, that is has compatible elements?

IsInside *function tADT.IsInside(MatchElement: pElement): boolean; virtual;*

Is the specified element (contents) in this structure?

IsReadOnly *function tADT.IsReadOnly: boolean; virtual;*

Is this ADT restricted to read access?

PackageSize *function tADT.PackageSize: word; virtual;*

Returns the size of element packaging (container).

Put *procedure tADT.Put(Element: pElement); virtual;*

Stores a new element (tElement instance) to this ADT.

SetDefaultSize *procedure tADT.SetDefaultSize(SizeOfElements: word); virtual;*

Sets the default size of elements (in bytes).

Store *procedure tADT.Store(var Data); virtual;*

Stores a new element to this ADT (creates a tElement).

StreamBuild *procedure tADT.StreamBuild(NumberOfElements: word); virtual;*

Builds this instance after properties have been loaded.

StreamExport *procedure tADT.StreamExport(Stream: pStream); virtual;*

Exports all elements to a data stream.

StreamImport *procedure tADT.StreamImport(Stream: pStream; NumberOfElements: word); virtual;*

Imports some elements from a data stream.

StreamLoad *constructor tADT.StreamLoad(Stream: pStream);*

Loads an ADT from a stream.

Overrides tObject.StreamLoad.

StreamLoadProperties *procedure tADT.StreamLoadProperties(Stream: pStream); virtual;*

Loads properties of this ADT from a stream (not elements).

StreamStore *procedure tADT.StreamStore(Stream: pStream); virtual;*

Stores an ADT to a stream.

Overrides tObject.StreamStore.

StreamStoreProperties *procedure tADT.StreamStoreProperties(Stream: pStream); virtual;*

Stores properties of this ADT to a stream (not elements).

tADTPlug EFADT

EFADT.tADTPlug = object(tPlug)

Abstract plug class for data structures. This class define the interface for a plug (see for example *tPlug* and *tSortPlug*).

Fields and Methods:

public
procedure *Install...* virtual;
function *ADT...* virtual;
function *TypeOfPlug...* virtual;

Descendants:

tKeyPlug
  tBoundedKeyPlug
  tReversedKeyPlug
tSortPlug
  tQuickSortPlug

EFSORT.tBubbleSortPlug
EFSORT.tInsertionSortPlug
EFSORT.tMergeSortPlug
EFTABLE.tHashPlug
EFTABLE.tTextHashPlug
EFTREE.tPathPlug

Methods

ADT *function tADTPlug.ADT: pADT; virtual;*

Returns the owner of this plug (the resource).

Install *procedure tADTPlug.Install(PlugManager: pPlugManager); virtual;*

Installs the plug in the specified plug manager.

Overrides tPlug.Install.

TypeOfPlug *function tADTPlug.TypeOfPlug: string; virtual;*

The classification of this plug, e.g. ″tSortPlug″.

Overrides tPlug.TypeOfPlug.

tADTSelector EFADT

EFADT.tADTSelector = object(tObject)

This is a component selector class for ADT′s. This class provides methods that create (default) components when an ADT request such. You can override methods in this class to change the default behavior of ADT′s. See *ADTSelector*.

All component selectors must return valid components after they have been called. If this is not possible, the component selector trigger a fatal error that terminates program execution.

Fields and Methods:

public
constructor *Initialize*;
function *CreateIterator...* virtual;
function *CreateElement...* virtual;
function *CreateKeyPlug...* virtual;
function *CreateSortPlug...* virtual;

Methods

CreateElement *function tADTSelector.CreateElement: pElement; virtual;*

Creates an instance of the element class (tGenericElement).

CreateIterator *function tADTSelector.CreateIterator(ADT: pADT): pIterator; virtual;*

Creates an instance of the iterator class (tIterator).

CreateKeyPlug *function tADTSelector.CreateKeyPlug: pKeyPlug; virtual;*

Creates an instance of the key plug class.

CreateSortPlug *function tADTSelector.CreateSortPlug: pSortPlug; virtual;*

Creates an instance of the sort plug class.

Initialize *constructor tADTSelector.Initialize;*

Initializes an instance of this class.

Overrides tObject.Initialize.

tADTStream EFADT

EFADT.tADTStream = object(tStream)

This class implements a stream interface for an arbitrary linear and homogeneous ADT. Read and write operations are done directly to the element contents, with seamless access through different elements. tADTStream automatically selects the element(s) that are concerned by your operation.

Fields and Methods:

public
constructor *Initialize...*
procedure *Read...* virtual;
procedure *Write...* virtual;
function *Size...* virtual;
function *IsAllocated...* virtual;
function *IsReady...* virtual;
private

fLinearADT: *pLinearADT*;

Fields

fLinearADT *tADTStream.fLinearADT: pLinearADT;*

Pointer to the linear ADT used for the stream.

Methods

Initialize *constructor tADTStream.Initialize(LinearADT: pLinearADT);*

Initializes and constructs an instance of this type.

Overrides tStream.Initialize.

Overrides tObject.Initialize.

IsAllocated *function tADTStream.IsAllocated: boolean; virtual;*

Is the stream allocated, ie. ready to be used in some way?

Overrides tStream.IsAllocated.

IsReady *function tADTStream.IsReady: boolean; virtual;*

Is the stream ready for write access?

Read *procedure tADTStream.Read(var Data; Count: word); virtual;*

Reads specified amount of data from this stream.

Overrides tStream.Read.

Size *function tADTStream.Size: longint; virtual;*

Returns the streams size (length) in bytes.

Overrides tStream.Size.

Write *procedure tADTStream.Write(var Data; Count: word); virtual;*

Writes specified data to this stream.

Overrides tStream.Write.

tAlignment EFDEF

EFDEF.tAlignment = (LeftAlignment, RightAlignment, CenteredAlignment, JustifiedAlignment);

This type stores the alignment of GUI items and texts. The alignment can be left or right justifications, centered and justified on both left and rights sides.

tAllocation EFMEMORY

EFMEMORY.tAllocation = object(tGenericElement)

This class handles a dynamic memory allocation and provides a basic set of generic operations that be applied on that allocation. This class support DPMI and DOS memory.

Fields and Methods:

public
constructor *Initialize...*
constructor *InitializeEmpty*;
constructor *Duplicate...*
constructor *Compatible...*
procedure *CopyIn...* virtual;
procedure *Clear*; virtual;

Descendants:

tBuffer

Methods

Clear *procedure tAllocation.Clear; virtual;*

Clears the allocation, ie. sets all bytes to zero.

Compatible *constructor tAllocation.Compatible(Allocation: pAllocation);*

Initializes and constructs a compatible instance.

CopyIn *procedure tAllocation.CopyIn(Source: pointer; Count, Position: word); virtual;*

Copies the specified contents into this element.

Overrides tElement.CopyIn.

Duplicate *constructor tAllocation.Duplicate(Allocation: pAllocation);*

Initializes and duplicates an instance of this class.

Overrides tGenericElement.Duplicate.

Initialize *constructor tAllocation.Initialize(SizeOfData: word);*

Initializes and constructs an instance of this type.

Overrides tGenericElement.Initialize.

Overrides tObject.Initialize.

InitializeEmpty *constructor tAllocation.InitializeEmpty;*

Initializes and constructs an empty instance of this type.

Overrides tGenericElement.InitializeEmpty.

Tan EFMATH

function EFMATH.Tan(X: tBaseReal): tBaseReal;

Returns the tangent of the given angle.

Tanh EFMATH

function EFMATH.Tanh(X: tBaseReal): tBaseReal;

Returns the hyperbolic tangent of the given angle.

tApplication EFAPP

EFAPP.tApplication = object(tManager)

This is the application component class. The tApplication class manages the application event queue and should be the parent component for component networks.

Fields and Methods:

public
constructor *Initialize*;
destructor *Intercept*; virtual;

Methods

Initialize *constructor tApplication.Initialize;*

Initializes and constructs an instance of this type.

Overrides tManager.Initialize.

Overrides tComponent.Initialize.

Overrides tMessage.Initialize.

Overrides tObject.Initialize.

Intercept *destructor tApplication.Intercept; virtual;*

Intercepts and destructs an instance of this type.

Overrides tManager.Intercept.

Overrides tElement.Intercept.

Overrides tObject.Intercept.

tArchive EFRES

EFRES.tArchive = object(tResource)

This class defines an archive, that is a *tResource* that exclusively contain *tStreamableFile* instances. An archive provide two important methods: *Add* and *Extract*. Both accept wildcards and return TRUE to indicate success and FALSE to announce an failure. FALSE is returned if no file could be processed or an error occured.

Fields and Methods:

public
function *Add...* virtual;
function *Extract...* virtual;

Methods

Add *function tArchive.Add(Filename: string): boolean; virtual;*

Attempts to add the specified file to the archive.

Extract *function tArchive.Extract(Wildcard: string): boolean; virtual;*

Attempts to extract filenames that match the wildcard.

tArrangement EFVIEWS

EFVIEWS.tArrangement = (TileArrange, CascadeArrange, MaximizeArrange, MimimizeArrange);

tArray EFARRAY

EFARRAY.tArray = object(tLinearADT)

This class defines an array ADT, ie. a random access structure with a fixed number of elements, indexed by integer keys. Arrays are useful when the number of items to be maintained by an ADT is known in advance or can at least be bounded. Arrays are also important when the ability to rapdily access arbitrary elements is important, since individual elements can be directly indexed (random access). The elements in an array is usually assumed to be unsequenced.

Common properties of arrays:

(1) An array can have empty (non-used) elements. *Get* returns NIL for empty elements.

(2) Arrays have cursors, that is one element is marked as the current element. Store automatically places the element after the cursor, and may overwrite an existing element.

(3) *Elements* = the number of elements inside the array.

(4) *Capacity* = the maximum number of elements that can be stored inside the array (including un-used slots).

Fields and Methods:

public
constructor *Initialize...*
constructor *Resemble...*
constructor *Duplicate...*
destructor *Intercept*; virtual;

procedure *SetCursor...* virtual;
procedure *Clear*; virtual;
procedure *Put...* virtual;
function *Get...* virtual;
procedure *SetElement...* virtual;
procedure *Update...* virtual;
procedure *Erase...* virtual;
procedure *Insert...* virtual;
procedure *Swap...* virtual;
procedure *Adjust*; virtual;
procedure *Touch...* virtual;
procedure *Resize...* virtual;
procedure *Defragment*; virtual;
procedure *Move...* virtual;
function *CreateIterator...* virtual;
function *CreateAllocation...* virtual;
constructor *StreamLoad...*
procedure *StreamBuild...* virtual;
function *Elements...* virtual;
function *Capacity...* virtual;
function *PackageSize...* virtual;
function *High...* virtual;
function *Low...* virtual;
function *IsAllocated...* virtual;
function *IsFree...* virtual;
function *IsValid...* virtual;
private
fAllocation: *pAllocation*;
fCursor: word;

Descendants:

tBoundedArray
tVirtualArray

Fields

fAllocation *tArray.fAllocation: pAllocation;*

Pointer to the data allocation that holds elements.

fCursor *tArray.fCursor: word;*

Cursor index: the current element.

Methods

Adjust *procedure tArray.Adjust; virtual;*

Adjust the element capacity (grow or shrink)

Capacity *function tArray.Capacity: word; virtual;*

Returns the storage capacity in number of elements.

Overrides tADT.Capacity.

Clear *procedure tArray.Clear; virtual;*

Clears all elements inside data type.

Overrides tADT.Clear.

CreateAllocation *function tArray.CreateAllocation(NumberOfElements: word): pAllocation; virtual;*

Creates an element allocation for current properties.

CreateIterator *function tArray.CreateIterator: pIterator; virtual;*

Creates an instance of the iterator class.

Overrides tADT.CreateIterator.

Defragment *procedure tArray.Defragment; virtual;*

Defragments the array (moves elements to the front).

Duplicate *constructor tArray.Duplicate(ADT: pLinearADT);*

Initializes a duplicate ADT instance (with equal elements).

Elements *function tArray.Elements: word; virtual;*

Returns the number of elements that currently are used.

Overrides tADT.Elements.

Erase *procedure tArray.Erase(Index: word); virtual;*

Erases an element from the data structure.

Overrides tLinearADT.Erase.

Get *function tArray.Get(Index: word): pElement; virtual;*

Gets the specified element instance.

Overrides tLinearADT.Get.

High *function tArray.High: word; virtual;*

Returns index for the highest used element.

Overrides tLinearADT.High.

Initialize *constructor tArray.Initialize(NumberOfElements, SizeOfElements: word);*

Initializes an instance of this ADT class.

Overrides tLinearADT.Initialize.

Overrides tADT.Initialize.

Overrides tObject.Initialize.

Insert *procedure tArray.Insert(var Data; Index: word); virtual;*

Inserts an element after the specified index.

Overrides tLinearADT.Insert.

Intercept *destructor tArray.Intercept; virtual;*

Intercepts and destructs an instance of this type.

Overrides tADT.Intercept.

Overrides tObject.Intercept.

IsAllocated *function tArray.IsAllocated: boolean; virtual;*

Is this ADT allocated, that is ready for use?

Overrides tADT.IsAllocated.

IsFree *function tArray.IsFree: boolean; virtual;*

Does this ADT provide storage for at least another element?

Overrides tADT.IsFree.

IsValid *function tArray.IsValid(Index: word): boolean; virtual;*

Is the specified element index valid?

Overrides tLinearADT.IsValid.

Low *function tArray.Low: word; virtual;*

Returns index for the lowest used element.

Overrides tLinearADT.Low.

Move *procedure tArray.Move(Start, Stop: word; Steps: integer); virtual;*

Moves a group of elements in a direction.

PackageSize *function tArray.PackageSize: word; virtual;*

Returns the size of element packaging (container).

Overrides tLinearADT.PackageSize.

Overrides tADT.PackageSize.

Put *procedure tArray.Put(Element: pElement); virtual;*

Stores a new element (tElement instance) to this ADT.

Overrides tADT.Put.

Resemble *constructor tArray.Resemble(ADT: pLinearADT);*

Initializes an ADT that resembles the specified ADT.

Resize *procedure tArray.Resize(NumberOfElements: word); virtual;*

Resizes the array and truncates elements if needed.

SetCursor *procedure tArray.SetCursor(Index: word); virtual;*

Sets the cursor element.

SetElement *procedure tArray.SetElement(Element: pElement; Index: word); virtual;*

Sets the element instance at some index.

Overrides tLinearADT.SetElement.

StreamBuild *procedure tArray.StreamBuild(NumberOfElements: word); virtual;*

Builds this instance after properties have been loaded.

Overrides tADT.StreamBuild.

StreamLoad *constructor tArray.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

Overrides tADT.StreamLoad.

Overrides tObject.StreamLoad.

Swap *procedure tArray.Swap(A, B: word); virtual;*

Swaps places of two elements.

Overrides tLinearADT.Swap.

Touch *procedure tArray.Touch(Index: word); virtual;*

Touch an element (called an element has been updated).

Update *procedure tArray.Update(var Data; Index: word); virtual;*

Updates an element (specified by an index value).

Overrides tLinearADT.Update.

EFTREE.tAVLLinkage = object(tTreeLinkage)

This class defines an AVL tree node, that is an node that is automatically balanced in the tree.

Fields and Methods:

public
constructor *Initialize...*
function *SingleRotateLeft...* virtual;
function *SingleRotateRight...* virtual;
procedure *SetBalance...* virtual;
function *Balance...* virtual;
private
fBalance: shortint;

Fields

fBalance *tAVLLinkage.fBalance: shortint;*

Balance factor for this node (internal).

Methods

Balance *function tAVLLinkage.Balance: longint; virtual;*

Difference in height between first and last children.

Overrides tTreeLinkage.Balance.

Initialize *constructor tAVLLinkage.Initialize(HoldElement: pElement; ParentNode: pTreeLinkage);*

Initializes a new node and attachs it to a parent node.

Overrides tTreeLinkage.Initialize.

Overrides tLinkage.Initialize.

Overrides tCarrier.Initialize.

Overrides tContainer.Initialize.

Overrides tObject.Initialize.

SetBalance *procedure tAVLLinkage.SetBalance(NodeBalance: shortint); virtual;*

Sets the balance of this AVL tree node.

SingleRotateLeft *function tAVLLinkage.SingleRotateLeft: pTreeLinkage; virtual;*

Performs a single rotation of this node, returns root.

SingleRotateRight *function tAVLLinkage.SingleRotateRight: pTreeLinkage; virtual;*

Performs a single rotation of this node, returns root.

tAVLTree EFTREE

EFTREE.tAVLTree = object(tBinaryTree)

This class defines a Adelson-Velskii-Landis binary tree. The AVL automatically maintains a balanced tree.

Fields and Methods:

procedure *Put...* virtual;
procedure *Erase...* virtual;

Methods

Erase *procedure tAVLTree.Erase(var Key; KeySize: word); virtual;*

Erases element(s) with the specified search key.

Overrides tOrderedADT.Erase.

Put *procedure tAVLTree.Put(Element: pElement); virtual;*

Stores a new element (tElement instance) to this ADT.

Overrides tADT.Put.

tBaseReal EFMATH

EFMATH.tBaseReal = extended;

Base real type for mathematical components. Two libraries compiled with different tBaseReal settings may not be entirely compatible.

tBinaryTree EFTREE

EFTREE.tBinaryTree = object(tOrderedADT)

This class defines a binary tree data structure, that is a *tTree* with at the most two children for each node.

Fields and Methods:

Descendants:

tAVLTree
tSplayTree

tBit EFDEF

EFDEF.tBit = 0..1;

tBorder EFDEF

EFDEF.tBorder = (SingleBorder, DoubleBorder, SingleTopBorder, DoubleTopBorder, SolidBorder, ThinBorder, NoBorder);

This type contains the border that is assigned to an item. Only standard borders are available.

tBoundedArray EFARRAY

EFARRAY.tBoundedArray = object(tArray)

This class defines bounded array, that is an arrays knows the location of the first and last used element, and guarantee that there are no unused element between these two elements, that is, a bounded array is never fragmented.

Bounded arrays are hence sequential arrays, and so are virtual arrays that descend from this class (see *tVirtualArray*).

Fields and Methods:

constructor *Initialize...*
constructor *Resemble...*
constructor *Duplicate...*
procedure *SetBoundaries...* virtual;
procedure *Clear*; virtual;
procedure *Touch...* virtual;
procedure *Resize...* virtual;
procedure *Defragment*; virtual;
procedure *Move...* virtual;
constructor *StreamLoad...*
procedure *StreamLoadProperties...* virtual;
procedure *StreamStoreProperties...* virtual;
function *Elements...* virtual;
function *High...* virtual;
function *Low...* virtual;
private
fStart,
fStop: word;

Descendants:

tVirtualArray

Fields

fStart *tBoundedArray.fStart,*

fStop: word;

Boundary index values (elements must be inside them).

fStop *See fStart*

Methods

Clear *procedure tBoundedArray.Clear; virtual;*

Clears all elements inside data type.

Overrides tArray.Clear.

Overrides tADT.Clear.

Defragment *procedure tBoundedArray.Defragment; virtual;*

Defragments the array (moves elements to the front).

Overrides tArray.Defragment.

Duplicate *constructor tBoundedArray.Duplicate(ADT: pLinearADT);*

Initializes a duplicate ADT instance (with equal elements).

Overrides tArray.Duplicate.

Elements *function tBoundedArray.Elements: word; virtual;*

Returns the number of elements that currently are used.

Overrides tArray.Elements.

Overrides tADT.Elements.

High *function tBoundedArray.High: word; virtual;*

Returns index for the highest used element.

Overrides tArray.High.

Overrides tLinearADT.High.

Initialize *constructor tBoundedArray.Initialize(NumberOfElements, SizeOfElements: word);*

Initializes an instance of this ADT class.

Overrides tArray.Initialize.

Overrides tLinearADT.Initialize.

Overrides tADT.Initialize.

Overrides tObject.Initialize.

Low *function tBoundedArray.Low: word; virtual;*

Returns index for the lowest used element.

Overrides tArray.Low.

Overrides tLinearADT.Low.

Move *procedure tBoundedArray.Move(Start, Stop: word; Steps: integer); virtual;*

Moves a group of elements in a direction.

Overrides tArray.Move.

Resemble *constructor tBoundedArray.Resemble(ADT: pLinearADT);*

Initializes an ADT that resembles the specified ADT.

Overrides tArray.Resemble.

Resize *procedure tBoundedArray.Resize(NumberOfElements: word); virtual;*

Resizes the array and truncates elements if needed.

Overrides tArray.Resize.

SetBoundaries *procedure tBoundedArray.SetBoundaries(Start, Stop: word); virtual;*

Sets upper and lower boundary for elements in this array.

StreamLoad *constructor tBoundedArray.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

Overrides tArray.StreamLoad.

Overrides tADT.StreamLoad.

Overrides tObject.StreamLoad.

StreamLoadProperties *procedure tBoundedArray.StreamLoadProperties(Stream: pStream); virtual;*

Loads properties of this ADT from a stream (not elements).

Overrides tADT.StreamLoadProperties.

StreamStoreProperties *procedure tBoundedArray.StreamStoreProperties(Stream: pStream); virtual;*

Stores properties of this ADT to a stream (not elements).

Overrides tADT.StreamStoreProperties.

Touch *procedure tBoundedArray.Touch(Index: word); virtual;*

Touch an element (called an element has been updated).

Overrides tArray.Touch.

tBoundedConsole EFCONDRV

EFCONDRV.tBoundedConsole = object(tConsole)

This is an abstract class that defines the properties of a console with boundaries, that is, with upper-left and lower-right bounds.

Fields and Methods:

public
constructor *Initialize*;
destructor *Intercept*; virtual;
constructor *StreamLoad...*
procedure *StreamStore...* virtual;
function *Width...* virtual;
function *Height...* virtual;
private
fBoundaries: *pField*;

Descendants:

tCRT
  EFSCREEN.tScreen
tVirtualConsole
  tImage
  EFSCREEN.tBufferedScreen

Fields

fBoundaries *tBoundedConsole.fBoundaries: pField;*

Boundaries for the current CRT mode.

Methods

Height *function tBoundedConsole.Height: word; virtual;*

Returns the height of the console in coordinate resolution.

Overrides tConsole.Height.

Initialize *constructor tBoundedConsole.Initialize;*

Initializes a console with the default properties.

Overrides tConsole.Initialize.

Overrides tComponent.Initialize.

Overrides tMessage.Initialize.

Overrides tObject.Initialize.

Intercept *destructor tBoundedConsole.Intercept; virtual;*

Intercepts and destructs an instance of this type.

Overrides tElement.Intercept.

Overrides tObject.Intercept.

StreamLoad *constructor tBoundedConsole.StreamLoad(Stream: pStream);*

Constructs and loads an instance from a stream.

Overrides tConsole.StreamLoad.

Overrides tComponent.StreamLoad.

Overrides tStaticElement.StreamLoad.

Overrides tElement.StreamLoad.

Overrides tObject.StreamLoad.

StreamStore *procedure tBoundedConsole.StreamStore(Stream: pStream); virtual;*

Stores an instance to a stream.

Overrides tConsole.StreamStore.

Overrides tComponent.StreamStore.

Overrides tStaticElement.StreamStore.

Overrides tElement.StreamStore.

Overrides tObject.StreamStore.

Width *function tBoundedConsole.Width: word; virtual;*

Returns the width of the console in coordinate resolution.

Overrides tConsole.Width.

tBoundedKeyPlug EFADT

EFADT.tBoundedKeyPlug = object(tKeyPlug)

Key plug class: this class provides a mechanism of assigning search keys to elements. tBoundedKeyPlug considers a bounded region of elements as the search key. The Compare method uses the same region to compare elements.

Use tBoundedKeyPlug to force ADT′s to ignore *tElement.Compare* method and instead compare a bounded region of the element.

Also, tBoundedKeyPlug should be used whenever an ordered ADT requires search keys that directly correspond to a region of an arbitrary element.

Fields and Methods:

public
constructor *Initialize...*
procedure *SetBoundaries...*
function *CompareElements...* virtual;
function *CompareKey...* virtual;
function *CreateKey...* virtual;
constructor *StreamLoad...*
procedure *StreamStore...* virtual;
function *IsEqual...* virtual;
private
fStart,
fStop: word;

Fields

fStart *tBoundedKeyPlug.fStart,*

fStop: word;

Boundaries of the key (region of an arbitrary element).

fStop *See fStart*

Methods

CompareElements *function tBoundedKeyPlug.CompareElements(A, B: pElement): shortint; virtual;*

Compares element A with element B and returns [-1, 0, 1].

Overrides tKeyPlug.CompareElements.

CompareKey *function tBoundedKeyPlug.CompareKey(var A; B: pElement): shortint; virtual;*

Compares the key A with the element B. Returns [-1,0,1].

Overrides tKeyPlug.CompareKey.

CreateKey *function tBoundedKeyPlug.CreateKey(A: pElement): pAllocation; virtual;*

Creates a memory allocation with the key.

Overrides tKeyPlug.CreateKey.

Initialize *constructor tBoundedKeyPlug.Initialize(PlugManager: pPlugManager; Start, Stop: word);*

Initializes a bounded key plug (a key region).

Overrides tKeyPlug.Initialize.

Overrides tPlug.Initialize.

Overrides tObject.Initialize.

IsEqual *function tBoundedKeyPlug.IsEqual(Instance: pObject): boolean; virtual;*

Are contents of these instances equal (= boundaries equal)?

Overrides tElement.IsEqual.

Overrides tObject.IsEqual.

SetBoundaries *procedure tBoundedKeyPlug.SetBoundaries(Start, Stop: word);*

Set boundaries of the key region in an arbitrary element.

StreamLoad *constructor tBoundedKeyPlug.StreamLoad(Stream: pStream);*

Constructs and loads an instance from a stream.

Overrides tKeyPlug.StreamLoad.

Overrides tPlug.StreamLoad.

Overrides tStaticElement.StreamLoad.

Overrides tElement.StreamLoad.

Overrides tObject.StreamLoad.

StreamStore *procedure tBoundedKeyPlug.StreamStore(Stream: pStream); virtual;*

Stores an instance to a stream.

Overrides tStaticElement.StreamStore.

Overrides tElement.StreamStore.

Overrides tObject.StreamStore.

tBubbleSortPlug EFSORT

EFSORT.tBubbleSortPlug = object(tSortPlug)

Bubble sort plug class derived from *tSortPlug*. This class provides the Bubble sort algorithm. Bubble sort has limited performance and is very simple. Consider *tQuickSortPlug* if performance is important. Complexity: O(N) - O(N^2).

Fields and Methods:

public
constructor *Initialize...*
procedure *Sort...* virtual;
constructor *StreamLoad...*

Methods

Initialize

constructor tBubbleSortPlug.Initialize(PlugManager: pPlugManager);

Constructs a plug instance for the specified manager.

Overrides tPlug.Initialize.

Overrides tObject.Initialize.

Sort

procedure tBubbleSortPlug.Sort(Start, Stop: word; Order: tSortOrder); virtual;

Sorts elements within the specified interval.

Overrides tSortPlug.Sort.

StreamLoad

constructor tBubbleSortPlug.StreamLoad(Stream: pStream);

Constructs and loads an instance from a stream.

Overrides tSortPlug.StreamLoad.

Overrides tPlug.StreamLoad.

Overrides tStaticElement.StreamLoad.

Overrides tElement.StreamLoad.

Overrides tObject.StreamLoad.

tBucket EFTABLE

EFTABLE.tBucket = object(tStaticElement)

This class defines a bucket for *tBucketHashTable*. A bucket is an element that contain an ordered ADT with some elements. Buckets increase the performance of hash tables since collisions effectively avoided. See *tBucketHashTable* for details.

Fields and Methods:

public
constructor *Initialize...*
destructor *Intercept*; virtual;
constructor *StreamLoad...*
procedure *StreamStore...* virtual;
function *ADT...* virtual;
private
fBaseADT: *pOrderedADT*;

Fields

fBaseADT *tBucket.fBaseADT: pOrderedADT;*

Contents of this bucket (ordered ADT).

Methods

ADT *function tBucket.ADT: pOrderedADT; virtual;*

Provides direct access to the ordered ADT.

Initialize *constructor tBucket.Initialize(BaseADT: pOrderedADT);*

Construct a bucked based on the specified ordered ADT.

Overrides tObject.Initialize.

Intercept *destructor tBucket.Intercept; virtual;*

Intercepts the bucket including all elements.

Overrides tElement.Intercept.

Overrides tObject.Intercept.

StreamLoad *constructor tBucket.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

Overrides tStaticElement.StreamLoad.

Overrides tElement.StreamLoad.

Overrides tObject.StreamLoad.

StreamStore *procedure tBucket.StreamStore(Stream: pStream); virtual;*

Stores an instance to a stream.

Overrides tStaticElement.StreamStore.

Overrides tElement.StreamStore.

Overrides tObject.StreamStore.

EFTABLE.tBucketHashTable = object(tHashTable)

This class defines an hash table with a bucket mechanism. A hash table is a generalization of an array into an ordered ADT. For details, see *tHashTable*.

This hash table uses buckets to prevent collisions. Each bucket is itself a collection of elements. There can hence be artibrary many elements associated to a specific hash code. The hash function is used to determine which of the buckets the element is stored in. The bucket ADT supply the search mechanism.

Fields and Methods:

public
constructor *Initialize...*
procedure *Erase...* virtual;
procedure *Put...* virtual;
function *Get...* virtual;
function *CreateIterator...* virtual;
function *CreateBucket...* virtual;
constructor *StreamLoad...*
function *Elements...* virtual;

Methods

CreateBucket

function tBucketHashTable.CreateBucket: pBucket; virtual;

Creates an instance of the bucket class.

CreateIterator

function tBucketHashTable.CreateIterator: pIterator; virtual;

Creates an instance of the iterator class.

Overrides tCompositeADT.CreateIterator.

Overrides tADT.CreateIterator.

Elements

function tBucketHashTable.Elements: word; virtual;

Returns the number of elements that currently are used.

Overrides tCompositeADT.Elements.

Overrides tADT.Elements.

Erase *procedure tBucketHashTable.Erase(var Key; KeySize: word); virtual;*

Erases element(s) with the specified search key.

Overrides tCompositeADT.Erase.

Overrides tOrderedADT.Erase.

Get *function tBucketHashTable.Get(var Key; KeySize: word): pElement; virtual;*

Gets the specified element instance.

Overrides tCompositeADT.Get.

Overrides tOrderedADT.Get.

Initialize *constructor tBucketHashTable.Initialize(CapacityOfElements, SizeOfElements: word);*

Initializes an instance of this ADT class.

Overrides tHashTable.Initialize.

Overrides tCompositeADT.Initialize.

Overrides tADT.Initialize.

Overrides tObject.Initialize.

Put *procedure tBucketHashTable.Put(Element: pElement); virtual;*

Stores a new element (tElement instance) to this ADT.

Overrides tCompositeADT.Put.

Overrides tADT.Put.

StreamLoad *constructor tBucketHashTable.StreamLoad(Stream: pStream);*

Loads an ADT from a stream.

Overrides tCompositeADT.StreamLoad.

Overrides tADT.StreamLoad.

Overrides tObject.StreamLoad.

EFTABLE.tBucketIterator = object(tIterator)

This class implements a bucket iterator, that is an iterator that walks between elements in *tBucketHashTable*.

Fields and Methods:

public
constructor *Initialize*...
destructor *Intercept*; virtual;
procedure *First*; virtual;
procedure *Last*; virtual;
procedure *ResetBucket*; virtual;
function *Element*... virtual;
function *Position*... virtual;
procedure *WalkForward*; virtual;
procedure *WalkBackward*; virtual;
function *IsAllocated*... virtual;
function *IsEnd*... virtual;
function *IsValid*... virtual;
function *IsEqual*... virtual;
function *IsCompatible*... virtual;
private
fIterator: *pIterator*;
fBucketIterator: *pIterator*;
fPosition: word;

Fields

fBucketIterator *tBucketIterator.fBucketIterator: pIterator;*

Iterator inside the current bucket (an ordered ADT).

fIterator *tBucketIterator.fIterator: pIterator;*

Iterator for the buckets (inside the hash table).

fPosition *tBucketIterator.fPosition: word;*

Position of the current element relative to the first.

Methods

Element *function tBucketIterator.Element: pElement; virtual;*

Returns a pointer to the current element instance.

Overrides tIterator.Element.

First *procedure tBucketIterator.First; virtual;*

Walks to the first element in this ADT.

Overrides tIterator.First.

Initialize *constructor tBucketIterator.Initialize(ADT: pBucketHashTable);*

Initializes an iterator for the specified ADT.

Overrides tObject.Initialize.

Intercept *destructor tBucketIterator.Intercept; virtual;*

Intercepts this iterator.

Overrides tObject.Intercept.

IsAllocated *function tBucketIterator.IsAllocated: boolean; virtual;*

Is this iterator assigned to an ADT?

Overrides tIterator.IsAllocated.

IsCompatible *function tBucketIterator.IsCompatible(Instance: pObject): boolean; virtual;*

Are ADT′s compatible, that is has compatible elements.

Overrides tIterator.IsCompatible.

Overrides tObject.IsCompatible.

IsEnd *function tBucketIterator.IsEnd: boolean; virtual;*

Is iterator at boundaries (beginning or end)?

Overrides tIterator.IsEnd.

IsEqual *function tBucketIterator.IsEqual(Instance: pObject): boolean; virtual;*

Are contents of these instances equal (same cursor)?

Overrides tObject.IsEqual.

IsValid *function tBucketIterator.IsValid(ADT: pADT): boolean; virtual;*

Is the specified ADT of a valid class?

Overrides tIterator.IsValid.

Last *procedure tBucketIterator.Last; virtual;*

Walks to the last element in this ADT.

Overrides tIterator.Last.

Position *function tBucketIterator.Position: word; virtual;*

Returns the current position inside the ADT.

Overrides tIterator.Position.

ResetBucket *procedure tBucketIterator.ResetBucket; virtual;*

Resets the iterator for the elements inside current bucket.

WalkBackward *procedure tBucketIterator.WalkBackward; virtual;*

Walk backwards (to the predecessor element).

Overrides tIterator.WalkBackward.

WalkForward *procedure tBucketIterator.WalkForward; virtual;*

Walk forwards (to the successor element).

Overrides tIterator.WalkForward.

tBuffer EFMEMORY

EFMEMORY.tBuffer = object(tAllocation)

This class defines an intelligent memory allocation that can be used for buffering other processes, for example streams. The buffer knows the current position inside the buffer (the cursor) and the used space in the buffer. A buffer does not have to use the entire allocation.

Buffers can be used to transfer data in segments from a stream. The following example shows how this can be done:

while DoneSize <> WantedSize do begin if Buffer.IsFull then Buffer.Reread (Data, Size); Inc (DoneSize, Buffer.Get (@Data, SizeOf(Data))); end;

You are responsible to flush the buffer whenever *IsFull* tells you to do so.

Fields and Methods:

public
constructor *Initialize...*
procedure *SetCursor...* virtual;
procedure *SetUsage...* virtual;
procedure *Reset*; virtual;
procedure *Reread...* virtual;
function *Get...* virtual;
function *Cursor...* virtual;
function *Usage...* virtual;
function *IsFull...* virtual;
private
fCursor: word;
fUsage: word;

Fields

fCursor *tBuffer.fCursor: word;*

Current position in buffer.

fUsage *tBuffer.fUsage: word;*

Last used position in the buffer.

Methods

Cursor *function tBuffer.Cursor: word; virtual;*

Returns the current cursor position.

Get *function tBuffer.Get(Target: pointer; Count: word): word; virtual;*

Attempts to get the specified size, returns actual size.

Initialize *constructor tBuffer.Initialize(SizeOfBuffer: word);*

Initializes a buffer with the specified size in bytes.

Overrides tAllocation.Initialize.

Overrides tGenericElement.Initialize.

Overrides tObject.Initialize.

IsFull *function tBuffer.IsFull: boolean; virtual;*

Is buffer full, ie. last position in buffer reached?

Reread *procedure tBuffer.Reread(Source: pointer; Count: word); virtual;*

Rereads the buffer and resets the cursor and size.

Reset *procedure tBuffer.Reset; virtual;*

Resets the cursor position inside the buffer.

SetCursor *procedure tBuffer.SetCursor(Position: word); virtual;*

Sets the cursor positin in the buffer.

SetUsage *procedure tBuffer.SetUsage(UsedSize: word); virtual;*

Sets the used number of bytes in the buffer.

Usage *function tBuffer.Usage: word; virtual;*

Returns the used number of bytes in this buffer.

tBufferedScreen EFSCREEN

EFSCREEN.tBufferedScreen = object(tVirtualConsole)

Fields and Methods:

public
private
fBase: *pConsole*;
fBuffer: *pImage*;

Fields

fBase *tBufferedScreen.fBase: pConsole;*

The buffered console instance.

fBuffer *tBufferedScreen.fBuffer: pImage;*

The buffer image instance.

tBufferFilter EFFILTER

EFFILTER.tBufferFilter = object(tFilter)

This is an abstract class definition for a buffered filter. This filter can be connected to an arbitrary stream that permit random acess, normally a file, to improve performance.

Buffered streams operate on their data via an arbitrary large memory buffer (*tAllocation* descendant), and are generally much faster than non-buffered streams, especially when a large number of small data transfers are requested.

Fields and Methods:

public
constructor *Initialize...*
destructor *Intercept*; virtual;
procedure *SetMode...* virtual;
procedure *SetBufferMode...*
procedure *Read...* virtual;
procedure *Write...* virtual;
function *GetString...* virtual;
procedure *PutByte...* virtual;
function *GetByte...* virtual;
procedure *Flush*; virtual;
procedure *Truncate*; virtual;
procedure *Seek...* virtual;
function *Buffer...* virtual;
function *BufferSize...* virtual;
function *IsAllocated...* virtual;
function *IsBuffered...* virtual;
function *IsFull...* virtual;
private
fAllocation: *pAllocation*;
fBufferStart: longint;
fBufferStop: longint;
fBufferMode: *tAccess*;

Descendants:

EFFILE.tFile
  EFFILE.tIFile
  EFFILE.tOFile
  EFFILE.tTemporaryFile
  EFRES.tStreamableFile

Fields

fAllocation *tBufferFilter.fAllocation: pAllocation;*

Data allocation for the stream buffer.

fBufferMode *tBufferFilter.fBufferMode: tAccess;*

Buffer access permittance (read or write).

fBufferStart *tBufferFilter.fBufferStart: longint;*

Start of the used block in the buffer.

fBufferStop *tBufferFilter.fBufferStop: longint;*

End of the used block in the buffer.

Methods

Buffer *function tBufferFilter.Buffer(Where: word): pointer; virtual;*

Pointer to buffer byte.

BufferSize *function tBufferFilter.BufferSize: word; virtual;*

Returns size of buffer.

Flush *procedure tBufferFilter.Flush; virtual;*

Flushes any buffer.

Overrides tFilter.Flush.

Overrides tStream.Flush.

GetByte *function tBufferFilter.GetByte: byte; virtual;*

Gets a byte from stream.

Overrides tStream.GetByte.

GetString *function tBufferFilter.GetString(Delimeter: string): string; virtual;*

Gets a text string from the stream, separated by delimeter.

Overrides tFilter.GetString.

Overrides tStream.GetString.

Initialize *constructor tBufferFilter.Initialize(Stream: pStream; SizeOfBuffer: word);*

Initializes an instance with specified buffer size.

Overrides tFilter.Initialize.

Overrides tStream.Initialize.

Overrides tObject.Initialize.

Intercept *destructor tBufferFilter.Intercept; virtual;*

Intercepts and destructs an instance of this class.

Overrides tFilter.Intercept.

Overrides tStream.Intercept.

Overrides tObject.Intercept.

IsAllocated *function tBufferFilter.IsAllocated: boolean; virtual;*

Is stream allocated (and buffer initialized)?

Overrides tFilter.IsAllocated.

Overrides tStream.IsAllocated.

IsBuffered *function tBufferFilter.IsBuffered: boolean; virtual;*

Is this a buffered stream with a non-zero buffer?

IsFull *function tBufferFilter.IsFull: boolean; virtual;*

Is buffer full, ie. last position in buffer reached?

PutByte *procedure tBufferFilter.PutByte(Data: byte); virtual;*

Puts a byte into stream.

Overrides tStream.PutByte.

Read *procedure tBufferFilter.Read(var Data; Count: word); virtual;*

Reads data from stream.

Overrides tFilter.Read.

Overrides tStream.Read.

Seek *procedure tBufferFilter.Seek(Where: longint); virtual;*

Moves to a position.

Overrides tFilter.Seek.

Overrides tStream.Seek.

SetBufferMode *procedure tBufferFilter.SetBufferMode(Access: tAccess);*

Sets the buffer access mode (either read or write mode).

SetMode *procedure tBufferFilter.SetMode(Access: tAccess); virtual;*

Sets access mode, ie. the way the stream can be accessed.

Overrides tFilter.SetMode.

Overrides tStream.SetMode.

Truncate *procedure tBufferFilter.Truncate; virtual;*

Truncates the stream, ie. deletes remaining data.

Overrides tFilter.Truncate.

Overrides tStream.Truncate.

Write *procedure tBufferFilter.Write(var Data; Count: word); virtual;*

Writes data to stream.

Overrides tFilter.Write.

Overrides tStream.Write.

tByteArray EFDEF

EFDEF.tByteArray = array[byte] of byte;

tCarrier EFKERNEL

EFKERNEL.tCarrier = object(tContainer)

This is instance carrier - a tContainer descendant that enables arbitrary instances to be linked in a doubly linked data structures.

Fields and Methods:

public
constructor *Initialize...*
destructor *Intercept*; virtual;
procedure *Take...* virtual;
procedure *Leave...* virtual;
constructor *StreamLoad...*
procedure *StreamStore...* virtual;
procedure *StreamWrite...* virtual;
function *Content...* virtual;
function *IsAllocated...* virtual;
function *IsValid...* virtual;
private
fInstance: *pObject*;

Descendants:

EFLIST.tLinkage
  EFTREE.tTreeLinkage
    EFTREE.tAVLLinkage

Fields

fInstance *tCarrier.fInstance: pObject;*

Pointer to the owned instance.

Methods

Content *function tCarrier.Content: pObject; virtual;*

Returns the content of this carrier - or NIL.

Initialize *constructor tCarrier.Initialize(Instance: pObject; Where: pContainer);*

Initializes a container that carries the given instance.

Overrides tContainer.Initialize.

Overrides tObject.Initialize.

Intercept *destructor tCarrier.Intercept; virtual;*

Intercepts the carrier and the carried instance.

Overrides tContainer.Intercept.

Overrides tObject.Intercept.

IsAllocated *function tCarrier.IsAllocated: boolean; virtual;*

Is this element allocated, ie. has some contents?

IsValid *function tCarrier.IsValid(Instance: pObject): boolean; virtual;*

Is the specified instance valid for this carrier?

Leave *procedure tCarrier.Leave(var Instance: pObject); virtual;*

Leaves the carried instance to the specified variable.

StreamLoad *constructor tCarrier.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

Overrides tObject.StreamLoad.

StreamStore *procedure tCarrier.StreamStore(Stream: pStream); virtual;*

Stores an instance to a stream.

Overrides tObject.StreamStore.

StreamWrite *procedure tCarrier.StreamWrite(Stream: pStream); virtual;*

Writes the content of this element to a stream.

Overrides tObject.StreamWrite.

Take *procedure tCarrier.Take(Instance: pObject); virtual;*

Takes the specified instance into the carrier.

tCarrierElement EFELEM

EFELEM.tCarrierElement = object(tElement)

This class defines a carrier. A carrier is a special kind of *tElement* descendant, designed to carry an arbitrary *tObject* instance. The carried instance is "owned" by the carrier until it "leaves" the instance to some external component. The carrier then becomes empty (IsAllocated = FALSE) until it "takes" an instance. Thus, the carrier has the following properties:

(1) A carrier is either empty or "owns" a tObject instance.

(2) A carrier can "leave" and "take" its instance to and from some other component. It then loses all control and resposibility of that instance and becomes empty.

(3) When a carrier is loaded from or stored to a stream, the "owned" instance is resposible for its own loading and storing.

Fields and Methods:

public
fInstance: *pObject*;
constructor *Initialize...*
constructor *InitializeEmpty*;
function *Data...* virtual;
function *Size...* virtual;
procedure *Dispose*; virtual;
procedure *Take...* virtual;
procedure *Leave...* virtual;
function *Compare...* virtual;
procedure *Swap...* virtual;
constructor *StreamLoad...*
procedure *StreamStore...* virtual;
procedure *StreamWrite...* virtual;
function *IsAllocated...* virtual;
function *IsValid...* virtual;
function *IsEqual...* virtual;

function *IsCompatible...* virtual;

Fields

fInstance *tCarrierElement.fInstance: pObject;*

Pointer to an abitrary, owned instance.

Methods

Compare *function tCarrierElement.Compare(What: pElement): shortint; virtual;*

Compares elements and returns [1, 0, -1] as result.

Overrides tElement.Compare.

Data *function tCarrierElement.Data(Position: word): pointer; virtual;*

Returns a pointer to contents at a position within [0,n].

Overrides tElement.Data.

Dispose *procedure tCarrierElement.Dispose; virtual;*

Disposes any data allocation or erases the contents.

Overrides tElement.Dispose.

Initialize *constructor tCarrierElement.Initialize(Instance: pObject);*

Initializes an element with the specified contents.

Overrides tObject.Initialize.

InitializeEmpty *constructor tCarrierElement.InitializeEmpty;*

Initializes an empty element (not allocated).

IsAllocated *function tCarrierElement.IsAllocated: boolean; virtual;*

Is this element allocated, ie. has some contents?

Overrides tElement.IsAllocated.

IsCompatible *function tCarrierElement.IsCompatible(Instance: pObject): boolean; virtual;*

Are these elements compatible?

Overrides tElement.IsCompatible.

Overrides tObject.IsCompatible.

IsEqual *function tCarrierElement.IsEqual(Instance: pObject): boolean; virtual;*

Are the contents of these elements equal?

Overrides tElement.IsEqual.

Overrides tObject.IsEqual.

IsValid *function tCarrierElement.IsValid(Instance: pObject): boolean; virtual;*

Is the specified instance valid for this carrier?

Leave *procedure tCarrierElement.Leave(var Instance: pObject); virtual;*

Leaves the carried instance to the specified variable.

Size *function tCarrierElement.Size: word; virtual;*

Returns the size of the contents of this element in bytes.

Overrides tElement.Size.

StreamLoad *constructor tCarrierElement.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

Overrides tElement.StreamLoad.

Overrides tObject.StreamLoad.

StreamStore *procedure tCarrierElement.StreamStore(Stream: pStream); virtual;*

Stores an instance to a stream.

Overrides tElement.StreamStore.

Overrides tObject.StreamStore.

StreamWrite *procedure tCarrierElement.StreamWrite(Stream: pStream); virtual;*

Writes the content of this element to a stream.

Overrides tElement.StreamWrite.

Overrides tObject.StreamWrite.

Swap *procedure tCarrierElement.Swap(What: pElement); virtual;*

Swaps contents of current element with the specified.

Overrides tElement.Swap.

Take *procedure tCarrierElement.Take(Instance: pObject); virtual;*

Takes the specified instance into the carrier.

tCarrierRegister EFKERNEL

EFKERNEL.tCarrierRegister = object(tClassRegister)

This is a class register - that is a register with some owned instances. *tCarrierRegister* is based on *tCarrier* instances. tRegister in EFREG provides a more sophisticated mechanism using a tree ADT.

Fields and Methods:

public
fRegister: *pCarrier*;
constructor *Initialize*;
destructor *Intercept*; virtual;
procedure *Register...* virtual;
function *Release...* virtual;
function *ReleaseClass...* virtual;
function *Search...* virtual;
constructor *StreamLoad...*
procedure *StreamStore...* virtual;
function *First...* virtual;
function *Last...* virtual;
function *Instances...* virtual;
function *IsRegistered...* virtual;
function *IsClassRegistered...* virtual;
function *IsValid...* virtual;

Descendants:

tErrorHandler

Fields

fRegister *tCarrierRegister.fRegister: pCarrier;*

Pointer to linked instances in this register.

Methods

First *function tCarrierRegister.First: word; virtual;*

Returns the lowest registered identity (zero if empty).

Overrides tClassRegister.First.

Initialize *constructor tCarrierRegister.Initialize;*

Constructs an empty class register.

Overrides tObject.Initialize.

Instances *function tCarrierRegister.Instances: word; virtual;*

Returns the number of registered instances.

Overrides tClassRegister.Instances.

Intercept *destructor tCarrierRegister.Intercept; virtual;*

Intercepts a class register and all registered instances.

Overrides tObject.Intercept.

IsClassRegistered *function tCarrierRegister.IsClassRegistered(Class: pointer): boolean; virtual;*

Is an instance of this class or a subclass registered?

Overrides tClassRegister.IsClassRegistered.

IsRegistered *function tCarrierRegister.IsRegistered(IdentityOfInstance: word): boolean; virtual;*

Is there an instance with the specified identity?

Overrides tClassRegister.IsRegistered.

IsValid *function tCarrierRegister.IsValid(What: pObject): boolean; virtual;*

Can this instance (class) be registered - is it valid?

Overrides tClassRegister.IsValid.

Last *function tCarrierRegister.Last: word; virtual;*

Returns the highest registered identity (zero if empty).

Overrides tClassRegister.Last.

Register *procedure tCarrierRegister.Register(What: pObject); virtual;*

Registers an instance, that is, puts it into the register.

Overrides tClassRegister.Register.

Release *function tCarrierRegister.Release(IdentityOfInstance: word): pObject; virtual;*

Releases the instance with this identity - or returns NIL.

Overrides tClassRegister.Release.

ReleaseClass *function tCarrierRegister.ReleaseClass(Class: pointer): pObject; virtual;*

Releases a descendant from this class - or returns NIL.

Overrides tClassRegister.ReleaseClass.

Search *function tCarrierRegister.Search(Identity: longint; Class: pointer): pCarrier; virtual;*

Searchs for the carrier of the specified identity / class.

StreamLoad *constructor tCarrierRegister.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

Overrides tObject.StreamLoad.

StreamStore *procedure tCarrierRegister.StreamStore(Stream: pStream); virtual;*

Stores an instance to a stream.

Overrides tObject.StreamStore.

tCaster EFMATH

EFMATH.tCaster = object(tObject)

This class provides a single method for type-casting of a mathematical object to another mathematical object, truncating the source object if needed. Both source and target must be two instances of mathematical objects that have been initialized. Example: real ==> complex number. Extend this class and replace the *Caster* instance in EFMATH, if you add new mathematical objects to EFLIB and want type-casting capabilities.

Fields and Methods:

public
constructor *Initialize*;
function *Cast...* virtual;

Methods

Cast *function tCaster.Cast(Source: pMathObject; Target: pMathObject): boolean; virtual;*

Casts the specified math objects to another type.

Initialize *constructor tCaster.Initialize;*

Initializes an instance of this class.

Overrides tObject.Initialize.

tCharacters EFBASIC

EFBASIC.tCharacters = object(tSet)

This is a character set, ie. a set table of characters there each character (1 .. 256) either exist or doesn't exist. This class requires only 16 bytes of memory (in streams).

Fields and Methods:

public
constructor *Initialize*;
procedure *Insert...*
procedure *Remove...*
function *IsInside...*
function *IsOneInside...*

Methods

Initialize *constructor tCharacters.Initialize;*

Initializes and constructs an instance of this object type.

Overrides tSet.Initialize.

Overrides tObject.Initialize.

Insert *procedure tCharacters.Insert(Characters: string);*

Inserts a string with characters into the character set.

IsInside *function tCharacters.IsInside(Characters: string): boolean;*

Are all characters inside this character set?

IsOneInside *function tCharacters.IsOneInside(Characters: string): boolean;*

Is at least one character inside this character set?

Remove *procedure tCharacters.Remove(Characters: string);*

Removes a string with characters from the character set.

tCharArray EFDEF

EFDEF.tCharArray = array[char] of char;

tChars EFDEF

EFDEF.tChars = set of char;

tCircularQueue EFDATA

EFDATA.tCircularQueue = object(tQueue)

This class implements a circular queue. This class behaves like *tQueue* except for the method *Dequeue*. When this method is called, the element is not removed, but moved to the first position in the queue - it is recycled.

Fields and Methods:

public
constructor *Initialize...*
constructor *Resemble...*
constructor *Duplicate...*
procedure *Dequeue...* virtual;
constructor *StreamLoad...*

Methods

Dequeue *procedure tCircularQueue.Dequeue(var Data); virtual;*

Returns the last (oldest) element in the queue.

Overrides tQueue.Dequeue.

Duplicate *constructor tCircularQueue.Duplicate(ADT: pADT);*

Initializes a duplicate ADT instance (with equal elements).

Overrides tQueue.Duplicate.

Initialize *constructor tCircularQueue.Initialize(SizeOfElements: word);*

Initializes an instance of this ADT class.

Overrides tQueue.Initialize.

Overrides tCompositeADT.Initialize.

Overrides tADT.Initialize.

Overrides tObject.Initialize.

Resemble *constructor tCircularQueue.Resemble(ADT: pADT);*

Initializes an ADT that resembles the specified ADT.

Overrides tQueue.Resemble.

StreamLoad *constructor tCircularQueue.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

Overrides tQueue.StreamLoad.

Overrides tCompositeADT.StreamLoad.

Overrides tADT.StreamLoad.

Overrides tObject.StreamLoad.

tClassIdentity EFKERNEL

EFKERNEL.tClassIdentity = object(tContainer)

Class identity class: this class maintains information about a class in a class hierarchy. It knows the type name, the stream load and store methods (if available) and the relation to other classes.

Classes that are on the same level connect to each other using *tContainer* links. Each class can be a parent for a new class level (fDescendant).

Fields and Methods:

public
constructor *Initialize...*
destructor *Intercept*; virtual;
function *Search...*
procedure *StreamWrite...* virtual;
function *Identity...*

function *Name...*
function *Address...*
function *ParentAddress...*
function *Loader...*
function *Storer...*
function *Level...*
function *IsParent...*
function *IsDescendant...*
function *IsMatch...*
function *IsStreamable...*
private
fIdentity: word;
fName: pChar;
fVMT: pointer;
fLoader: pointer;
fStorer: pointer;
fParent: *pClassIdentity*;
fDescendant: *pClassIdentity*;

Fields

fDescendant *tClassIdentity.fDescendant: pClassIdentity;*

Pointer to the first descendants identity instance.

fIdentity *tClassIdentity.fIdentity: word;*

Specific identity number within [0, 65535].

fLoader *tClassIdentity.fLoader: pointer;*

Stream load constructor offset, [DS:xx] (optional).

fName *tClassIdentity.fName: pChar;*

Full class type name (eg. "tObject").

fParent *tClassIdentity.fParent: pClassIdentity;*

Pointer to the identity instance of the parent class.

fStorer *tClassIdentity.fStorer: pointer;*

Stream store method offset, [DS:xx] (optional).

fVMT *tClassIdentity.fVMT: pointer;*

VMT address for the referenced class.

Methods

Address *function tClassIdentity.Address: pointer;*

Returns the address to the class in this identity.

Identity *function tClassIdentity.Identity: word;*

Returns the identity of the carried data within [0, 65535].

Initialize *constructor tClassIdentity.Initialize(Link, nParent, nDescendant: pClassIdentity; nIdentity: word; nName: string; nAddress: pointer; nLoader, nStorer: pointer);*

Initializes an instance with the given properties.

Overrides tContainer.Initialize.

Overrides tObject.Initialize.

Intercept *destructor tClassIdentity.Intercept; virtual;*

Intercepts and destructs classes from this level and below.

Overrides tContainer.Intercept.

Overrides tObject.Intercept.

IsDescendant *function tClassIdentity.IsDescendant(What: pObject): boolean;*

Is the specified instance an immediate subclass?

Overrides tObject.IsDescendant.

IsMatch *function tClassIdentity.IsMatch(nIdentity: word; nName: string; nInstance, nParent: pObject): boolean;*

Is at least one of the specified properties correct?

IsParent *function tClassIdentity.IsParent(What: pObject): boolean;*

Is this class an immediate parent for the specified?

Overrides tObject.IsParent.

IsStreamable *function tClassIdentity.IsStreamable: boolean;*

Does this class support stream storage?

Level *function tClassIdentity.Level: word;*

Returns the level (depth) of this class identity.

Loader *function tClassIdentity.Loader: pointer;*

Returns the address to the stream load constructor.

Name *function tClassIdentity.Name: string;*

Returns the full type name of the class in this identity.

ParentAddress *function tClassIdentity.ParentAddress: pointer;*

Returns the address to the parent class.

Search *function tClassIdentity.Search(nIdentity: word; nName: string; nInstance, nParent: pObject): pClassIdentity;*

Find an identity that match at least one parameter.

Storer *function tClassIdentity.Storer: pointer;*

Returns the address to the stream store method.

StreamWrite *procedure tClassIdentity.StreamWrite(Stream: pStream); virtual;*

Writes the hierarchy as formatted text to a stream.

Overrides tObject.StreamWrite.

tClassManager EFKERNEL

EFKERNEL.tClassManager = object(tObject)

This is the class manager, ie. the class hiearchy database with EFLIB classes and optionally user-defined classes. The manager maintains the following information about all registered classes:

(1) Identity word [1 .. 65535] specific for all class.

(2) Full Pascal type name (eg. "tClassManager").

(3) Class address (VMT offset).

(4) Relations (parent and child classes).

(5) Stream store and load methods (optional but required for stream support).

The tClassManager is used for instance identification (eg. finding the type of an instance), event identification and compatibility checks.

Fields and Methods:

public
constructor *Initialize*;
destructor *Intercept*; virtual;

```
procedure Register... virtual;
procedure Load... virtual;
procedure Store... virtual;
function Search...
procedure StreamWrite... virtual;
function IsRegistered...
private
fHierarchy: pClassIdentity;
```

Fields

fHierarchy *tClassManager.fHierarchy: pClassIdentity;*

Class hierarchy (top-level identity instance).

Methods

Initialize *constructor tClassManager.Initialize;*

Initializes an empty class manager instance.

Overrides tObject.Initialize.

Intercept *destructor tClassManager.Intercept; virtual;*

Intercepts the entire class hierarchy.

Overrides tObject.Intercept.

IsRegistered *function tClassManager.IsRegistered(Instance: pObject): boolean;*

Is the instance (or VMT address) registered as a class?

Load *procedure tClassManager.Load(Identity: word; Stream: pStream; var Instance: pObject); virtual;*

Loads an instance from a stream, ie. constructs it.

Register *procedure tClassManager.Register(Identity: word; Name: string; Address, ParentAddress: pointer; Loader, Storer: pointer); virtual;*

Registers or replaces a class in the class hierarchy.

Search *function tClassManager.Search(Identity: word; Name: string; Instance, Parent: pObject): pClassIdentity;*

Find an identity that match at least one parameter.

Store *procedure tClassManager.Store(Stream: pStream; var Instance: pObject); virtual;*

Stores an instance to a stream.

StreamWrite *procedure tClassManager.StreamWrite(Stream: pStream); virtual;*

Writes the hierarchy as formatted text to a stream.

Overrides tObject.StreamWrite.

tClassRegister EFKERNEL

EFKERNEL.tClassRegister = object(tObject)

This is a class register - that is a register with some owned instances. tClassRegister is an abstract class, inherited into concrete classes such as *tCarrierRegister* and tRegister. A register support the following operatons:

(1) Registration of any *tObject* instance, provided that this instance is not blocked by *IsValid*. Upon registration, the register becomes responsible for the instance - even for its interception if the register is destructed before the instance is released from the register.

(2) An instance can be released from the register. The register is then no longer responsible for that instance.

(3) An arbitrary instance can be accessed using the identity value of that instance. The identity is a word-sized, positive integer value within [0, 65535]. The default identity is the location of the instance in the register.

(4) Identities are specific and unambiguous. There cannot be two instances with the same identity. The identity of an instance cannot be changed.

(5) A register can retrieve an arbitrary instance of a given type. The method *ReleaseClass* does this.

(6) Registers can be loaded or stored to streams. Then, all registered instances are also loaded or stored.

Fields and Methods:

public
procedure *Register...* virtual;
function *Release...* virtual;
function *ReleaseClass...* virtual;

function *First...* virtual;
function *Last...* virtual;
function *Instances...* virtual;
function *IsEmpty...* virtual;
function *IsRegistered...* virtual;
function *IsClassRegistered...* virtual;
function *IsValid...* virtual;

Descendants:

tCarrierRegister
tErrorHandler
EFREG.tRegister

Methods

First *function tClassRegister.First: word; virtual;*

Returns the lowest registered identity (zero if empty).

Instances *function tClassRegister.Instances: word; virtual;*

Returns the number of registered instances.

IsClassRegistered *function tClassRegister.IsClassRegistered(Class: pointer): boolean; virtual;*

Is an instance of this class or a subclass registered?

IsEmpty *function tClassRegister.IsEmpty: boolean; virtual;*

Is the register empty?

IsRegistered *function tClassRegister.IsRegistered(IdentityOfInstance: word): boolean; virtual;*

Is there an instance with the specified identity?

IsValid *function tClassRegister.IsValid(What: pObject): boolean; virtual;*

Can this instance (class) be registered - is it valid?

Last *function tClassRegister.Last: word; virtual;*

Returns the highest registered identity (zero if empty).

Register *procedure tClassRegister.Register(What: pObject); virtual;*

Registers an instance, that is, puts it into the register.

Release *function tClassRegister.Release(IdentityOfInstance: word): pObject; virtual;*

Releases the instance with this identity - or returns NIL.

ReleaseClass *function tClassRegister.ReleaseClass(Class: pointer): pObject; virtual;*

Releases a descendant from this class - or returns NIL.

tClassSelector EFSYSTEM

EFSYSTEM.tClassSelector = object(tContainer)

This class contains a reference to either an instance or a class. tClassSelector can connect to other instances of the same class, thus defining an entire set of classes, instances or both.

Fields and Methods:

private
fAddress: pointer;
fInstance: boolean;

Fields

fAddress *tClassSelector.fAddress: pointer;*

TypeOf address for a class or address to tObject instance.

fInstance *tClassSelector.fInstance: boolean;*

Is fAddress a tObject instance and not a class address?

tColor EFCONDRV

EFCONDRV.tColor = object(tNamedElement)

This class implements a color. Colors are assigned to text or graphical objects such as boxes or circles. A color has two member fields: the foreground color and the background color. Both are word-sized, positive integer values. Colors must be associated to a name (e.g. "Blue"). See tNamedElement, tColorMap and tColors.

Fields and Methods:

public
constructor *Initialize...*
constructor *StreamLoad...*
function *Foreground...* virtual;
function *Background...* virtual;
private
fForeground: word;
fBackground: word;

Fields

fBackground *tColor.fBackground: word;*

The background color.

fForeground *tColor.fForeground: word;*

The foreground color.

Methods

Background *function tColor.Background: word; virtual;*

Returns the background color value.

Foreground *function tColor.Foreground: word; virtual;*

Returns the foreground color value.

Initialize *constructor tColor.Initialize(NameOfColor: string; ColorForeground: word; ColorBackground: word);*

Initializes a static and named color instance.

StreamLoad *constructor tColor.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

tCommandLine EFCLINE

EFCLINE.tCommandLine = object(tParser)

This class implements a command-line argument parser. Arguments that was passed to the program on the command-line are easily read with the tCommandLine class.

Fields and Methods:

public
constructor *Initialize*;
function *IsOption...* virtual;

Methods

Initialize *constructor tCommandLine.Initialize;*

Initializes and constructs an instance of this type.

Overrides tText.Initialize.

Overrides tObject.Initialize.

IsOption *function tCommandLine.IsOption(What: string): boolean; virtual;*

Is the specified text an "/"-option passed to the program.

tCommunicator EFCMAN

EFCMAN.tCommunicator = object(tManager)

This class defines a special kind of manager, a communicator. A communicator does nothing itself, it is just a link between components and managers that controls the message flow.

The communicator has two responsibilies in addition to those of a manager:

(1) The communicator maintains a message queue with messages that are passed to components (and managers) that are owned by the communicator.

(2) The communicator enables filtering of messages, that is, an arbitrary component can be connected to the communicator. Only messages that are not classified as valid for that component (using IsValid), are passed through the communicator to the children.

Fields and Methods:

public
constructor *Initialize...*
destructor *Intercept*; virtual;
procedure *Post...* virtual;
function *GetMessage...* virtual;

function *PeekMessage...* virtual;
procedure *EnableDependence*;
procedure *DisableDependence*;
constructor *StreamLoad...*
procedure *StreamStore...* virtual;
private
fMessageQueue: *pMessage*;
fFilter: *pComponent*;
fDependentFilter: boolean;

Fields

fDependentFilter *tCommunicator.fDependentFilter: boolean;*

Is the filter owned by the communicator?

fFilter *tCommunicator.fFilter: pComponent;*

Optional message filter (component with #IsValid method).

fMessageQueue *tCommunicator.fMessageQueue: pMessage;*

Message queue with waiting messages and events.

Methods

DisableDependence *procedure tCommunicator.DisableDependence;*

Classifies the filter as an external component (default).

EnableDependence *procedure tCommunicator.EnableDependence;*

Makes the filter dependent of this class (owned).

GetMessage *function tCommunicator.GetMessage(var Message: pMessage): boolean; virtual;*

Gets a message from any message queue.

Overrides tComponent.GetMessage.

Initialize *constructor tCommunicator.Initialize(Filter: pComponent);*

Initializes this manager without any children.

Overrides tManager.Initialize.

Overrides tComponent.Initialize.

Overrides tMessage.Initialize.

Overrides tObject.Initialize.

Intercept *destructor tCommunicator.Intercept; virtual;*

Intercepts this manager and all its owned components.

Overrides tManager.Intercept.

Overrides tElement.Intercept.

Overrides tObject.Intercept.

PeekMessage *function tCommunicator.PeekMessage(var Message: pMessage): boolean; virtual;*

Peeks in the message queue without dispatching message.

Overrides tComponent.PeekMessage.

Post *procedure tCommunicator.Post(Message: pMessage); virtual;*

Posts the specified message via the manager.

Overrides tComponent.Post.

StreamLoad *constructor tCommunicator.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

Overrides tManager.StreamLoad.

Overrides tComponent.StreamLoad.

Overrides tStaticElement.StreamLoad.

Overrides tElement.StreamLoad.

Overrides tObject.StreamLoad.

StreamStore *procedure tCommunicator.StreamStore(Stream: pStream); virtual;*

Stores an instance to a stream.

Overrides tManager.StreamStore.

Overrides tComponent.StreamStore.

Overrides tStaticElement.StreamStore.

Overrides tElement.StreamStore.

Overrides tObject.StreamStore.

EFMATH.tComplex = object(tMathObject)

This class defines a complex number with a real and imaginary part and most of the standard arithmethics for these numbers.

Fields and Methods:

public
constructor *Initialize...*
constructor *Duplicate...*
procedure *SetValue...*
procedure *Add...* virtual;
procedure *Subtract...* virtual;
procedure *Multiply...* virtual;
procedure *Divide...* virtual;
procedure *Conjugate*;
procedure *Reciprocal*;
function *Argument...*
function *Magnitude...*
procedure *Power...*
procedure *Square...*
procedure *SquareRoot...*
procedure *Exp...*
procedure *Ln...*
constructor *StreamLoad...*
procedure *StreamStore...* virtual;
procedure *StreamWrite...* virtual;
function *Re...*
function *Im...*
function *IsZero...* virtual;
private
fRe,
fIm: *tBaseReal*;

Fields

fIm *See fRe*

fRe *tComplex.fRe,*

fIm: tBaseReal;

Real and imaginary part of number.

Methods

Add *procedure tComplex.Add(What: pMathObject); virtual;*

Adds an object to this object.

Overrides tMathObject.Add.

Argument *function tComplex.Argument: tBaseReal;*

Returns the argument within [-Pi/2, Pi/2].

Conjugate *procedure tComplex.Conjugate;*

Conjugates number (a+ib => a-ib).

Divide *procedure tComplex.Divide(What: pMathObject); virtual;*

Divides this object by another object.

Overrides tMathObject.Divide.

Duplicate *constructor tComplex.Duplicate(What: pMathObject);*

Initializes an duplicated type-casted mathematical object. Setup methods

Exp *procedure tComplex.Exp(What: pComplex);*

Makes this number the exponential of another number.

Im *function tComplex.Im: tBaseReal;*

Returns the imaginary part of the number.

Initialize *constructor tComplex.Initialize(nRe, nIm: tBaseReal);*

Initializes an instance with the specified content.

Overrides tObject.Initialize.

IsZero *function tComplex.IsZero: boolean; virtual;*

Is number equal to zero?

Overrides tMathObject.IsZero.

Ln *procedure tComplex.Ln(What: pComplex);*

Makes this number the natural logarithm of another number.

Magnitude *function tComplex.Magnitude: tBaseReal;*

Returns the magnitude of the number, ie. the length.

Multiply *procedure tComplex.Multiply(What: pMathObject); virtual;*

Multiplies this object with another object.

Overrides tMathObject.Multiply.

Power *procedure tComplex.Power(Exponent: pComplex);*

Makes this number the power of itself and the specified.

Re *function tComplex.Re: tBaseReal;*

Returns the real part of the number.

Reciprocal *procedure tComplex.Reciprocal;*

Takes the reciprocal of the complex number (z=1/z).

SetValue *procedure tComplex.SetValue(nRe, nIm: tBaseReal);*

Sets the number.

Square *procedure tComplex.Square(What: pComplex);*

Makes this number the square of another number.

SquareRoot *procedure tComplex.SquareRoot(What: pComplex);*

Makes this number the square root of another number.

StreamLoad *constructor tComplex.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

Overrides tStaticElement.StreamLoad.

Overrides tElement.StreamLoad.

Overrides tObject.StreamLoad.

StreamStore *procedure tComplex.StreamStore(Stream: pStream); virtual;*

Stores an instance to a stream.

Overrides tStaticElement.StreamStore.

Overrides tElement.StreamStore.

Overrides tObject.StreamStore.

StreamWrite *procedure tComplex.StreamWrite(Stream: pStream); virtual;*

Writes this math object as formatted text, e.g. "2+2i".

Overrides tElement.StreamWrite.

Overrides tObject.StreamWrite.

Subtract *procedure tComplex.Subtract(What: pMathObject); virtual;*

Subtracts an object from this object.

Overrides tMathObject.Subtract.

tComponent EFCMAN

EFCMAN.tComponent = object(tMessage)

This is the abstract component class that defines what components must be able to do. A component is an exention of the *tObject* class. In EFLIB, *tComponent* is the core of the event-driven architecture known as component- manager architecture.

Components receive and send messages (events) to other components. This is very different from a common programming model. A component is merely permitted to reply to some messages and to do the associated actions. After these actions have been performed, the component must enable other components to handle their messages. All components are handled via the application engine with the message loop (tApplication). tApplication is defined in *EFAPP*.

Properties of components:

(1) Components can be focused, that is activated or selected. Components are collected in Z-order. The first component is hence the focused. See *tManager*.

(3) If a component is destructed, all its child components are also destructed - if not this process is explicitly overridden in a subclass.

(4) Components can connect to other components since they descend from the *tContainer* class.

(5) Components are locked if they cannot process messages (other than service messages) and cannot update their status, e.g. redraw themselves (see *IsLocked*). Components are locked when other components are updated (drawn) or when the user request a termination query and the component is waiting for the termination to be confirmed.

(6) Components are protected if they cannot be destructed and does not permit status requests. Managers are normally protected until all child components have been removed (see IsProteced).

(7) Components can be visual, ie. assigned to an external view (see *IsVisual*).

(8) Components are capable of handling messages, that is, descendants of the *tMessage* class. A message is an external action or a simply a notification of an event. The component must not respond to all messages, but is responsible for their destruction. Unknown messages should be passed to inherited methods.

(9) A message that cannot be handled by this component or any children component can be blocked by the *IsValid* method.

(10) Components can be responsible for other components. However, *tComponent* is only responsible for itself. *tManager* overrides the *IsResponsible* method since it also is responsible for the child components. This responsiblity means that a message can be handled by the manager even though the receiver is a children and not the manager itself. *IsResponsible* guides messages through the network of components and manager to the right receiver - it defines the message path.

When a message is send to a component, it is passed as an argument to the *Receive* method. You must always invoke the inherited *Receive* method if your class does not recognize the message. The inherited method takes care of standard messages, destructs any unrecognized messages and. Standard messages are those that lockes or protects the component, intercepts the component or request status information about the component.

Component can themselves contain more components and are then called managers (see *tManager*).

Fields and Methods:

public
constructor *Initialize*;
procedure *Waiting*; virtual;
procedure *Update*; virtual;
procedure *Finish*; virtual;
procedure *Continue*; virtual;

procedure *Receive...* virtual;
procedure *Handle...* virtual;
procedure *Post...* virtual;
procedure *Send...* virtual;
function *GetMessage...* virtual;
function *PeekMessage...* virtual;
constructor *StreamLoad...*
procedure *StreamStore...* virtual;
function *IsFocused...* virtual;
function *IsManaged...* virtual;
function *IsLocked...* virtual;
function *IsProtected...* virtual;
function *IsSelectable...* virtual;
function *IsExclusive...* virtual;
function *IsVisual...* virtual;
function *IsFinished...* virtual;
function *IsServiceMessage...* virtual;
function *IsResponsible...* virtual;
function *IsValid...* virtual;
private
fManager: *pManager*;
fLocked: boolean;
fProtected: boolean;
fSelectable: boolean;
fExclusive: boolean;
fFinished: boolean;

Descendants:

tManager
  EFAPP.tApplication
  tCommunicator
EFCONDRV.tConsole
  EFCONDRV.tBoundedConsole
    EFCONDRV.tCRT
      EFSCREEN.tScreen
    EFCONDRV.tVirtualConsole
      EFCONDRV.tImage
      EFSCREEN.tBufferedScreen
EFDEVICE.tDevice
  EFDEVICE.tSoundDevice
  EFMOUSE.tMouseDevice
EFVIEWS.tView
  EFVIEWS.tGroup

Fields

fExclusive *tComponent.fExclusive: boolean;*

Does this component lock other components when focused?

fFinished *tComponent.fFinished: boolean;*

Is this component waiting for termination?

fLocked *tComponent.fLocked: boolean;*

Is this component locked (ie. cannot receive events)?

fManager *tComponent.fManager: pManager;*

Manager of this component (optional).

fProtected *tComponent.fProtected: boolean;*

Is this component protected (i.e. cannot be destructed)?

fSelectable *tComponent.fSelectable: boolean;*

Is this component selectable - can it be focused?

Methods

Continue *procedure tComponent.Continue; virtual;*

Abort a termination request by unsetting the finish flag.

Finish *procedure tComponent.Finish; virtual;*

Send a termination request message to the manager.

GetMessage *function tComponent.GetMessage(var Message: pMessage): boolean; virtual;*

Gets a message from any message queue.

Handle *procedure tComponent.Handle(Message: pMessage); virtual;*

Handles a single (non-chained) message.

Initialize *constructor tComponent.Initialize;*

Initializes and constructs a component with a conenction.

Overrides tMessage.Initialize.

Overrides tObject.Initialize.

IsExclusive *function tComponent.IsExclusive: boolean; virtual;*

Returns TRUE if this component require exclusive execution.

IsFinished *function tComponent.IsFinished: boolean; virtual;*

Is this component waiting for termination?

IsFocused *function tComponent.IsFocused: boolean; virtual;*

Is this a foreground component, i.e. activated.

IsLocked *function tComponent.IsLocked: boolean; virtual;*

Is this component locked - cannot receive messages?

IsManaged *function tComponent.IsManaged: boolean; virtual;*

Is this component a part of a component-manager network?

IsProtected *function tComponent.IsProtected: boolean; virtual;*

Is the specified component protected (not controllable)?

IsResponsible *function tComponent.IsResponsible(Component: pComponent; var Result: pComponent): boolean; virtual;*

Is this component responsible for the specified component?

IsSelectable *function tComponent.IsSelectable: boolean; virtual;*

Is this component selectable - can it be focused?

IsServiceMessage *function tComponent.IsServiceMessage(Message: pMessage): boolean; virtual;*

Is the specified message a service message?

IsValid *function tComponent.IsValid(Message: pMessage): boolean; virtual;*

Returns TRUE if the specified message is valid.

IsVisual *function tComponent.IsVisual: boolean; virtual;*

Is this component visual, that is, uses a console device?

PeekMessage *function tComponent.PeekMessage(var Message: pMessage): boolean; virtual;*

Peeks in the message queue without dispatching message.

Post *procedure tComponent.Post(Message: pMessage); virtual;*

Posts the specified message via the manager.

Receive *procedure tComponent.Receive(Message: pMessage); virtual;*

Attempts to receive the specified message(s).

Send *procedure tComponent.Send(Message: pMessage); virtual;*

Sends the specified message directly to the receiver.

StreamLoad *constructor tComponent.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

Overrides tStaticElement.StreamLoad.

Overrides tElement.StreamLoad.

Overrides tObject.StreamLoad.

StreamStore *procedure tComponent.StreamStore(Stream: pStream); virtual;*

Stores an instance to a stream.

Overrides tStaticElement.StreamStore.

Overrides tElement.StreamStore.

Overrides tObject.StreamStore.

Update *procedure tComponent.Update; virtual;*

Updates the component after its status has changed.

Waiting *procedure tComponent.Waiting; virtual;*

Permits this component to do something when it is waiting.

tComponentSelector EFSYSTEM

EFSYSTEM.tComponentSelector = object(tSetting)

This is a run-time setting for component selectors. A component selector selects what instance (derived from a known parent class) a certain class require in a certain situation. For example, ADT′s may require a plug for sorting. The component selector then selects the sorting plug. Thus, if you change this setting you can change the default behavior of your ADT′s - and of other classes.

Fields and Methods:

public
private

tCompositeADT EFDATA

EFDATA.tCompositeADT = object(tOrderedADT)

This class defines an abstract composite ADT. A composite ADT is an ordered data structure that depends on (and owns an instance of) some linear ADT.

In other words: tCompositeADT is an abstract class from which ordered ADT′s that depend on a linear structure are derived. See for example *tStack* and *tQueue*.

The component selector methos *CreateComposite* returns a tList instance, but it can be overridden so it returns an linear ADT of arbitrary type. When a composite class is loaded from a stream, this component selector is first called to construct the owned linear ADT.

Fields and Methods:

public
fComposite: *pLinearADT*;
constructor *Initialize...*
destructor *Intercept*; virtual;
procedure *Clear*; virtual;
procedure *Erase...* virtual;
function *Compare...* virtual;
procedure *Put...* virtual;
function *Get...* virtual;
function *CreateIterator...* virtual;
function *CreateComposite...* virtual;
constructor *StreamLoad...*
procedure *StreamStore...* virtual;
function *Elements...* virtual;
function *Capacity...* virtual;
function *PackageSize...* virtual;
function *IsAllocated...* virtual;
function *IsReadOnly...* virtual;
function *IsEmpty...* virtual;
function *IsHomogeneous...* virtual;

Descendants:

tQueue
  tCircularQueue
  tPriorityQueue
tStack
EFTABLE.tHashTable
  EFTABLE.tBucketHashTable
  EFTABLE.tOpenHashTable
    EFTABLE.tDoubleHashTable
    EFTABLE.tProbeHashTable

Fields

fComposite *tCompositeADT.fComposite: pLinearADT;*

Pointer to the linear composite ADT (owned).

Methods

Capacity *function tCompositeADT.Capacity: word; virtual;*

Returns the storage capacity in number of elements.

Overrides tADT.Capacity.

Clear *procedure tCompositeADT.Clear; virtual;*

Clears or removes all elements inside data type.

Overrides tADT.Clear.

Compare *function tCompositeADT.Compare(A, B: pElement): shortint; virtual;*

Compares elements. Uses fKeyPlug if it′s assigned.

Overrides tADT.Compare.

CreateComposite *function tCompositeADT.CreateComposite(NumberOfElements: word): pLinearADT; virtual;*

Creates an instance of the composite ADT.

CreateIterator *function tCompositeADT.CreateIterator: pIterator; virtual;*

Creates an instance of the iterator class.

Overrides tADT.CreateIterator.

Elements *function tCompositeADT.Elements: word; virtual;*

Returns the number of elements that currently are used.

Overrides tADT.Elements.

Erase *procedure tCompositeADT.Erase(var Key; KeySize: word); virtual;*

Erases element(s) with the specified search key.

Overrides tOrderedADT.Erase.

Get *function tCompositeADT.Get(var Key; KeySize: word): pElement; virtual;*

Gets the specified element instance.

Overrides tOrderedADT.Get.

Initialize *constructor tCompositeADT.Initialize(SizeOfElements: word; Composite: pLinearADT);*

Initializes an instance of this ADT class.

Overrides tADT.Initialize.

Overrides tObject.Initialize.

Intercept *destructor tCompositeADT.Intercept; virtual;*

Intercepts and destructs an instance of this ADT class.

Overrides tADT.Intercept.

Overrides tObject.Intercept.

IsAllocated *function tCompositeADT.IsAllocated: boolean; virtual;*

Is this ADT allocated, that is ready for use?

Overrides tADT.IsAllocated.

IsEmpty *function tCompositeADT.IsEmpty: boolean; virtual;*

Is this ADT type empty, that is does not have any elements?

Overrides tADT.IsEmpty.

IsHomogeneous *function tCompositeADT.IsHomogeneous: boolean; virtual;*

Is this ADT homogeneous, that is has compatible elements?

Overrides tADT.IsHomogeneous.

IsReadOnly *function tCompositeADT.IsReadOnly: boolean; virtual;*

Is this ADT restricted to read access?

Overrides tADT.IsReadOnly.

PackageSize *function tCompositeADT.PackageSize: word; virtual;*

Returns the size of element packaging (container).

Overrides tADT.PackageSize.

Put *procedure tCompositeADT.Put(Element: pElement); virtual;*

Stores a new element (tElement instance) to this ADT.

Overrides tADT.Put.

StreamLoad *constructor tCompositeADT.StreamLoad(Stream: pStream);*

Loads an ADT from a stream.

Overrides tADT.StreamLoad.

Overrides tObject.StreamLoad.

StreamStore *procedure tCompositeADT.StreamStore(Stream: pStream); virtual;*

Stores an ADT to a stream.

Overrides tADT.StreamStore.

Overrides tObject.StreamStore.

tCompressFilter EFFILTER

EFFILTER.tCompressFilter = object(tSequentialFilter)

This is a data compression filter that uses a splay compression algorithm by Jeffrey Chilton and Douglas Jones (C). The filter compresses as it writes and decompresses as it reads. Normal data can be compressed to at least 50-85% of its size.

Compression is done whenever *Read* or *Write* is called. If you transfer data in blocks when writing, always make sure that you read it with the same block size.

This filter requires about 1 KB of static memory, and also allocates an internal buffer each time read or write is called.

Fields and Methods:

public
constructor *Initialize...*

procedure *Reset*; virtual;
procedure *Read*... virtual;
procedure *Write*... virtual;
private
fUp: array...
fLeft,
fRight: array...
procedure *Splay*...

Fields

fLeft *tCompressFilter.fLeft,*

fRight: array[0..255] of word;

Splay tree left and right child nodes.

fRight *See fLeft*

fUp *tCompressFilter.fUp: array[0..512] of byte;*

Splay tree nodes.

Methods

Initialize *constructor tCompressFilter.Initialize(Stream: pStream);*

Initializes an instance as dependent on a base stream.

Overrides tFilter.Initialize.

Overrides tStream.Initialize.

Overrides tObject.Initialize.

Read *procedure tCompressFilter.Read(var Data; Count: word); virtual;*

Reads specified amount of data from this stream.

Overrides tFilter.Read.

Overrides tStream.Read.

Reset *procedure tCompressFilter.Reset; virtual;*

Resets stream and moves to the first position.

Overrides tStream.Reset.

Splay *procedure tCompressFilter.Splay(What: word);*

Put the specified byte in the splay tree.

Write *procedure tCompressFilter.Write(var Data; Count: word); virtual;*

Writes specified data to this stream.

Overrides tFilter.Write.

Overrides tStream.Write.

tConsole EFCONDRV

EFCONDRV.tConsole = object(tComponent)

This class defines an abstract console. A console is a screen or buffer that acts like a screen (an image). Consoles provide all the low-level output mechanisms that are associated to screens. A console must:

(1) Know the current coordinate, font and color (instances of tCoordinate, *tFont* and *tColor*).

(2) Permit text output using a specific font and color - at arbitrary coordinates within the console.

(5) Know when it may and may not be updated. A console may only be updated when it is not locked (see *tComponent*).

Fields and Methods:

public
constructor *Initialize*;
procedure *SetColor...* virtual;
procedure *SetFont...* virtual;
procedure *MoveIn...* virtual;
procedure *MoveOut...* virtual;
procedure *FetchStatus...* virtual;
procedure *UpdateStatus...* virtual;
procedure *Clear*; virtual;
procedure *GotoXY...* virtual;
procedure *Move...* virtual;
procedure *Write...* virtual;
procedure *WriteLn...* virtual;
procedure *WriteXY...* virtual;
procedure *WriteCustom...* virtual;
procedure *LineFeed*; virtual;
procedure *Scroll...* virtual;
procedure *ScrollField...* virtual;

procedure *DrawLine...* virtual;
procedure *Fill...* virtual;
procedure *FillField...* virtual;
function *CreateBoundaries...* virtual;
function *DefaultColor...* virtual;
function *DefaultFont...* virtual;
function *DefaultLine...* virtual;
function *DefaultFill...* virtual;
constructor *StreamLoad...*
procedure *StreamStore...* virtual;
function *CurrentColor...* virtual;
function *CurrentFont...* virtual;
function *X...* virtual;
function *Y...* virtual;
function *Width...* virtual;
function *Height...* virtual;
function *Address...* virtual;
function *IsVisible...* virtual;
function *IsFieldVisible...* virtual;
function *IsEqual...* virtual;
function *IsCompatible...* virtual;
private
fColor: *pColor*;
fFont: *pFont*;
fResolution: byte;

Descendants:

tBoundedConsole
  tCRT
    EFSCREEN.tScreen
  tVirtualConsole
    tImage
    EFSCREEN.tBufferedScreen

Fields

fColor *tConsole.fColor: pColor;*

Current color instance.

fFont *tConsole.fFont: pFont;*

Current font instance.

fResolution *tConsole.fResolution: byte;*

Number of bytes per pixel entitity.

Methods

Address *function tConsole.Address(X1, Y1: word): pointer; virtual;*

Returns a pointer to the pixel at (X,Y), or NIL.

Clear *procedure tConsole.Clear; virtual;*

Clears the entire console.

CreateBoundaries *function tConsole.CreateBoundaries: pField; virtual;*

Creates a tField instance with the boundaries.

CurrentColor *function tConsole.CurrentColor: pColor; virtual;*

Returns the current color (a pointer to some map element).

CurrentFont *function tConsole.CurrentFont: pFont; virtual;*

Returns the current font (a pointer to some font element).

DefaultColor *function tConsole.DefaultColor: pColor; virtual;*

Returns a pointer to the default console color.

DefaultFill *function tConsole.DefaultFill: pFillStyle; virtual;*

Returns a pointer to the default fill style.

DefaultFont *function tConsole.DefaultFont: pFont; virtual;*

Returns a pointer to the default console font.

DefaultLine *function tConsole.DefaultLine: pLineStyle; virtual;*

Returns a pointer to the default line style.

DrawLine *procedure tConsole.DrawLine(X1, Y1, X2, Y2: word; Color: pColor; Style: pLineStyle); virtual;*

Draws a line from (X1,Y1) to (X2,Y2) using some settings.

FetchStatus *procedure tConsole.FetchStatus(Console: pConsole); virtual;*

Fetchs status (properties) from specified console.

Fill *procedure tConsole.Fill(Color: pColor; Style: pFillStyle); virtual;*

Fills the entire console with some pattern.

FillField *procedure tConsole.FillField(X1, Y1, X2, Y2: word; Color: pColor; Style: pFillStyle); virtual;*

Fills a region of the console with some pattern.

GotoXY *procedure tConsole.GotoXY(X1, Y1: word); virtual;*

Moves to the specified coordinates.

Height *function tConsole.Height: word; virtual;*

Returns the height of the console in coordinate resolution.

Initialize *constructor tConsole.Initialize;*

Initializes a console with the default properties.

Overrides tComponent.Initialize.

Overrides tMessage.Initialize.

Overrides tObject.Initialize.

IsCompatible *function tConsole.IsCompatible(Instance: pObject): boolean; virtual;*

Are classes type-compatible (fully replacable).

Overrides tStaticElement.IsCompatible.

Overrides tElement.IsCompatible.

Overrides tObject.IsCompatible.

IsEqual *function tConsole.IsEqual(Instance: pObject): boolean; virtual;*

Are contents of these instances equal (bitwise compare)?

Overrides tElement.IsEqual.

Overrides tObject.IsEqual.

IsFieldVisible *function tConsole.IsFieldVisible(Field: pField): boolean; virtual;*

Is the specified field inside the console and visible?

IsVisible *function tConsole.IsVisible: boolean; virtual;*

Is at least some of this console visible?

Linefeed *procedure tConsole.LineFeed; virtual;*

Moves to next virtual text row and first column.

Move *procedure tConsole.Move(StepX, StepY: integer); virtual;*

Moves some steps relative to the current position.

MoveIn *procedure tConsole.MoveIn(Where: pCoordinates; Source: pConsole; Field: pField); virtual;*

Moves the specfied field from another console driver.

MoveOut *procedure tConsole.MoveOut(Field: pField; Target: pConsole; Where: pCoordinates); virtual;*

Moves the specified field into another console driver.

Scroll *procedure tConsole.Scroll(Direction: tDirection; Steps: word; Color: pColor; Style: pFillStyle); virtual;*

Scrolls the content of the console in a direction.

ScrollField *procedure tConsole.ScrollField(Field: pField; Direction: tDirection; Steps: word; Color: pColor; Style: pFillStyle); virtual;*

Scrolls a field in the console in a direction.

SetColor *procedure tConsole.SetColor(Color: pColor); virtual;*

Sets the current color (for outputs).

SetFont *procedure tConsole.SetFont(Font: pFont); virtual;*

Sets the current font (fot text output).

StreamLoad *constructor tConsole.StreamLoad(Stream: pStream);*

Constructs and loads an instance from a stream.

Overrides tComponent.StreamLoad.

Overrides tStaticElement.StreamLoad.

Overrides tElement.StreamLoad.

Overrides tObject.StreamLoad.

StreamStore *procedure tConsole.StreamStore(Stream: pStream); virtual;*

Stores an instance to a stream.

Overrides tComponent.StreamStore.

Overrides tStaticElement.StreamStore.

Overrides tElement.StreamStore.

Overrides tObject.StreamStore.

UpdateStatus *procedure tConsole.UpdateStatus(Console: pConsole); virtual;*

Updates status (properties) in the specified console.

Width *function tConsole.Width: word; virtual;*

Returns the width of the console in coordinate resolution.

Write *procedure tConsole.Write(Text: pString); virtual;*

Writes the specified text using current settings.

WriteCustom *procedure tConsole.WriteCustom(X1, Y1: word; Color: pColor; Font: pFont; Text: pString); virtual;*

Writes the specified text with some settings (NIL=default).

WriteLn *procedure tConsole.WriteLn(Text: pString); virtual;*

Writes the specified text and CR/LF using current settings.

WriteXY *procedure tConsole.WriteXY(X1, Y1: word; Text: pString); virtual;*

Writes the specified text at some coordinate.

X *function tConsole.X: word; virtual;*

Returns the current X coordinate.

Y *function tConsole.Y: word; virtual;*

Returns the current Y coordinate.

tConsoleSelector EFCONDRV

EFCONDRV.tConsoleSelector = object(tObject)

This is an abstract component selector class for console devices. This class provides methods that create or return an instance to a default component when the console driver require one. You can override methods in this class to change the default behavior of consoles. See GUISelector.

All component selectors must return valid components after they have been called. If this is not possible, the component selector trigger a fatal error that terminates program execution.

Fields and Methods:

public
constructor *Initialize*;
function *CreateBoundaries*... virtual;
function *DefaultColor*... virtual;
function *DefaultFont*... virtual;
function *DefaultLine*... virtual;
function *DefaultFill*... virtual;

Methods

CreateBoundaries *function tConsoleSelector.CreateBoundaries(Console: pConsole): pField; virtual;*

Creates an instance of the boundary class (tField).

DefaultColor *function tConsoleSelector.DefaultColor: pColor; virtual;*

Returns a pointer to the default console color.

DefaultFill *function tConsoleSelector.DefaultFill: pFillStyle; virtual;*

Returns a pointer to the default fill style.

DefaultFont *function tConsoleSelector.DefaultFont: pFont; virtual;*

Returns a pointer to the default console font.

DefaultLine *function tConsoleSelector.DefaultLine: pLineStyle; virtual;*

Returns a pointer to the default line style.

Initialize *constructor tConsoleSelector.Initialize;*

Initializes an instance of this class.

Overrides tObject.Initialize.

tContainer EFKERNEL

EFKERNEL.tContainer = object(tObject)

This class implements an abstract container, that is a class that can link to other instances. Containers form structures or chains, where arbitrary many ordered instances can be attached to each other.

When a container is intercepted, it is detached from any structure. The order of the remaining containers is therefore preserved.

This class has two member fields: the successor and the predecessor container. Both can be addresses to other containers.

Fields and Methods:

public
constructor *Initialize...*
destructor *Intercept*; virtual;
destructor *FreeAll*; virtual;
procedure *SetSuccessor...* virtual;
procedure *SetPredecessor...* virtual;
procedure *AttachAfter...* virtual;
procedure *AttachBefore...* virtual;
procedure *Detach*; virtual;
function *Successor...* virtual;
function *Predecessor...* virtual;
function *First...* virtual;
function *Last...* virtual;
function *Width...* virtual;
function *Relative...* virtual;
function *HasSuccessor...* virtual;
function *HasPredecessor...* virtual;
function *IsAttached...* virtual;
function *IsIntact...* virtual;
private
fSuccessor: *pContainer*;
fPredecessor: *pContainer*;

Descendants:

EFBASIC.tCoordinates
EFBASIC.tField
tCarrier
  EFLIST.tLinkage
    EFTREE.tTreeLinkage
      EFTREE.tAVLLinkage
tClassIdentity
EFSYSTEM.tClassSelector

Fields

fPredecessor *tContainer.fPredecessor: pContainer;*

Pointer to predecessor container.

fSuccessor *tContainer.fSuccessor: pContainer;*

Pointer to successor container (or NIL if none).

Methods

AttachAfter *procedure tContainer.AttachAfter(Container: pContainer); virtual;*

Attachs container(s) after this container (by insertion).

AttachBefore *procedure tContainer.AttachBefore(Container: pContainer); virtual;*

Attachs container(s) before this container (by insertion).

Detach *procedure tContainer.Detach; virtual;*

Detachs this container (preserves any structure).

First *function tContainer.First: pContainer; virtual;*

Returns a pointer to the first container in this structure.

FreeAll *destructor tContainer.FreeAll; virtual;*

Intercepts and releases memory for this and attached.

HasPredecessor *function tContainer.HasPredecessor: boolean; virtual;*

Has this container at least one predecessor?

HasSuccessor *function tContainer.HasSuccessor: boolean; virtual;*

Has this container at least one successor?

Initialize *constructor tContainer.Initialize(Where: pContainer);*

Initializes a linked container and attachs it after Where.

Overrides tObject.Initialize.

Intercept *destructor tContainer.Intercept; virtual;*

Intercepts and detachs a single container.

Overrides tObject.Intercept.

IsAttached *function tContainer.IsAttached: boolean; virtual;*

Is this container attached, ie. linked to other containers?

IsIntact *function tContainer.IsIntact: boolean; virtual;*

Cerfifies that this container has valid links (or none).

Last *function tContainer.Last: pContainer; virtual;*

Returns a pointer to the last container in this structure.

Predecessor *function tContainer.Predecessor: pContainer; virtual;*

Returns a pointer to the predecessor container, or NIL.

Relative *function tContainer.Relative(Steps: integer): pContainer; virtual;*

Returns a pointer to a relative node some steps away.

SetPredecessor *procedure tContainer.SetPredecessor(Link: pContainer); virtual;*

Sets the predecessor container link.

SetSuccessor *procedure tContainer.SetSuccessor(Link: pContainer); virtual;*

Sets the successor container link.

Successor *function tContainer.Successor: pContainer; virtual;*

Returns a pointer to the successor container, or NIL.

Width *function tContainer.Width: word; virtual;*

Returns the number of containers in this chain.

tConverter EFCONV

EFCONV.tConverter = object(tFilter)

This is an extension of the *tFilter* class that automatically converts certain text expression while the filter is reading or writing data. These expressions are stored in a hash table.

Fields and Methods:

public
function *GetToken*...
private
fReplaceTable: *pHashTable*;

Fields

fReplaceTable *tConverter.fReplaceTable: pHashTable;*

Table with expressions that shall be replaced.

Methods

GetToken *function tConverter.GetToken(Delimeters: pString): string;*

tCoordinates EFBASIC

EFBASIC.tCoordinates = object(tContainer)

This class defines a coordinate pair, ie. two word-sized, positive integer numbers representing a position, e.g. on the screen or in a view. tCoordinates descend from tContainer and can hence be connected to other container classes.

Fields and Methods:

public
fX,
fY: word;
constructor *Initialize...*
constructor *Duplicate...*
procedure *Assign...*
procedure *SetCoordinates...*
procedure *Swap...*
procedure *Move...*
procedure *Adjust...*
constructor *StreamLoad...*
procedure *StreamStore...* virtual;
procedure *StreamWrite...* virtual;
function *IsEqual...* virtual;
function *X...*
function *Y...*
function *IsInsideBoundaries...* virtual;

Fields

fX *tCoordinates.fX,*

fY: word;

Carthesian coordinate pair: (X, Y).

fY *See fX*

Methods

Adjust *procedure tCoordinates.Adjust(Field: pField);*

Adjusts coordinate to be inside the specified field.

Assign *procedure tCoordinates.Assign(Coordinates: pCoordinates);*

Assigns the specified coordinates to this instance.

Duplicate *constructor tCoordinates.Duplicate(Coordinates: pCoordinates);*

Initializes and duplicates an instance of this class.

Initialize *constructor tCoordinates.Initialize(nX, nY: word);*

Initializes and constructs an instance of this class.

Overrides tContainer.Initialize.

Overrides tObject.Initialize.

IsEqual *function tCoordinates.IsEqual(Instance: pObject): boolean; virtual;*

Are contents of these instances equal (bitwise compare)?

Overrides tObject.IsEqual.

IsInsideBoundaries *function tCoordinates.IsInsideBoundaries(X1, Y1: word; X2, Y2: word): boolean; virtual;*

Are coordinates inside the specified boundaries?

Move *procedure tCoordinates.Move(DeltaX, DeltaY: integer);*

Moves the coordinates a relative number of steps.

SetCoordinates *procedure tCoordinates.SetCoordinates(nX, nY: word);*

Sets the coordinates.

StreamLoad *constructor tCoordinates.StreamLoad(Stream: pStream);*

Constructs and loads an instance from a stream.

Overrides tObject.StreamLoad.

StreamStore *procedure tCoordinates.StreamStore(Stream: pStream); virtual;*

Stores an instance to a stream.

Overrides tObject.StreamStore.

StreamWrite *procedure tCoordinates.StreamWrite(Stream: pStream); virtual;*

Writes a coordinate text to the stream, ie. "(X, Y)".

Overrides tObject.StreamWrite.

Swap *procedure tCoordinates.Swap(Coordinates: pCoordinates);*

Swaps coordinates with another instance.

X *function tCoordinates.X: word;*

Returns the X coordinate.

Y *function tCoordinates.Y: word;*

Returns the Y coordinate.

tCRC16Filter EFFILTER

EFFILTER.tCRC16Filter = object(tSequentialFilter)

This is a sequential filter that calculates a 16-bit cyclic redundancy check (XMODEM CRC).

Fields and Methods:

public
constructor *Initialize...*
procedure *Reset*; virtual;
procedure *Read...* virtual;
procedure *Write...* virtual;
function *Result...* virtual;
private
fCRC: longint;
procedure *Update...* virtual;

Descendants:

tCRC32Filter

Fields

fCRC *tCRC16Filter.fCRC: longint;*

Methods

Initialize *constructor tCRC16Filter.Initialize(Stream: pStream);*

Initializes an instance as dependent on a base stream.

Overrides tFilter.Initialize.

Overrides tStream.Initialize.

Overrides tObject.Initialize.

Read *procedure tCRC16Filter.Read(var Data; Count: word); virtual;*

Reads specified amount of data from this stream.

Overrides tFilter.Read.

Overrides tStream.Read.

Reset *procedure tCRC16Filter.Reset; virtual;*

Resets stream and moves to the first position.

Overrides tStream.Reset.

Result *function tCRC16Filter.Result: longint; virtual;*

Returns the current CRC value.

Update *procedure tCRC16Filter.Update(var Data; Count: word); virtual;*

Updates the CRC.

Write *procedure tCRC16Filter.Write(var Data; Count: word); virtual;*

Writes specified data to this stream.

Overrides tFilter.Write.

Overrides tStream.Write.

tCRC32Filter EFFILTER

EFFILTER.tCRC32Filter = object(tCRC16Filter)

This class defines a filter that calculates 32-bit CRC (cyclic redundancy check) of ZMODEM style. If CRCTABLES is defined in FLAGS.INC, a static table are used (fast), otherwise the required table is generated when the filter is constructed (slower).

Fields and Methods:

public
constructor *Initialize...*
procedure *Reset*; virtual;
function *Result...* virtual;
private
procedure *Update...* virtual;

Methods

Initialize *constructor tCRC32Filter.Initialize(Stream: pStream);*

Initializes an instance as dependent on a base stream.

Overrides tCRC16Filter.Initialize.

Overrides tFilter.Initialize.

Overrides tStream.Initialize.

Overrides tObject.Initialize.

Reset *procedure tCRC32Filter.Reset; virtual;*

Resets stream and moves to the first position.

Overrides tCRC16Filter.Reset.

Overrides tStream.Reset.

Result *function tCRC32Filter.Result: longint; virtual;*

Returns the current CRC value.

Overrides tCRC16Filter.Result.

Update *procedure tCRC32Filter.Update(var Data; Count: word); virtual;*

Updates the CRC for the specified data.

Overrides tCRC16Filter.Update.

tCRT EFCONDRV

EFCONDRV.tCRT = object(tBoundedConsole)

This class defines a CRT console, that is a console that communicates directly with the standard CRT unit in Pascal. CRT is handles the current cursor position. However, this class uses direct screen writes instead of using the Write and WriteLn functions.

Fields and Methods:

public
constructor *Initialize*;
procedure *DetectMode*; virtual;
procedure *GotoXY*... virtual;
constructor *StreamLoad*...
function *X*... virtual;
function *Y*... virtual;
function *Address*... virtual;
private
fAddress: pointer;

Descendants:

EFSCREEN.tScreen

Fields

fAddress *tCRT.fAddress: pointer;*

Video memory address ($B800 or $B000).

Methods

Address *function tCRT.Address(X1, Y1: word): pointer; virtual;*

Returns a pointer to the pixel at (X,Y), or NIL.

Overrides tConsole.Address.

DetectMode *procedure tCRT.DetectMode; virtual;*

Detects the current text mode and adjusts properties.

GotoXY *procedure tCRT.GotoXY(X1, Y1: word); virtual;*

Moves to the specified coordinates.

Overrides tConsole.GotoXY.

Initialize *constructor tCRT.Initialize;*

Initializes a console with the default properties.

Overrides tBoundedConsole.Initialize.

Overrides tConsole.Initialize.

Overrides tComponent.Initialize.

Overrides tMessage.Initialize.

Overrides tObject.Initialize.

StreamLoad *constructor tCRT.StreamLoad(Stream: pStream);*

Constructs and loads an instance from a stream.

Overrides tBoundedConsole.StreamLoad.

Overrides tConsole.StreamLoad.

Overrides tComponent.StreamLoad.

Overrides tStaticElement.StreamLoad.

Overrides tElement.StreamLoad.

Overrides tObject.StreamLoad.

X *function tCRT.X: word; virtual;*

Returns the current X coordinate.

Overrides tConsole.X.

Y *function tCRT.Y: word; virtual;*

Returns the current Y coordinate.

Overrides tConsole.Y.

tCursorList EFLIST

EFLIST.tCursorList = object(tList)

This class defines a linked list with a cursor node. Such a list provides faster access than ordinary linked lists since it uses a cursor to remember the last used node.

Fields and Methods:

constructor *Initialize...*
constructor *Resemble...*
constructor *Duplicate...*
destructor *Intercept*; virtual;
function *IndexedNode...* virtual;
procedure *Touch...* virtual;
constructor *StreamLoad...*
private
fCursor: *pNodeIterator*;

Fields

fCursor *tCursorList.fCursor: pNodeIterator;*

Current node: "cursor" inside the structure.

Methods

Duplicate

constructor tCursorList.Duplicate(ADT: pADT);

Initializes a duplicate ADT instance (with equal elements).

Overrides tList.Duplicate.

IndexedNode

function tCursorList.IndexedNode(Index: word): pLinkage; virtual;

Index ==> node; returns node with specified index.

Overrides tList.IndexedNode.

Initialize

constructor tCursorList.Initialize(SizeOfElements: word);

Initializes an instance of this ADT class.

Overrides tList.Initialize.

Overrides tLinearADT.Initialize.

Overrides tADT.Initialize.

Overrides tObject.Initialize.

Intercept

destructor tCursorList.Intercept; virtual;

Intercepts and destructs an instance of this type.

Overrides tList.Intercept.

Overrides tADT.Intercept.

Overrides tObject.Intercept.

Resemble

constructor tCursorList.Resemble(ADT: pADT);

Initializes an ADT that resembles the specified ADT.

Overrides tList.Resemble.

StreamLoad

constructor tCursorList.StreamLoad(Stream: pStream);

Loads an instance from a stream.

Overrides tList.StreamLoad.

Overrides tADT.StreamLoad.

Overrides tObject.StreamLoad.

Touch

procedure tCursorList.Touch(Node: pLinkage; Index: word); virtual;

Touchs a node, i.e. notifies that it was used.

Overrides tList.Touch.

EFSTREAM.tDataStream = object(tStream)

This class implements a data stream, ie. a stream that operate on arbitrary data specified by a pointer and a size in bytes.

Fields and Methods:

public
constructor *Initialize...*
procedure *Read...* virtual;
procedure *Write...* virtual;
function *Size...* virtual;
function *IsAllocated...* virtual;
private
fData: pointer;
fSize: word;

Fields

fData *tDataStream.fData: pointer;*

Pointer to the streamed data.

fSize *tDataStream.fSize: word;*

Size of the streamed data (bytes).

Methods

Initialize *constructor tDataStream.Initialize(Data: pointer; Length: word);*

Initializes and constructs an instance of this class.

Overrides tStream.Initialize.

Overrides tObject.Initialize.

IsAllocated *function tDataStream.IsAllocated: boolean; virtual;*

Is the stream allocated, ie. ready to be used in some way?

Overrides tStream.IsAllocated.

Read *procedure tDataStream.Read(var Data; Count: word); virtual;*

Reads specified amount of data from this stream.

Overrides tStream.Read.

Size *function tDataStream.Size: longint; virtual;*

Returns the streams size (length) in bytes.

Overrides tStream.Size.

Write *procedure tDataStream.Write(var Data; Count: word); virtual;*

Writes specified data to this stream.

Overrides tStream.Write.

tDate EFTIME

EFTIME.tDate = object(tTiming)

Fields and Methods:

procedure *SetDate...*
function *DayOfWeek...* virtual;
function *DayOfYear...* virtual;
function *IsLeapYear...* virtual;
private
fYear: word;
fMonth: byte;
fDay: byte;

Descendants:

tTimeDate

Fields

fDay *tDate.fDay: byte;*

Day within (1..31).

fMonth *tDate.fMonth: byte;*

Month within (1..12).

fYear *tDate.fYear: word;*

Year within (0..65535).

Methods

DayOfWeek *function tDate.DayOfWeek: byte; virtual;*

Returns the day of the week (1 .. 7).

DayOfYear *function tDate.DayOfYear: word; virtual;*

Returns the day of the year (0 .. 365).

IsLeapYear *function tDate.IsLeapYear: boolean; virtual;*

Is this a leap year?

SetDate *procedure tDate.SetDate(Year, Month, Day: word);*

Sets the date stored in the instance.

tDelayedSearchFilter EFSEARCH

EFSEARCH.tDelayedSearchFilter = object(tSearchFilter)

This is an abstract filter derived from *tSearchFilter*. It uses the Knuth-Morris-Pratt algorithm to delay the writing of data until it has decided if that data match the pattern. If the data match the pattern, the method *Match* is called. *Match* then dumps the matched pattern to the stream. *Match* calls *Dump* to move the currently matched characters to the base stream.

Fields and Methods:

public
procedure *PutByte...* virtual;
function *GetByte...* virtual;
procedure *Match*; virtual;
procedure *Dump...* virtual;
private
fLastAccess: *tAccess*;

Descendants:

tReplaceFilter

Fields

fLastAccess *tDelayedSearchFilter.fLastAccess: tAccess;*

Last access operation (either read or write).

Methods

Dump *procedure tDelayedSearchFilter.Dump(PatternIndex: word); virtual;*

Dump the remaining pattern characters to the base stream.

GetByte *function tDelayedSearchFilter.GetByte: byte; virtual;*

Gets a byte from this stream.

Overrides tMatchFilter.GetByte.

Overrides tStream.GetByte.

Match *procedure tDelayedSearchFilter.Match; virtual;*

Dump the entire pattern to the base stream - it is matched.

PutByte *procedure tDelayedSearchFilter.PutByte(Data: byte); virtual;*

Puts a byte into this stream.

Overrides tMatchFilter.PutByte.

Overrides tStream.PutByte.

tDevice EFDEVICE

EFDEVICE.tDevice = object(tComponent)

This class defines an abstract device. A device is a special kind of component that communicate with a system resource. The following devices are defined in EFLIB: the keyboard (tKeyboardDevice), the mouse (tMouseDevice), the speaker (tSpeakerDevice) and the console (tConsoleDevice).

A device must:

(1) Watch a system resource and post strict event messages when that resource updates its status.

(2) Enable primitive communication with the device.

(3) Respond to a a "is-ready" query. When a touch service message is sent to the device it must reply with a tResponseMessage (if the device is ready) or reject the message (if the device is not ready or locked).

Most devices define new message classes that handles specific events associated to that devices.

Fields and Methods:

public
procedure *Handle...* virtual;
function *IsInstalled...* virtual;
function *IsReady...* virtual;
function *IsCompatible...* virtual;

Descendants:

tSoundDevice
EFMOUSE.tMouseDevice

Methods

Handle *procedure tDevice.Handle(Message: pMessage); virtual;*

Handles messages to this device.

Overrides tComponent.Handle.

IsCompatible *function tDevice.IsCompatible(What: pObject): boolean; virtual;*

Are objects compatible, ie. can they replace each other?

Overrides tStaticElement.IsCompatible.

Overrides tElement.IsCompatible.

Overrides tObject.IsCompatible.

IsInstalled *function tDevice.IsInstalled: boolean; virtual;*

Is the device installed?

IsReady *function tDevice.IsReady: boolean; virtual;*

Is the device ready for use?

tDirection EFDEF

EFDEF.tDirection = (Upward, Downward, Leftward, Rightward);

This type registers the motion direction of a GUI movement (view presentations).

EFIO.tDirectory = object(tObject)

This object handles directories and includes support for wildcards, searching and data type output of file listings.

Fields and Methods:

public
constructor *Initialize...*
procedure *SetPath...* virtual;
procedure *SetMask...* virtual;
procedure *SetAttributeMask...* virtual;
procedure *EnableDirectories*; virtual;
procedure *DisableDirectories*; virtual;
procedure *EnableExclusion*; virtual;
procedure *DisableExclusion*; virtual;
procedure *Start*; virtual;
procedure *Next*; virtual;
function *Filename...* virtual;
function *Fullname...* virtual;
function *Name...* virtual;
function *Extention...* virtual;
function *Path...* virtual;
function *Size...* virtual;
function *Attribute...* virtual;
function *Time...* virtual;
function *Date...* virtual;
function *Entry...* virtual;
function *IsEnd...* virtual;
private
fCurrentEntry: SearchRec;
fMask: PathStr;
fAttributeMask: byte;
function *LeadingZero...* virtual;

Fields

fAttributeMask *tDirectory.fAttributeMask: byte;*

Attribute mask.

fCurrentEntry *tDirectory.fCurrentEntry: SearchRec;*

Directory search profile.

fMask *tDirectory.fMask: PathStr;*

Search specification mask.

Methods

Attribute *function tDirectory.Attribute: byte; virtual;*

Returns the attribute byte for the file.

Date *function tDirectory.Date: string; virtual;*

Returns a date string for the file.

DisableDirectories *procedure tDirectory.DisableDirectories; virtual;*

Disables inclusion of directories.

DisableExclusion *procedure tDirectory.DisableExclusion; virtual;*

Disables exclusion.

EnableDirectories *procedure tDirectory.EnableDirectories; virtual;*

Enables inclusion of directories.

EnableExclusion *procedure tDirectory.EnableExclusion; virtual;*

Enables exclusion of hidden / system files and directories.

Entry *function tDirectory.Entry(Format: string): string; virtual;*

Returns a formatted MS-DOS directory string for the file.

Extention *function tDirectory.Extention: string; virtual;*

Returns the extention of the file.

Filename *function tDirectory.Filename: string; virtual;*

Returns the filename with extention.

Fullname *function tDirectory.Fullname: string; virtual;*

Returns the full filename with extention and path.

Initialize *constructor tDirectory.Initialize(WhatMask: string);*

Initializes an instance with specified directory mask.

Overrides tObject.Initialize.

IsEnd *function tDirectory.IsEnd: boolean; virtual;*

Is the end of the search reached?

LeadingZero *function tDirectory.LeadingZero(Number: word; Digits: byte): string; virtual;*

Converts a number to a two-digit number with leading zeroes.

Name *function tDirectory.Name: string; virtual;*

Returns the name without extention.

Next *procedure tDirectory.Next; virtual;*

Searchs for next entry in the directory.

Path *function tDirectory.Path: string; virtual;*

Returns the path for the file.

SetAttributeMask *procedure tDirectory.SetAttributeMask(WhatMask: byte); virtual;*

Sets the directory attribute mask.

SetMask *procedure tDirectory.SetMask(WhatMask: string); virtual;*

Sets the directory mask, ie. the search preference.

SetPath *procedure tDirectory.SetPath(WhatPath: string); virtual;*

Sets the path for the directory.

Size *function tDirectory.Size: longint; virtual;*

Returns the file size.

Start *procedure tDirectory.Start; virtual;*

Starts a directory search with the current preferences.

Time *function tDirectory.Time: string; virtual;*

Returns a time string for the file.

tDoubleHashTable EFTABLE

EFTABLE.tDoubleHashTable = object(tOpenHashTable)

This class defines an open hash table that uses double hashing to handle collisions. A hash table is a generalization of an array into an ordered ADT (see *tHashTable*).

This is an open hash table with double hashing. All elements are inserted at positions calculated by hash function (see *tHashPlug*). If there occur a collision (a calculated slot is used), the *Rehash* method is called until an empty slot is found. The double hash mechanism uses the collision index (the wrong hash code) to calculate a new hash code. Double hashing is generally more powerful than linear probing (see *tProbeHashTable*), but it is not as rapid as the bucket mechanism in *tBucketHashTable*.

Fields and Methods:

public
constructor *Initialize...*
function *Rehash...* virtual;
constructor *StreamLoad...*

Methods

Initialize

constructor tDoubleHashTable.Initialize(CapacityOfElements, SizeOfElements: word);

Initializes an instance of this ADT class.

Overrides tHashTable.Initialize.

Overrides tCompositeADT.Initialize.

Overrides tADT.Initialize.

Overrides tObject.Initialize.

Rehash

function tDoubleHashTable.Rehash(Index, Collisions: word): word; virtual;

Rehashes a hash code to resolve a collision.

Overrides tOpenHashTable.Rehash.

StreamLoad

constructor tDoubleHashTable.StreamLoad(Stream: pStream);

Loads an ADT from a stream.

Overrides tCompositeADT.StreamLoad.

Overrides tADT.StreamLoad.

Overrides tObject.StreamLoad.

EFFILTER.tDuplicateFilter = object(tFilter)

This filter passes all write operations to two base streams, thus duplicating all information it receives. Reading is always done from the first base stream.

Both streams are initially classified as dependent, that is they are both owned by the filter and are destructed whenever the filter is destructed.

Fields and Methods:

public
fSecondBase: *pStream*;
constructor *Initialize*...
destructor *Intercept*; virtual;
procedure *SetSecondBase*...
procedure *Write*... virtual;
procedure *Reset*; virtual;
procedure *Flush*; virtual;
procedure *Truncate*; virtual;
procedure *Seek*... virtual;
function *IsAllocated*... virtual;

Fields

fSecondBase *tDuplicateFilter.fSecondBase: pStream;*

Second base stream.

Methods

Flush *procedure tDuplicateFilter.Flush; virtual;*

Flushes any buffer that is associated to this stream.

Overrides tFilter.Flush.

Overrides tStream.Flush.

Initialize *constructor tDuplicateFilter.Initialize(First, Second: pStream);*

Initializes an instance as dependent on a base stream.

Overrides tFilter.Initialize.

Overrides tStream.Initialize.

Overrides tObject.Initialize.

Intercept *destructor tDuplicateFilter.Intercept; virtual;*

Intercepts and destructs an instance of this type.

Overrides tFilter.Intercept.

Overrides tStream.Intercept.

Overrides tObject.Intercept.

IsAllocated *function tDuplicateFilter.IsAllocated: boolean; virtual;*

Is the stream allocated, ie. ready to be used in some way?

Overrides tFilter.IsAllocated.

Overrides tStream.IsAllocated.

Reset *procedure tDuplicateFilter.Reset; virtual;*

Resets stream and moves to the first position.

Overrides tStream.Reset.

Seek *procedure tDuplicateFilter.Seek(Where: longint); virtual;*

Seeks, ie. moves to, a position in the stream [0 .. n].

Overrides tFilter.Seek.

Overrides tStream.Seek.

SetSecondBase *procedure tDuplicateFilter.SetSecondBase(Base: pStream);*

Sets the second base stream on which this filter operates.

Truncate *procedure tDuplicateFilter.Truncate; virtual;*

Truncates the stream, ie. deletes remaining data.

Overrides tFilter.Truncate.

Overrides tStream.Truncate.

Write *procedure tDuplicateFilter.Write(var Data; Count: word); virtual;*

Writes specified data to this stream.

Overrides tFilter.Write.

Overrides tStream.Write.

EFELEM.tElement = object(tObject)

This is an abstract element class, that a class that define the properties of elements in data structures. All data structures store there data in elements that descend from this class. This element is abstract. You must override *Data* and *Size*, and add some data members.

Elements must:

(1) Provide the methods *Data* and *Size*. These methods enable the data structure to access the content of an element without knowing what data is stored. Static elements does not have to support these methods. All other elements must provide full access to their contents with the *Data* method.

(2) Know if the element is static. This is done in the *IsStatic* method. Static elements cannot be modified, but they can optionally have some key (accessed with *Data* and *Size*).

(3) Know the relation between the element and some other element. This is done in the *Compare* method. Comparisons result in one of the values [1,0,-1].

(4) Non-static elements must permit content swapping with any compatible element. This is done in the *Swap* method.

(5) Know if an element is compatible with another element. This is done in the *IsCompatible* method. This method should return TRUE if they have the same kind of contents.

Some elements also can:

(6) Return TRUE when *IsPattern* is called and the specified string can be converted to this element. Many elements always returns FALSE since they cannot be reconstructed from a string.

(7) Convert a string to an element when the *Patternize* method is called and *IsPattern* already have returned TRUE.

(8) Read and write the element as formatted text to or from an arbitrary stream whenever *StreamRead* or *StreamWrite* is called.

You sometimes do not have to override all methods in *tElement* to create new element classes. However, you must always override the method *Data* and *Size* for non-static elements.

tElement provide you with a *Compare* method that determine element relations by comparing the content bitwise. Not all data support bitwise comparisons: the relation may be defined in some other way. For example, some Pascal numbers data types (real, double, integer, etc) are stored in a special way that does not support bitwise comparisons.

Fields and Methods:

public
destructor *Intercept*; virtual;
procedure *Copy...* virtual;
procedure *CopyIn...* virtual;
procedure *CopyOut...* virtual;
function *Data...* virtual;
function *Size...* virtual;
function *Length...* virtual;
function *Allocate...* virtual;
procedure *Dispose*; virtual;
procedure *Swap...* virtual;
function *Compare...* virtual;
constructor *StreamLoad...*
procedure *StreamStore...* virtual;
procedure *StreamRead...* virtual;
procedure *StreamWrite...* virtual;
function *IsPattern...* virtual;
function *Patternize...* virtual;
function *IsAllocated...* virtual;
function *IsStatic...* virtual;
function *IsEqual...* virtual;
function *IsCompatible...* virtual;

Descendants:

tCarrierElement
tGenericElement
   EFMEMORY.tAllocation

- *EFMEMORY.tBuffer*
- *EFSTRING.tString*
  - *EFPATRN.tPatternString*
  - *EFSTRING.tToken*

tStaticElement

- *EFCMAN.tMessage*
  - *EFCMAN.tComponent*
    - *EFCMAN.tManager*
      - *EFAPP.tApplication*
      - *EFCMAN.tCommunicator*
    - *EFCONDRV.tConsole*
      - *EFCONDRV.tBoundedConsole*
        - *EFCONDRV.tCRT*
          - *EFSCREEN.tScreen*
        - *EFCONDRV.tVirtualConsole*
          - *EFCONDRV.tImage*
          - *EFSCREEN.tBufferedScreen*
    - *EFDEVICE.tDevice*
      - *EFDEVICE.tSoundDevice*
      - *EFMOUSE.tMouseDevice*
    - *EFVIEWS.tView*
      - *EFVIEWS.tGroup*
  - *EFCMAN.tServiceMessage*
  - *EFDEVICE.tInstallMessage*
  - *EFKEYBRD.tKeyEvent*
  - *EFMOUSE.tMouseEvent*
  - *EFMSG.tFocusedEvent*
  - *EFMSG.tLocalMessage*
- *tIdentityElement*
- *EFMATH.tMathObject*
  - *EFMATH.tComplex*
  - *EFMATH.tInteger*
  - *EFMATH.tMatrix*
    - *EFMATH.tPolynomial*
  - *EFMATH.tRational*
  - *EFMATH.tReal*
  - *EFMATH.tVector*
  - *EFMEXP.tExpression*
  - *EFMEXP.tMathFunction*
- *EFPLUG.tPlug*
  - *EFADT.tADTPlug*
    - *EFADT.tKeyPlug*
      - *EFADT.tBoundedKeyPlug*
      - *EFADT.tReversedKeyPlug*

      - *EFADT.tSortPlug*
        - *EFADT.tQuickSortPlug*
        - *EFSORT.tBubbleSortPlug*
        - *EFSORT.tInsertionSortPlug*
        - *EFSORT.tMergeSortPlug*
      - *EFTABLE.tHashPlug*
        - *EFTABLE.tTextHashPlug*
      - *EFTREE.tPathPlug*
  - *EFSYSTEM.tNamedElement*
    - *EFSYSTEM.tFlag*
    - *EFSYSTEM.tMapping*
    - *EFSYSTEM.tProfile*
      - *EFSYSTEM.tEnvironment*
      - *EFSYSTEM.tSystemProfile*
    - *EFSYSTEM.tSetting*
      - *EFSYSTEM.tComponentSelector*
  - *EFTABLE.tBucket*
  - *EFTIME.tTiming*
    - *EFTIME.tDate*
      - *EFTIME.tTimeDate*
    - *EFTIME.tTime*
- *tStreamElement*
- *EFMEMORY.tStreamHandle*
- *EFSYSTEM.tLocalElement*
- *EFSYSTEM.tNumberElement*

Methods

Allocate *function tElement.Allocate(SizeOfData: word): boolean; virtual;*

Attempts to allocate memory for element contents.

Compare *function tElement.Compare(What: pElement): shortint; virtual;*

Compares elements and returns [1, 0, -1] as result.

Copy *procedure tElement.Copy(Element: pElement); virtual;*

Copies another allocation into this instance (allocates).

CopyIn *procedure tElement.CopyIn(Source: pointer; Count, Position: word); virtual;*

Copies the specified contents into this element.

CopyOut *procedure tElement.CopyOut(Target: pointer; Count, Position: word); virtual;*

Copies (parts of the) contents to some external address.

Data *function tElement.Data(Position: word): pointer; virtual;*

Returns a pointer to contents at some position.

Dispose *procedure tElement.Dispose; virtual;*

Disposes any data allocation or erases the contents.

Intercept *destructor tElement.Intercept; virtual;*

Intercepts and destructs an element instance.

Overrides tObject.Intercept.

IsAllocated *function tElement.IsAllocated: boolean; virtual;*

Is this element allocated, ie. has some contents?

IsCompatible *function tElement.IsCompatible(Instance: pObject): boolean; virtual;*

Are these elements compatible?

Overrides tObject.IsCompatible.

IsEqual *function tElement.IsEqual(Instance: pObject): boolean; virtual;*

Are the contents of these elements equal?

Overrides tObject.IsEqual.

IsPattern *function tElement.IsPattern(Pattern: pElement): boolean; virtual;*

Can this element be rebuilt from the specified pattern?

IsStatic *function tElement.IsStatic: boolean; virtual;*

Is this element static, that is cannot be modified?

Length *function tElement.Length: word; virtual;*

Returns length (number of used bytes after position 0).

Patternize *function tElement.Patternize(Pattern: pElement): boolean; virtual;*

Attempts to rebuild element from a pattern.

Size *function tElement.Size: word; virtual;*

Returns the size of the contents of this element in bytes.

StreamLoad *constructor tElement.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

Overrides tObject.StreamLoad.

StreamRead *procedure tElement.StreamRead(Stream: pStream); virtual;*

Reads formatted text from a stream and creates an element.

StreamStore *procedure tElement.StreamStore(Stream: pStream); virtual;*

Stores an instance to a stream.

Overrides tObject.StreamStore.

StreamWrite *procedure tElement.StreamWrite(Stream: pStream); virtual;*

Writes formatted text that describe the element contents.

Overrides tObject.StreamWrite.

Swap *procedure tElement.Swap(What: pElement); virtual;*

Swaps contents of current element with the specified.

tEncryptFilter EFFILTER

EFFILTER.tEncryptFilter = object(tFilter)

This class defines an encryption filter, that is an filter that automatically encodes all information that are written and decodes immediately after the information have been read.

Encryption normally uses a keyword. A keyword is a Pascal string of at the most 255 character. You can also specify an arbitrary longint value (the seed) for higher safety. tEncryptFilter supply you with a *Scramble* method that encrypts a keyword with another keyword. This method can be used to prevent any other program from encrypting your data, even if they know the keyword and use EFLIB.

This filter uses the built-in Pascal random number generator to scramble the information.

Fields and Methods:

public
constructor *Initialize...*
procedure *SetKey...*
procedure *SetSeed...*
procedure *Read...* virtual;
procedure *Write...* virtual;
procedure *Seek...* virtual;
function *Scramble...*
private
fSeed: longint;
procedure *Encrypt...*
procedure *Cycle...*
function *Seed...*

Fields

fSeed *tEncryptFilter.fSeed: longint;*

Current seed for the Pascal random number generator.

Methods

Cycle *procedure tEncryptFilter.Cycle(Cycles: longint);*

Cycles the seed to simulate sequential reading or writing.

Encrypt *procedure tEncryptFilter.Encrypt(var Data; Count: word);*

Encrypts or decrypts the specified data.

Initialize *constructor tEncryptFilter.Initialize(Keyword: string; Stream: pStream);*

Initializes an encryption filter with a keyword.

Overrides tFilter.Initialize.

Overrides tStream.Initialize.

Overrides tObject.Initialize.

Read *procedure tEncryptFilter.Read(var Data; Count: word); virtual;*

Reads specified amount of data from this stream.

Overrides tFilter.Read.

Overrides tStream.Read.

Scramble *function tEncryptFilter.Scramble(A, K: string): string;*

Creates a scrambled key B out of two keys A and K.

Seed *function tEncryptFilter.Seed(Keyword: string): longint;*

Calculates the random seed from the specified key.

Seek *procedure tEncryptFilter.Seek(Where: longint); virtual;*

Seeks, ie. moves to, a position in the stream [0 .. n].

Overrides tFilter.Seek.

Overrides tStream.Seek.

SetKey *procedure tEncryptFilter.SetKey(Keyword: string);*

Sets encryption keyword (an arbitrary string).

SetSeed *procedure tEncryptFilter.SetSeed(SeedNumber: longint);*

Sets encryption seed (a number instead of keyword).

Write *procedure tEncryptFilter.Write(var Data; Count: word); virtual;*

Writes specified data to this stream.

Overrides tFilter.Write.

Overrides tStream.Write.

tEnvironment EFSYSTEM

EFSYSTEM.tEnvironment = object(tProfile)

Fields and Methods:

public
constructor *Initialize*;
destructor *Intercept*; virtual;

Methods

Initialize *constructor tEnvironment.Initialize;*

Initializes the default environment profile.

Overrides tNamedElement.Initialize.

Overrides tObject.Initialize.

Intercept *destructor tEnvironment.Intercept; virtual;*

Intercepts the environment profile.

Overrides tElement.Intercept.

Overrides tObject.Intercept.

EFKERNEL.tError = object(tObject)

This class defines an error. tError associates an error to an identity (word-sized, positive integer values within [0, 65535]) and to some action (defined in *Action*).

tError is responsible for the handling of an error. The method *Action* must guarantee that further execution will be safe, or the program must terminate - optionally with an error message.

tError is an abstract base class for all errors. You can create new errors and even reassign old identities. Thus, you can control the behavior of EFLIB when an error occurr.

Fields and Methods:

public
constructor *Initialize...*
procedure *Action...* virtual;
function *Identity...* virtual;
function *IsFatal...* virtual;
private
fIdentity: word;

Descendants:

tErrorMessage

Fields

fIdentity *tError.fIdentity: word;*

Specific identity number for this error.

Methods

Action *procedure tError.Action(Component: pObject; ForceNonFatal: boolean); virtual;*

Perform some action (handle this error).

Identity *function tError.Identity: word; virtual;*

Returns the identity number associated to this error.

Initialize *constructor tError.Initialize(ErrorIdentity: word);*

Initializes an error with the specified identity.

Overrides tObject.Initialize.

IsFatal *function tError.IsFatal: boolean; virtual;*

Is this error fatal - i.e. requires program termination?

tErrorHandler EFKERNEL

EFKERNEL.tErrorHandler = object(tCarrierRegister)

This is the error handler, the class that is called if some component in EFLIB report an error or an invalid assertion. If an error occurr, this class decides what actions shall be taken. If the error is fatal, it must result in program termination.

All errors are maintained as independent *tError* instances. The action that is associated to a specific error identity, is defined in *tError* - or some descendant to that class. Normally, the descendant *tErrorMessage* is used. This class simply prints an error message and (optionally) terminates the program.

Not all errors result in program termination. Such errors are called non-fatal. When a non-fatal error is reported, it is moved into a report queue. These errors are not triggered until a fatal error occurr. That is, there can be non-fatal errors that are not reported, but whenever a fatal error occurr, all these errors are triggered together with the fatal error.

You can override this behavior and force any error to be reported immediately by enabling the ReportAllErrors setting.

Fields and Methods:

public
constructor *Initialize*;
destructor *Intercept*; virtual;
procedure *Handle...* virtual;
procedure *CreateMessage...* virtual;
procedure *Register...* virtual;
procedure *Delay...* virtual;
function *Search...* virtual;
constructor *StreamLoad...*
function *IsDelayed...* virtual;
function *IsValid...* virtual;
private
fDelayed: *pCarrierRegister*;

Fields

fDelayed *tErrorHandler.fDelayed: pCarrierRegister;*

Delayed errors (non-fatal errors).

Methods

CreateMessage *procedure tErrorHandler.CreateMessage(Identity: word; Message: string; Fatal: boolean); virtual;*

Registers a new error message (tErrorMessage instance).

Delay *procedure tErrorHandler.Delay(Identity: word); virtual;*

Register an error (identity) as a delayed error.

Handle *procedure tErrorHandler.Handle(Identity: word; Component: pObject; ForceNonFatal: boolean); virtual;*

Handles the specified error (identity number).

Initialize *constructor tErrorHandler.Initialize;*

Initializes and constructs an error handler.

Overrides tCarrierRegister.Initialize.

Overrides tObject.Initialize.

Intercept *destructor tErrorHandler.Intercept; virtual;*

Intercepts and destructs the error handler with mes+sages.

Overrides tCarrierRegister.Intercept.

Overrides tObject.Intercept.

IsDelayed *function tErrorHandler.IsDelayed(IdentityOfError: word): boolean; virtual;*

Is the specified error waiting to be handled?

IsValid *function tErrorHandler.IsValid(What: pObject): boolean; virtual;*

Can this instance (class) be registered - is it valid?

Overrides tCarrierRegister.IsValid.

Overrides tClassRegister.IsValid.

Register *procedure tErrorHandler.Register(What: pObject); virtual;*

Registers a new tError instance with some error identity.

Overrides tCarrierRegister.Register.

Overrides tClassRegister.Register.

Search *function tErrorHandler.Search(Identity: longint; Class: pointer): pCarrier; virtual;*

Searchs for the carrier of the specified identity / class.

Overrides tCarrierRegister.Search.

StreamLoad *constructor tErrorHandler.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

Overrides tCarrierRegister.StreamLoad.

Overrides tObject.StreamLoad.

tErrorMessage EFKERNEL

EFKERNEL.tErrorMessage = object(tError)

This class defines a error with an associated message (the description of the error). The message is printed if the error is triggered. Also, this class knows if the error require program termination (is fatal). See *tError* for details.

Fields and Methods:

public
constructor *Initialize...*

destructor *Intercept*; virtual;
procedure *SetMessage*... virtual;
procedure *Action*... virtual;
function *Message*... virtual;
function *IsFatal*... virtual;
private
fMessage: pChar;
fFatal: boolean;

Fields

fFatal *tErrorMessage.fFatal: boolean;*

Fatal flag: set if this error require program termination.

fMessage *tErrorMessage.fMessage: pChar;*

Error message assigned to this error.

Methods

Action *procedure tErrorMessage.Action(Component: pObject; ForceNonFatal: boolean); virtual;*

Perform some action (handle this error).

Overrides tError.Action.

Initialize *constructor tErrorMessage.Initialize(ErrorIdentity: word; ErrorMessage: string; RequireTermination: boolean);*

Initializes an error with the specified identity.

Overrides tError.Initialize.

Overrides tObject.Initialize.

Intercept *destructor tErrorMessage.Intercept; virtual;*

Intercepts and destructs an instance of this class.

Overrides tObject.Intercept.

IsFatal *function tErrorMessage.IsFatal: boolean; virtual;*

Is this error fatal - i.e. requires program termination?

Overrides tError.IsFatal.

Message *function tErrorMessage.Message: string; virtual;*

Returns the message associated to this error.

SetMessage *procedure tErrorMessage.SetMessage(ErrorMessage: string); virtual;*

Sets the error message (the description of this error).

EFMEXP.tEvaluator = object(tRegister)

This is the expression evaluator class. It is responsible for the evaluation of expression, that is it converts a expressions (instances of *tExpression*) into numbers.

The evaluator can be configurated at run-time since new operators can be attached. Operations for standard arithetmics are built-in and need not be attached. There is a set of external operators that are installed automatically in the evaulator. These are operators for common trigonometry, logarithms and powers.

You can replace any operator, even those that are built-in, by installing new operator classes. Installed operators are choosen in front of the built-in. If you install an operator that already is installed (though in some other form), the old operator is automatically removed. This technology is known as the plug-in technology. See tPlugManager and tPlug.

Fields and Methods:

public
function *Evaluate...* virtual;
private
fOperators: pOrderedADT;

Fields

fOperators

tEvaluator.fOperators: pOrderedADT;

Table with operators classes.

Methods

Evaluate

function tEvaluator.Evaluate(Expression: pExpression; var Result: pMathObject): boolean; virtual;

Evaluates the specified expression and returns results.

tExpression EFMEXP

EFMEXP.tExpression = object(tMathObject)

This class defines a mathematical expression stored in reveresed polish notation (RPN). This class provide a *tMathObject* interface for expressions. Thus, you can use an expression as if it would be any number. Whenever some process want to access the expression, it is automatically evaluated.

Evauluation is done in the *tEvaluator* class. All used variables must be defined in the expression, otherwise it cannot be evaluated.

Fields and Methods:

public
procedure *Add...* virtual;
procedure *Subtract...* virtual;
procedure *Multiply...* virtual;
procedure *Divide...* virtual;
function *IsZero...*
private
fOperators: *pStack*;
fOperands: *pStack*;

Fields

fOperands *tExpression.fOperands: pStack;*

Pointer to a stack with math object classes.

fOperators *tExpression.fOperators: pStack;*

Pointer to a stack with math operator classes.

Methods

Add *procedure tExpression.Add(What: pMathObject); virtual;*

Adds an object to this object.

Overrides tMathObject.Add.

Divide *procedure tExpression.Divide(What: pMathObject); virtual;*

Divides this object by another object.

Overrides tMathObject.Divide.

IsZero *function tExpression.IsZero: boolean;*

Is number equal to zero?

Overrides tMathObject.IsZero.

Multiply *procedure tExpression.Multiply(What: pMathObject); virtual;*

Multiplies this object with another object.

Overrides tMathObject.Multiply.

Subtract *procedure tExpression.Subtract(What: pMathObject); virtual;*

Subtracts an object from this object.

Overrides tMathObject.Subtract.

TextChars EFDEF

Const EFDEF.TextChars: tChars = ['A'..'Z', 'a'..'z', '', '', '', '', '', '', '', '', '', #32] ;

Text character set: all valid text characters for the English, Swedish and German alphabet. This constant is extended according to national requirements.

tExternalF EFMATH

EFMATH.tExternalF = function(X: tBaseReal): tBaseReal;

tExternalH EFMATH

EFMATH.tExternalH = function(X, Y: tBaseReal): tBaseReal;

Mathematical function type with two inputs (x, y). This type declaration is used by *tSolver* for external functions (used by Euler's and Heun's method only).

tField EFBASIC

EFBASIC.tField = object(tContainer)

This class defines a field, ie. a region represented by two coordinate pairs: (X,Y)-(X,Y). The first pair is the start of the field (fStart), and the second the stop of the field (fStop).

Fields can attach to other fields, hence forming a more generic region (not limited to four corners). The method *Delta* uses this technique, known as the delta technique.

Fields and Methods:

public
fStart: *tCoordinates*;
fStop: *tCoordinates*;
constructor *Initialize...*
constructor *Duplicate...*
procedure *Copy...*
procedure *SetCoordinates...*
procedure *Move...*
procedure *Grow...*
procedure *Union...*
procedure *Intersection...*
procedure *Delta...*
procedure *Adjust*;
procedure *Normalize*;
constructor *StreamLoad...*
procedure *StreamStore...* virtual;
procedure *StreamWrite...* virtual;
function *IsEqual...* virtual;
function *IsCompatible...* virtual;
function *Height...* virtual;
function *Width...* virtual;
function *IsEmpty...* virtual;
function *IsValid...* virtual;
function *IsInside...* virtual;
function *IsFieldInside...* virtual;
function *IsInsideBoundaries...* virtual;
function *IsCoordinatesInside...* virtual;

Fields

fStart *tField.fStart: tCoordinates;*

Upper-left coordinate pair.

fStop *tField.fStop: tCoordinates;*

Lower-right coordinate pair.

Methods

Adjust *procedure tField.Adjust;*

Adjusts the field (fStart smaller or equal to fStop).

Copy *procedure tField.Copy(Field: pField);*

Copies the specified field to this instance.

Delta *procedure tField.Delta(Exclude: pField; Result: pField);*

Returns a chain of four fields, the exact intersection.

Duplicate *constructor tField.Duplicate(Field: pField);*

Initializes and duplicates an instance of this class.

Grow *procedure tField.Grow(DeltaX, DeltaY: integer);*

Resizes the field according to a growth ratio.

Height *function tField.Height: word; virtual;*

Returns the height of the field.

Initialize *constructor tField.Initialize(X1, Y1, X2, Y2: word);*

Initializes and constructs an instance of this class.

Overrides tContainer.Initialize.

Overrides tObject.Initialize.

Intersection *procedure tField.Intersection(Field: pField);*

Makes this the intersection of this and another field.

IsCompatible *function tField.IsCompatible(Instance: pObject): boolean; virtual;*

Are objects compatible, ie. can they replace each other?

Overrides tObject.IsCompatible.

IsCoordinatesInside *function tField.IsCoordinatesInside(nX, nY: word): boolean; virtual;*

Are coordinates inside the specified field?

IsEmpty *function tField.IsEmpty: boolean; virtual;*

Is the field empty (start smaller than or equal to stop).

IsEqual *function tField.IsEqual(Instance: pObject): boolean; virtual;*

Are contents of these instances equal (bitwise compare)?

Overrides tObject.IsEqual.

IsFieldInside *function tField.IsFieldInside(SubField: pField): boolean; virtual;*

Is the specified field inside this field?

IsInside *function tField.IsInside(Coordinates: pCoordinates): boolean; virtual;*

Is the specified coordinates inside this field?

IsInsideBoundaries *function tField.IsInsideBoundaries(X1, Y1, X2, Y2: word): boolean; virtual;*

Is this field inside the specified boundaries?

IsValid *function tField.IsValid: boolean; virtual;*

Are the fields boundaries valid?

Move *procedure tField.Move(DeltaX, DeltaY: integer);*

Moves the field the specified distance.

Normalize *procedure tField.Normalize;*

Normalizes the field: fStart = (1,1) and fStop arbitrary.

SetCoordinates *procedure tField.SetCoordinates(X1, Y1, X2, Y2: word);*

Sets the field coordinates for this instance.

StreamLoad *constructor tField.StreamLoad(Stream: pStream);*

Constructs and loads an instance from a stream.

Overrides tObject.StreamLoad.

StreamStore *procedure tField.StreamStore(Stream: pStream); virtual;*

Stores an instance to a stream.

Overrides tObject.StreamStore.

StreamWrite *procedure tField.StreamWrite(Stream: pStream); virtual;*

Writes a field text to the stream, ie. "(X, Y)-(X, Y)".

Overrides tObject.StreamWrite.

Union *procedure tField.Union(Field: pField);*

Makes this the union of this and another field.

Width *function tField.Width: word; virtual;*

Returns the width of the field.

Overrides tContainer.Width.

EFFILE.tFile = object(tBufferFilter)

tFile is a general file stream class that provide both read and write access. tFile can use a buffer that speeds up intesive access, or it can operate directly on the file (specify a zero buffer when you construct such a file).

tFile automatically creates any non-existing file when it is constructed. Make sure to destruct all files, since the file then is closed and any buffer is flushed.

Fields and Methods:

public
constructor *Initialize...*

Descendants:

tIFile
tOFile
tTemporaryFile
EFRES.tStreamableFile

Methods

Initialize *constructor tFile.Initialize(Filename: string; SizeOfBuffer: word);*

Initializes a file stream, optionally with an buffer.

Overrides tBufferFilter.Initialize.

Overrides tFilter.Initialize.

Overrides tStream.Initialize.

Overrides tObject.Initialize.

tFilename EFIO

EFIO.tFilename = object(tObject)

Fields and Methods:

public
private

Descendants:

tPath

tFileStream EFSTREAM

EFSTREAM.tFileStream = object(tStream)

tFileStream is a general file stream class that provide both read and write access. tFileStream automatically creates any non-existing file when it is constructed.

tFileStream is not buffered. See tFile in *EFFILE* for buffered file handling with optimized performance.

Fields and Methods:

public
constructor *Initialize...*
destructor *Intercept*; virtual;
procedure *Read...* virtual;
procedure *Write...* virtual;
procedure *Reset*; virtual;
procedure *Create*; virtual;
procedure *Truncate*; virtual;
function *Size...* virtual;
private
fHandle: file;

Fields

fHandle *tFileStream.fHandle: file ;*

File handle (internal only).

Methods

Create *procedure tFileStream.Create; virtual;*

Creates a file with this name.

Initialize *constructor tFileStream.Initialize(Filename: string);*

Initializes a file stream and opens a file.

Overrides tStream.Initialize.

Overrides tObject.Initialize.

Intercept *destructor tFileStream.Intercept; virtual;*

Intercepts this file stream and closes the file.

Overrides tStream.Intercept.

Overrides tObject.Intercept.

Read *procedure tFileStream.Read(var Data; Count: word); virtual;*

Reads data from stream into a variable.

Overrides tStream.Read.

Reset *procedure tFileStream.Reset; virtual;*

Resets stream and moves to the first position.

Overrides tStream.Reset.

Size *function tFileStream.Size: longint; virtual;*

Size of file in bytes.

Overrides tStream.Size.

Truncate *procedure tFileStream.Truncate; virtual;*

Truncates the stream at the current position.

Overrides tStream.Truncate.

Write *procedure tFileStream.Write(var Data; Count: word); virtual;*

Writes data to stream into a variable.

Overrides tStream.Write.

tFillStyle EFCONDRV

EFCONDRV.tFillStyle = object(tNamedElement)

This class defines a fill style. It is abstract. A fill style is a way of fill a rectangular area in an arbitrary console. Descendants must override the *Fill* method. See tNamedElement for more information.

Fields and Methods:

public
constructor *Initialize...*
procedure *Fill...* virtual;
constructor *StreamLoad...*

Methods

Fill *procedure tFillStyle.Fill(Console: pConsole; X1, Y1, X2, Y2: word; Color: pColor); virtual;*

Fills a region of the console with some pattern.

Initialize *constructor tFillStyle.Initialize(NameOfFont: string);*

Initializes a font with the specified name.

StreamLoad *constructor tFillStyle.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

tFilter EFFILTER

EFFILTER.tFilter = object(tStream)

This class defines an abstract filter class. A filter is a stream that operate on some physical information, the base stream. A filter provide a virtual representation for the base stream, that is the filter translates the data automatically.

A filter knows its offset it the base stream. All positions are relative to that offset.

You can create your own filters by overriding the method *Read* and *Write*.

Fields and Methods:

public
fBase: *pStream*;
constructor *Initialize...*
destructor *Intercept*; virtual;
procedure *SetMode...* virtual;
procedure *SetBase...* virtual;
procedure *EnableDependence*; virtual;
procedure *DisableDependence*; virtual;
procedure *Read...* virtual;
procedure *Write...* virtual;
function *GetString...* virtual;
procedure *Flush*; virtual;
procedure *Truncate*; virtual;
procedure *Seek...* virtual;
function *Mode...* virtual;

function *Size...* virtual;
function *Position...* virtual;
function *IsAllocated...* virtual;
function *IsDependent...* virtual;
private
fBaseOffset: longint;
fDependent: boolean;

Descendants:

EFCONV.tConverter
tBufferFilter
  EFFILE.tFile
    EFFILE.tIFile
    EFFILE.tOFile
    EFFILE.tTemporaryFile
    EFRES.tStreamableFile
tDuplicateFilter
tEncryptFilter
tSequentialFilter
  tCompressFilter
  tCRC16Filter
    tCRC32Filter
tTextFilter
EFRES.tResource
  EFRES.tArchive
EFSEARCH.tMatchFilter
  EFSEARCH.tSearchFilter
    EFSEARCH.tDelayedSearchFilter
      EFSEARCH.tReplaceFilter
EFSYSTEM.tSwapFilter

Fields

fBase *tFilter.fBase: pStream;*

Base stream.

fBaseOffset *tFilter.fBaseOffset: longint;*

Start offset in base.

fDependent *tFilter.fDependent: boolean;*

Base dependent status.

Methods

DisableDependence *procedure tFilter.DisableDependence; virtual;*

Makes the base stream independent (not destructed).

EnableDependence *procedure tFilter.EnableDependence; virtual;*

Makes the base stream dependent (owned by the filter).

Flush *procedure tFilter.Flush; virtual;*

Flushes any buffer that is associated to this stream.

Overrides tStream.Flush.

GetString *function tFilter.GetString(Delimeter: string): string; virtual;*

Gets a text string from the stream, separated by delimeter.

Overrides tStream.GetString.

Initialize *constructor tFilter.Initialize(Stream: pStream);*

Initializes an instance as dependent on a base stream.

Overrides tStream.Initialize.

Overrides tObject.Initialize.

Intercept *destructor tFilter.Intercept; virtual;*

Intercepts and destructs an instance of this type.

Overrides tStream.Intercept.

Overrides tObject.Intercept.

IsAllocated *function tFilter.IsAllocated: boolean; virtual;*

Is the stream allocated, ie. ready to be used in some way?

Overrides tStream.IsAllocated.

IsDependent *function tFilter.IsDependent: boolean; virtual;*

Is the base stream owned by the filter?

Mode *function tFilter.Mode: tAccess; virtual;*

Returns the access mode for this stream.

Overrides tStream.Mode.

Position *function tFilter.Position: longint; virtual;*

Returns current position in this stream.

Overrides tStream.Position.

Read *procedure tFilter.Read(var Data; Count: word); virtual;*

Reads specified amount of data from this stream.

Overrides tStream.Read.

Seek *procedure tFilter.Seek(Where: longint); virtual;*

Seeks, ie. moves to, a position in the stream [0 .. n].

Overrides tStream.Seek.

SetBase *procedure tFilter.SetBase(Base: pStream; Offset: word); virtual;*

Sets the base stream on which this filter operates.

SetMode *procedure tFilter.SetMode(Access: tAccess); virtual;*

Sets access mode, ie. the way the stream can be accessed.

Overrides tStream.SetMode.

Size *function tFilter.Size: longint; virtual;*

Returns the streams size (length) in bytes.

Overrides tStream.Size.

Truncate *procedure tFilter.Truncate; virtual;*

Truncates the stream, ie. deletes remaining data.

Overrides tStream.Truncate.

Write *procedure tFilter.Write(var Data; Count: word); virtual;*

Writes specified data to this stream.

Overrides tStream.Write.

tFlag EFSYSTEM

EFSYSTEM.tFlag = object(tNamedElement)

This class controls a flag, that is a text associated to a boolean state. The flag can be on (TRUE) or off (FALSE).

Fields and Methods:

public
constructor *Initialize...*
procedure *On*; virtual;
procedure *Off*; virtual;
function *IsOn...* virtual;
private
fOn: boolean;

Fields

fOn *tFlag.fOn: boolean;*

Methods

Initialize *constructor tFlag.Initialize(Name: string; State: boolean);*

Initializes the a flag with name and state.

Overrides tNamedElement.Initialize.

Overrides tObject.Initialize.

IsOn *function tFlag.IsOn: boolean; virtual;*

Is this flag enabled, that is turned on.

Off *procedure tFlag.Off; virtual;*

Disables this flag, that is turns it off.

On *procedure tFlag.On; virtual;*

Enables this flag, that is turns it on.

tFocusedEvent EFMSG

EFMSG.tFocusedEvent = object(tMessage)

Fields and Methods:

private
fPhase: *tProcessPhase*;

Fields

fPhase *tFocusedEvent.fPhase: tProcessPhase;*

EFCONDRV.tFont = object(tNamedElement)

This is an abstract class that defines a font. A font is a simply a way of writing text to a console, using a certain color. Fonts must have the following properties:

(1) They must know how to display themself in an arbitrary console instance.

(2) They must know the height of characters (all characters must have the same height).

(3) They must know the width of a text - that is, a font must know the width of all its characters.

(4) They must know what character are valid for the font - and return them in a character set on demand.

Fields and Methods:

public
constructor *Initialize...*
procedure *Write...* virtual;
constructor *StreamLoad...*
function *TextHeight...* virtual;
function *TextWidth...* virtual;
function *IsFixedWidth...* virtual;

Methods

Initialize *constructor tFont.Initialize(NameOfFont: string);*

Initializes a font with the specified name.

IsFixedWidth *function tFont.IsFixedWidth: boolean; virtual;*

Returns TRUE if this font has only one character width.

StreamLoad *constructor tFont.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

TextHeight *function tFont.TextHeight: word; virtual;*

Returns the height of an arbitrary text character.

TextWidth *function tFont.TextWidth(Text: pString): word; virtual;*

Returns the width of the specified text.

Write *procedure tFont.Write(Console: pConsole; X, Y: word; Color: pColor; Text: pString); virtual;*

Writes text using this font in the specified console.

tGenericElement EFELEM

EFELEM.tGenericElement = object(tElement)

This class handles a generic element for data structures. It uses a dynamic memory allocation for the content, and is this capable of storing anything - even instances.

tGenericElement is the default element in all of EFLIB′s data structures. Remember, that this element does not know what it actually is storing - it can be a pointer, some variables or even instances - you are responsible for interception of these pointers, etc.

Fields and Methods:

public
fData: pointer;
fSize: word;
constructor *Initialize...*
constructor *InitializeEmpty*;
constructor *Duplicate...*
function *Data...* virtual;
function *Size...* virtual;
function *Allocate...* virtual;
procedure *Dispose*; virtual;
procedure *Swap...* virtual;
constructor *StreamLoad...*
procedure *StreamStore...* virtual;
function *IsAllocated...* virtual;
function *IsFree...* virtual;
function *IsValidSize...* virtual;
function *IsCompatible...* virtual;

Descendants:

EFMEMORY.tAllocation
  EFMEMORY.tBuffer
EFSTRING.tString
  EFPATRN.tPatternString

Fields

fData *tGenericElement.fData: pointer;*

Pointer to the allocated block of memory.

fSize *tGenericElement.fSize: word;*

Size of the owned memory block.

Methods

Allocate *function tGenericElement.Allocate(SizeOfData: word): boolean; virtual;*

Attempts to allocate memory for some data.

Overrides tElement.Allocate.

Data *function tGenericElement.Data(Position: word): pointer; virtual;*

Returns a pointer to contents at some position.

Overrides tElement.Data.

Dispose *procedure tGenericElement.Dispose; virtual;*

Disposes any data allocation or erases the contents.

Overrides tElement.Dispose.

Duplicate *constructor tGenericElement.Duplicate(What: pElement);*

Initializes and constructs a duplicate element.

Initialize *constructor tGenericElement.Initialize(Source: pointer; SizeOfData: word);*

Constructs an memory allocation of the specified size.

Overrides tObject.Initialize.

InitializeEmpty *constructor tGenericElement.InitializeEmpty;*

Constructs an empty tAllocation instance ready to be used.

IsAllocated *function tGenericElement.IsAllocated: boolean; virtual;*

Is this element allocated, ie. has some contents?

Overrides tElement.IsAllocated.

IsCompatible *function tGenericElement.IsCompatible(Instance: pObject): boolean; virtual;*

Are these elements compatible?

Overrides tElement.IsCompatible.

Overrides tObject.IsCompatible.

IsFree *function tGenericElement.IsFree(SizeOfData: word): boolean; virtual;*

Can this element allocate the specified memory?

IsValidSize *function tGenericElement.IsValidSize(SizeOfData: word): boolean; virtual;*

Can this element handle contents of the specified size?

Size *function tGenericElement.Size: word; virtual;*

Returns the size of the contents of this element in bytes.

Overrides tElement.Size.

StreamLoad *constructor tGenericElement.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

Overrides tElement.StreamLoad.

Overrides tObject.StreamLoad.

StreamStore *procedure tGenericElement.StreamStore(Stream: pStream); virtual;*

Stores an instance to a stream.

Overrides tElement.StreamStore.

Overrides tObject.StreamStore.

Swap *procedure tGenericElement.Swap(What: pElement); virtual;*

Swaps contents of current element with the specified.

Overrides tElement.Swap.

EFVIEWS.tGroup = object(tView)

This class defines a group of subviews, though itself also is a view. In addition, tGroup can own

(the components).

This class defines a group of subviews and components. A group is simply a *tView* with built-in *tManager* features.

Technically, a group owns a *tCommunicator* instance with components (or subviews). The communicator is installed with the group as a filter. That is, messages passed to the communicator are first passed to the tGroup itself.

Fields and Methods:

public
procedure *Arrange...* virtual;
private
fCommunicator: *pCommunicator*;

Fields

fCommunicator *tGroup.fCommunicator: pCommunicator;*

Component communicator with a (optional) component network.

Methods

Arrange *procedure tGroup.Arrange(Arrangement: tArrangement); virtual;*

Arranges all subviews in the specified order.

tHandle EFKERNEL

EFKERNEL.tHandle = object(tObject)

Fields and Methods:

public
private

fHandle: longint;

EFREG.tHandle = object(tObject)

Fields and Methods:

public
private
fHandle: longint;

Fields

fHandle *tHandle.fHandle: longint;*

fHandle *tHandle.fHandle: longint;*

tHashPlug EFTABLE

EFTABLE.tHashPlug = object(tADTPlug)

Hash function class. This class defines a hash function, that is, a function with somekind of algorithm that associates a virtual index value to an arbitrary element. These indexes are slots or buckets in a hash table. tHashFunction is not compatible with *tKeyPlug*. There can exist both a *tKeyPlug* and a *tHashPlug* in the plug manager associated to *tHashTable*.

This is the default hash function. It simply returns a sum of the bytes in the element (key-region) odulus the capacity of the hash table. In most cases, you should choose other, more efficient hash functions.

Fields and Methods:

public
constructor *Initialize...*
procedure *Install...* virtual;
function *Hash...* virtual;
constructor *StreamLoad...*
function *TypeOfPlug...* virtual;

Descendants:

tTextHashPlug

Methods

Hash *function tHashPlug.Hash(var Key; KeySize: word): word; virtual;*

Returns a virtual index for the specified key.

Initialize *constructor tHashPlug.Initialize(PlugManager: pPlugManager);*

Initializes a hash function plug and attempts to install.

Overrides tPlug.Initialize.

Overrides tObject.Initialize.

Install *procedure tHashPlug.Install(PlugManager: pPlugManager); virtual;*

Installs the plug in the specified plug manager.

Overrides tADTPlug.Install.

Overrides tPlug.Install.

StreamLoad *constructor tHashPlug.StreamLoad(Stream: pStream);*

Constructs and loads an instance from a stream.

Overrides tPlug.StreamLoad.

Overrides tStaticElement.StreamLoad.

Overrides tElement.StreamLoad.

Overrides tObject.StreamLoad.

TypeOfPlug *function tHashPlug.TypeOfPlug: string; virtual;*

The classification of this plug, e.g. "tSortPlug".

Overrides tADTPlug.TypeOfPlug.

Overrides tPlug.TypeOfPlug.

tHashTable EFTABLE

EFTABLE.tHashTable = object(tCompositeADT)

This class defines an abstract hash table ADT. A hash table is a generalization of an array into an ordered ADT.

Instead of using integer indexing, the values of the elements themselves are transformed, using a function called a hash function, into an integer index value. When elements are stored in single slots the hash table is said to be open. If elements are stored in a special ADTs the hash table is said to use buckets (see *tHashTable*).

Hash tables can only store one copy of a certain element key. If an element match an existing key in the table, the new element must replace the existing - the existing element is then destructed. This procedure is fully automized in EFLIB.

Hash tables provide rapid access to scattered storage, but require a good hash function. The hash function interface is defined in *tHashPlug* which also is the default hash function.

Complexity: O(1) - O(log N).

Fields and Methods:

public
constructor *Initialize...*
function *HashCode...* virtual;
procedure *Rebuild*; virtual;
procedure *Resize...* virtual;
function *CreateComposite...* virtual;
function *CreateHashPlug...* virtual;
function *IsInside...* virtual;

Descendants:

tBucketHashTable
tOpenHashTable
  tDoubleHashTable
  tProbeHashTable

Methods

CreateComposite *function tHashTable.CreateComposite(NumberOfElements: word): pLinearADT; virtual;*

Creates an instance of the composite ADT.

Overrides tCompositeADT.CreateComposite.

CreateHashPlug *function tHashTable.CreateHashPlug: pHashPlug; virtual;*

Creates the default hash function (tHashFunction).

HashCode *function tHashTable.HashCode(var Key; KeySize: word): word; virtual;*

Returns the hash code for the specified element key.

Initialize *constructor tHashTable.Initialize(CapacityOfElements, SizeOfElements: word);*

Initializes an instance of this ADT class.

Overrides tCompositeADT.Initialize.

Overrides tADT.Initialize.

Overrides tObject.Initialize.

IsInside *function tHashTable.IsInside(MatchElement: pElement): boolean; virtual;*

Is the specified element (contents) in this structure?

Overrides tADT.IsInside.

Rebuild *procedure tHashTable.Rebuild; virtual;*

Rebuilds the hash table (eliminates deleted slots, etc).

Resize *procedure tHashTable.Resize(CapacityOfElements: word); virtual;*

Resizes the table (changes the maximum storage capacity).

tHugeArray EFDEF

EFDEF.tHugeArray = array[1..High(Word)] of byte;

tHyperText EFTEXT

EFTEXT.tHyperText = object(tText)

Fields and Methods:

EFELEM.tIdentityElement = object(tStaticElement)

This class defines a identifiable static element. The element is identifiable since it is associated to some word-sized, positive integer value - the identity. tIdentityElements are ordered according to their identity numbers.

For more inforamtion, see *tStaticElement*.

Fields and Methods:

public
constructor *Initialize...*
function *Data...* virtual;
function *Size...* virtual;
function *Identity...* virtual;
function *Compare...* virtual;
constructor *StreamLoad...*
procedure *StreamStore...* virtual;
private
fIdentity: word;

Fields

fIdentity

tIdentityElement.fIdentity: word;

The identity of this static element.

Methods

Compare

function tIdentityElement.Compare(What: pElement): shortint; virtual;

Compares elements and returns [1, 0, -1] as result.

Overrides tElement.Compare.

Data

function tIdentityElement.Data(Position: word): pointer; virtual;

Returns a pointer to contents at a position within [0,n].

Overrides tElement.Data.

Identity

function tIdentityElement.Identity: word; virtual;

Returns the identity value of this element.

Initialize *constructor tIdentityElement.Initialize(IdentityOfElement: word);*

Initializes a named element with the specified name.

Overrides tObject.Initialize.

Size *function tIdentityElement.Size: word; virtual;*

Returns the size of the contents of this element in bytes.

Overrides tElement.Size.

StreamLoad *constructor tIdentityElement.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

Overrides tStaticElement.StreamLoad.

Overrides tElement.StreamLoad.

Overrides tObject.StreamLoad.

StreamStore *procedure tIdentityElement.StreamStore(Stream: pStream); virtual;*

Stores an instance to a stream.

Overrides tStaticElement.StreamStore.

Overrides tElement.StreamStore.

Overrides tObject.StreamStore.

tIFile EFFILE

EFFILE.tIFile = object(tFile)

This class implements a read only *tFile*. If a file does not exist, a fatal error is always triggered.

Fields and Methods:

public
constructor *Initialize...*

Methods

Initialize *constructor tIFile.Initialize(Filename: string; SizeOfBuffer: word);*

Initializes a file stream, optionally with an buffer.

Overrides tFile.Initialize.

Overrides tBufferFilter.Initialize.

Overrides tFilter.Initialize.

Overrides tStream.Initialize.

Overrides tObject.Initialize.

tImage — EFCONDRV

EFCONDRV.tImage = object(tVirtualConsole)

This class defines an image. An image is a console that is entirely hidden: a memory buffer that acts precisely as a console. Images can be used to copy screen information and save it to the disk, etc.

Fields and Methods:

public
constructor *Initialize...*
constructor *StreamLoad...*
function *Address...* virtual;
private
fBuffer: *pAllocation*;

Fields

fBuffer *tImage.fBuffer: pAllocation;*

Methods

Address *function tImage.Address(X1, Y1: word): pointer; virtual;*

Returns a pointer to the pixel at (X,Y), or NIL.

Overrides tConsole.Address.

Initialize *constructor tImage.Initialize(Boundaries: pField);*

Initializes an empty image with the specified boundaries.

Overrides tVirtualConsole.Initialize.

Overrides tBoundedConsole.Initialize.

Overrides tConsole.Initialize.

Overrides tComponent.Initialize.

Overrides tMessage.Initialize.

Overrides tObject.Initialize.

StreamLoad *constructor tImage.StreamLoad(Stream: pStream);*

Constructs and loads an instance from a stream.

Overrides tVirtualConsole.StreamLoad.

Overrides tBoundedConsole.StreamLoad.

Overrides tConsole.StreamLoad.

Overrides tComponent.StreamLoad.

Overrides tStaticElement.StreamLoad.

Overrides tElement.StreamLoad.

Overrides tObject.StreamLoad.

tInsertionSortPlug EFSORT

EFSORT.tInsertionSortPlug = object(tSortPlug)

Insertion sort plug class derived from *tSortPlug*. This class provides the Insertion sort algorithm. Insertion sort has limited performance and is quite simple. Consider *tQuickSortPlug* if performance is important. Complexity: O(N^2).

Fields and Methods:

public
constructor *Initialize...*
procedure *Sort...* virtual;
constructor *StreamLoad...*

Methods

Initialize *constructor tInsertionSortPlug.Initialize(PlugManager: pPlugManager);*

Constructs a plug instance for the specified manager.

Overrides tPlug.Initialize.

Overrides tObject.Initialize.

Sort *procedure tInsertionSortPlug.Sort(Start, Stop: word; Order: tSortOrder); virtual;*

Sorts elements within the specified interval.

Overrides tSortPlug.Sort.

StreamLoad *constructor tInsertionSortPlug.StreamLoad(Stream: pStream);*

Constructs and loads an instance from a stream.

Overrides tSortPlug.StreamLoad.

Overrides tPlug.StreamLoad.

Overrides tStaticElement.StreamLoad.

Overrides tElement.StreamLoad.

Overrides tObject.StreamLoad.

tInstallMessage EFDEVICE

EFDEVICE.tInstallMessage = object(tMessage)

Fields and Methods:

tInteger EFMATH

EFMATH.tInteger = object(tMathObject)

This class defines a 32-bit integer number with the range [-2147483648, 2147483647].

Fields and Methods:

public
constructor *Initialize...*
constructor *Duplicate...*
procedure *SetValue...*
procedure *Add...* virtual;
procedure *Subtract...* virtual;
procedure *Multiply...* virtual;
procedure *Divide...* virtual;
constructor *StreamLoad...*
procedure *StreamStore...* virtual;
procedure *StreamWrite...* virtual;
function *Value...* virtual;
function *IsZero...* virtual;
private
fN: longint;

Fields

fN *tInteger.fN: longint;*

32-bit integer [-2147483648, 2147483647].

Methods

Add *procedure tInteger.Add(What: pMathObject); virtual;*

Adds an object to this object.

Overrides tMathObject.Add.

Divide *procedure tInteger.Divide(What: pMathObject); virtual;*

Divides this object by another object.

Overrides tMathObject.Divide.

Duplicate *constructor tInteger.Duplicate(What: pMathObject);*

Initializes an duplicated type-casted mathematical object.

Initialize *constructor tInteger.Initialize(N: tBaseReal);*

Initializes an instance with the specified content.

Overrides tObject.Initialize.

IsZero *function tInteger.IsZero: boolean; virtual;*

Is number equal to zero?

Overrides tMathObject.IsZero.

Multiply *procedure tInteger.Multiply(What: pMathObject); virtual;*

Multiplies this object with another object.

Overrides tMathObject.Multiply.

SetValue *procedure tInteger.SetValue(N: tBaseReal);*

Sets the number.

StreamLoad *constructor tInteger.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

Overrides tStaticElement.StreamLoad.

Overrides tElement.StreamLoad.

Overrides tObject.StreamLoad.

StreamStore *procedure tInteger.StreamStore(Stream: pStream); virtual;*

Stores an instance to a stream.

Overrides tStaticElement.StreamStore.

Overrides tElement.StreamStore.

Overrides tObject.StreamStore.

StreamWrite *procedure tInteger.StreamWrite(Stream: pStream); virtual;*

Writes this math object as formatted text, e.g. "2+2i".

Overrides tElement.StreamWrite.

Overrides tObject.StreamWrite.

Subtract *procedure tInteger.Subtract(What: pMathObject); virtual;*

Subtracts an object from this object.

Overrides tMathObject.Subtract.

Value *function tInteger.Value: tBaseReal; virtual;*

Returns the value of this real number.

tIterator EFADT

EFADT.tIterator = object(tObject)

This is a abstract iterator class. This class defines a generic iterator - that is, an iterator that can operate on any ADT - linear or ordered. However, only *tLinearIterator* with descendants is to be used with tLinarADT descendant ADTs.

Iterators are guaranteed to operate with high performance (e.g. if a linked list is used). An ADT must be assigned to the iterator when the iterator is initialized. The associated ADT must not be intercepted before that iterator is done. Also, linked structures may not modify the node order when an iterator is assigned (the current node may be damaged).

Fields and Methods:

public
fADT: *pADT*;

procedure *Assign*...
procedure *First*; virtual;
procedure *Last*; virtual;
function *Element*... virtual;
function *Content*... virtual;
function *Size*... virtual;
function *Position*... virtual;
procedure *Walk*... virtual;
procedure *WalkTo*... virtual;
procedure *WalkToUsing*... virtual;
procedure *WalkUsing*... virtual;
procedure *WalkForward*; virtual;
procedure *WalkBackward*; virtual;
function *IsAllocated*... virtual;
function *IsEnd*... virtual;
function *IsValid*... virtual;
function *IsCompatible*... virtual;

Descendants:

tLinearIterator
  EFARRAY.tSparceIterator
  EFLIST.tNodeIterator
EFTABLE.tBucketIterator
EFTREE.tTreeIterator

Fields

fADT *tIterator.fADT: pADT;*

Pointer to the associated ADT.

Methods

Assign *procedure tIterator.Assign(ADT: pADT);*

Assigns an ADT to this plug instance.

Content *function tIterator.Content: pointer; virtual;*

Returns a pointer to the content of the current element.

Element *function tIterator.Element: pElement; virtual;*

Returns a pointer to the current element instance.

First *procedure tIterator.First; virtual;*

Walks to the first element in this ADT.

IsAllocated *function tIterator.IsAllocated: boolean; virtual;*

Is this iterator assigned to an ADT?

IsCompatible *function tIterator.IsCompatible(Instance: pObject): boolean; virtual;*

Are ADT's compatible, that is has compatible elements.

Overrides tObject.IsCompatible.

IsEnd *function tIterator.IsEnd: boolean; virtual;*

Is iterator at boundaries (beginning or end)?

IsValid *function tIterator.IsValid(ADT: pADT): boolean; virtual;*

Is the specified ADT of a valid class?

Last *procedure tIterator.Last; virtual;*

Walks to the last element in this ADT.

Position *function tIterator.Position: word; virtual;*

Returns the current position inside the ADT.

Size *function tIterator.Size: word; virtual;*

Returns the size of the current element (in bytes).

Walk *procedure tIterator.Walk(Steps: integer); virtual;*

Walks some steps relative to the current element.

WalkBackward *procedure tIterator.WalkBackward; virtual;*

Walk backwards (to the predecessor element).

WalkForward *procedure tIterator.WalkForward; virtual;*

Walk forwards (to the successor element).

WalkTo *procedure tIterator.WalkTo(Where: word); virtual;*

Walks to a position inside the associated ADT.

WalkToUsing *procedure tIterator.WalkToUsing(Iterator: pIterator; Where: word); virtual;*

Use the specified iterator to walk somewhere.

WalkUsing *procedure tIterator.WalkUsing(Iterator: pIterator; Steps: integer); virtual;*

Use the specified iterator to walk some steps.

EFKEYBRD.tKeyboard = object(tObject)

Device class for the keyboard. This class implements low-level functions for keyboard access, including read and write to the keyboard buffer, conversions and status checking.

Fields and Methods:

public
procedure *Enable*;
procedure *Disable*;
procedure *SetSensitivity...*
procedure *SetDefaultSensitivity*;
procedure *SetClick...*
function *ReadKey...*
function *ReadScanCode...*
function *WriteKey...*
function *WriteScanCode...*
function *PeekKey...*
function *PeekScanCode...*
procedure *Clear*;
function *WaitForScanCode...*
function *WaitForKey...*
procedure *WaitForReturn*;
procedure *WaitForPress...*
function *KeyPressed...*
function *AnyKeyPressed...*
function *AltPressed...*
function *CtrlPressed...*
function *ShiftPressed...*
function *LeftShiftPressed...*
function *RightShiftPressed...*
function *ScanCodeCharacter...*
function *ScanCode...*
function *ScanCodeWithAlt...*
function *ScanCodeWithCtrl...*

Methods

AltPressed *function tKeyboard.AltPressed: boolean;*

Is ALT pressed?

AnyKeyPressed *function tKeyboard.AnyKeyPressed: boolean;*
Is any key in buffer, including ALT and CTRL?

Clear *procedure tKeyboard.Clear;*
Clear the keyboard buffer.

CtrlPressed *function tKeyboard.CtrlPressed: boolean;*
Is CTRL pressed?

Disable *procedure tKeyboard.Disable;*
Disable the keyboard device.

Enable *procedure tKeyboard.Enable;*
Enable the keyboard device.

KeyPressed *function tKeyboard.KeyPressed: boolean;*
Is a valid key in buffer?

LeftShiftPressed *function tKeyboard.LeftShiftPressed: boolean;*
Is left shift pressed?

PeekKey *function tKeyboard.PeekKey: char;*
Check what character is waiting in keyboard buffer.

PeekScanCode *function tKeyboard.PeekScanCode: word;*
Check what scancode is waiting in keyboard buffer.

ReadKey *function tKeyboard.ReadKey: char;*
Read and wait for a character in keyboard buffer.

ReadScanCode *function tKeyboard.ReadScanCode: word;*
Read and wait for a scancode in keyboard buffer.

RightShiftPressed *function tKeyboard.RightShiftPressed: boolean;*
Is right shift pressed?

ScanCode *function tKeyboard.ScanCode(Character: char): word;*
Converts the specified character to a scancode.

ScanCodeCharacter *function tKeyboard.ScanCodeCharacter(ThisScanCode: word): char;*
Convers the specified scancode to a character.

ScanCodeWithAlt *function tKeyboard.ScanCodeWithAlt(Character: char): word;*

Converts the specified character to a scancode with ALT.

ScanCodeWithCtrl *function tKeyboard.ScanCodeWithCtrl(Character: char): word;*

Converts the specified character to a scancode with CTRL.

SetClick *procedure tKeyboard.SetClick(Click: boolean);*

Turn click sound on / off.

SetDefaultSensitivity *procedure tKeyboard.SetDefaultSensitivity;*

Set default sensitivity (reset).

SetSensitivity *procedure tKeyboard.SetSensitivity(RepeatDelay, Rate: byte);*

Set sensitivity (repeat speed).

ShiftPressed *function tKeyboard.ShiftPressed: boolean;*

Is shift pressed?

WaitForKey *function tKeyboard.WaitForKey(ThisKey: char; Timeout: word): boolean;*

Wait for a key the specified number of milliseconds.

WaitForPress *procedure tKeyboard.WaitForPress(Timeout: word);*

Wait for any key the specified number of milliseconds.

WaitForReturn *procedure tKeyboard.WaitForReturn;*

Wait for return.

WaitForScanCode *function tKeyboard.WaitForScanCode(ThisScanCode: word; Timeout: word): boolean;*

Wait for scan code the specified number of milliseconds.

WriteKey *function tKeyboard.WriteKey(Key: char): boolean;*

Write a character to the keyboard buffer.

WriteScanCode *function tKeyboard.WriteScanCode(ThisScanCode: word): boolean;*

Write a scancode to the keyboard buffer.

EFKEYBRD.tKeyEvent = object(tMessage)

Keyboard event: this event carries an arbitrary keypress detected by the keyboard device (see tKeyboardDevice).

Fields and Methods:

public
function *Key...* virtual;
function *ScanCode...* virtual;
function *AltPressed...*
function *CtrlPressed...*
function *ShiftPressed...*
function *IsControlKey...* virtual;
private
fScanCode: word;

Fields

fScanCode *tKeyEvent.fScanCode: word;*

Methods

AltPressed *function tKeyEvent.AltPressed: boolean;*

Was Alt pressed together with the key?

CtrlPressed *function tKeyEvent.CtrlPressed: boolean;*

Was Ctrl pressed together with the key?

IsControlKey *function tKeyEvent.IsControlKey: boolean; virtual;*

Key *function tKeyEvent.Key: char; virtual;*

ScanCode *function tKeyEvent.ScanCode: word; virtual;*

ShiftPressed *function tKeyEvent.ShiftPressed: boolean;*

Was shift pressed together with the key?

tKeyPlug EFADT

EFADT.tKeyPlug = object(tADTPlug)

Key plug class: this class provides a mechanism of assigning search keys to elements. Optionally, tKeyPlug enables you to override the *tElement.Compare* method. The latter makes it possible to force ADT′s to change compare criterias without extending tElement classes.

Ordered ADT use tKeyPlug to create search keys. This is the base class of all tKeyPlug. It considers the entire element as the search key, that is, a key matches an element if, and only if, the key is equal to the content of the element.

Fields and Methods:

public
constructor *Initialize...*
function *CompareElements...* virtual;
function *CompareKey...* virtual;
function *CreateKey...* virtual;
constructor *StreamLoad...*
function *TypeOfPlug...* virtual;

Descendants:

tBoundedKeyPlug
tReversedKeyPlug

Methods

CompareElements *function tKeyPlug.CompareElements(A, B: pElement): shortint; virtual;*

Compares element A with element B and returns [-1, 0, 1].

CompareKey *function tKeyPlug.CompareKey(var A; B: pElement): shortint; virtual;*

Compares key A with element B and returns [-1, 0, 1].

CreateKey *function tKeyPlug.CreateKey(A: pElement): pAllocation; virtual;*

Creates a memory allocation with the key.

Initialize *constructor tKeyPlug.Initialize(PlugManager: pPlugManager);*

Initializes a standard key plug (similar to ADT compare).

Overrides tPlug.Initialize.

Overrides tObject.Initialize.

StreamLoad *constructor tKeyPlug.StreamLoad(Stream: pStream);*

Constructs and loads an instance from a stream.

Overrides tPlug.StreamLoad.

Overrides tStaticElement.StreamLoad.

Overrides tElement.StreamLoad.

Overrides tObject.StreamLoad.

TypeOfPlug *function tKeyPlug.TypeOfPlug: string; virtual;*

The classification of this plug, e.g. "tSortPlug".

Overrides tADTPlug.TypeOfPlug.

Overrides tPlug.TypeOfPlug.

tLinearADT EFADT

EFADT.tLinearADT = object(tADT)

This class defines the properties of a linear data structure, that is an ADT with elements that can be accessed using index values (like an array).

Common behavior for tLinearADT descendants:

(1) Elements are referenced with index values [1, n].

(2) An ADT is classified as sortable if the content can be rearranged arbitrary, ie. if elements can be moved.

In addition to abstract methods in *tADT*, a linear ADT must override the following abstract methods: Delete, *Erase*, *Insert*, *Get* and *SetElement*. If the descendant permit NIL element instances, then *Update* and *Swap* must also be overridden (see for example tArray).

Fields and Methods:

public
constructor *Initialize*;
procedure *Retrieve...* virtual;
procedure *Update...* virtual;
procedure *Erase...* virtual;
procedure *Insert...* virtual;

function *Get...* virtual;
procedure *SetElement...* virtual;
function *Search...* virtual;
function *Locate...* virtual;
procedure *Swap...* virtual;
procedure *Sort...* virtual;
procedure *AdjustInterval...*
function *PackageSize...* virtual;
function *High...* virtual;
function *Low...* virtual;
function *IsValid...* virtual;
function *IsUsed...* virtual;
function *IsSorted...* virtual;
function *IsCompatible...* virtual;

Descendants:

EFARRAY.tArray
  EFARRAY.tBoundedArray
    EFARRAY.tVirtualArray
EFLIST.tList
  EFLIST.tCursorList
  EFLIST.tReversedList
    EFLIST.tAdjustedList
    EFLIST.tOrderedList

Methods

AdjustInterval *procedure tLinearADT.AdjustInterval(var Start, Stop: word);*

Adjusts index interval to valid values for this data type.

Erase *procedure tLinearADT.Erase(Index: word); virtual;*

Erases an element from the data structure.

Get *function tLinearADT.Get(Index: word): pElement; virtual;*

Gets the specified element instance.

High *function tLinearADT.High: word; virtual;*

Returns index for the highest used element.

Initialize *constructor tLinearADT.Initialize;*

Initializes an ADT and assigns the default sort plug.

Overrides tADT.Initialize.

Overrides tObject.Initialize.

Insert *procedure tLinearADT.Insert(var Data; Index: word); virtual;*

Inserts an element after the specified index.

IsCompatible *function tLinearADT.IsCompatible(Instance: pObject): boolean; virtual;*

Are ADT′s compatible, that is has compatible elements.

Overrides tADT.IsCompatible.

Overrides tObject.IsCompatible.

IsSorted *function tLinearADT.IsSorted(Order: tSortOrder): boolean; virtual;*

Is this ADT sorted in specified order?

IsUsed *function tLinearADT.IsUsed(Index: word): boolean; virtual;*

Is the specified element used (not a NIL instance)?

IsValid *function tLinearADT.IsValid(Index: word): boolean; virtual;*

Is the specified element index valid?

Locate *function tLinearADT.Locate(MatchElement: pElement): word; virtual;*

Searchs for element and returns found element or zero.

Low *function tLinearADT.Low: word; virtual;*

Returns index for the lowest used element.

PackageSize *function tLinearADT.PackageSize: word; virtual;*

Returns the size of element packaging (container).

Overrides tADT.PackageSize.

Retrieve *procedure tLinearADT.Retrieve(var Data; Index: word); virtual;*

Retrieves the contents of the specified element.

Search *function tLinearADT.Search(var Data; Size: word): word; virtual;*

Searchs for data and returns found element or zero.

SetElement *procedure tLinearADT.SetElement(Element: pElement; Index: word); virtual;*

Sets the element instance at some index.

Sort *procedure tLinearADT.Sort(Order: tSortOrder); virtual;*

Sorts elements into the specified order using *tSortPlug*.

Swap *procedure tLinearADT.Swap(A, B: word); virtual;*

Swaps places of two elements.

Update *procedure tLinearADT.Update(var Data; Index: word); virtual;*

Updates an element (specified by an index value).

tLinearIterator EFADT

EFADT.tLinearIterator = object(tIterator)

This is a linear iterator class. An iterator walks through the elements inside a linear ADT in arbitrary direction. All other linear iterator classes are derived from this class.

This iterator starts at the first element (1) and ends at the last element (as returned by Elements). Arrays must use tSparceIterator since there elements does not have to be used. Thus, arrays Elements method returns the number of used elements, while the Capacity method returns the size of the array. See tSparceIterator.

The linear ADT is an extension of *tIterator*. It is a concrete class, but is subclassed into the performance enhanced iterator tNodeIterator. For more information about iterators, see *tIterator*.

Fields and Methods:

public
constructor *Initialize...*
procedure *First*; virtual;
procedure *Last*; virtual;
function *Element...* virtual;
function *Position...* virtual;
procedure *Walk...* virtual;
procedure *WalkTo...* virtual;

procedure *WalkToUsing...* virtual;
procedure *WalkUsing...* virtual;
procedure *WalkForward*; virtual;
procedure *WalkBackward*; virtual;
constructor *StreamLoad...*
procedure *StreamStore...* virtual;
function *IsEnd...* virtual;
function *IsValid...* virtual;
function *IsEqual...* virtual;
function *IsCompatible...* virtual;
public
fCursor: word;

Descendants:

EFARRAY.tSparceIterator
EFLIST.tNodeIterator

Fields

fCursor *tLinearIterator.fCursor: word;*

Cursor index (the current element).

Methods

Element *function tLinearIterator.Element: pElement; virtual;*

Returns a pointer to the current element instance.

Overrides tIterator.Element.

First *procedure tLinearIterator.First; virtual;*

Walks to the first element in this ADT.

Overrides tIterator.First.

Initialize *constructor tLinearIterator.Initialize(ADT: pLinearADT);*

Initializes an iterator for the specified ADT.

Overrides tObject.Initialize.

IsCompatible *function tLinearIterator.IsCompatible(Instance: pObject): boolean; virtual;*

Are ADT′s compatible, that is has compatible elements.

Overrides tIterator.IsCompatible.

Overrides tObject.IsCompatible.

IsEnd *function tLinearIterator.IsEnd: boolean; virtual;*

Is iterator at boundaries (beginning or end)?

Overrides tIterator.IsEnd.

IsEqual *function tLinearIterator.IsEqual(Instance: pObject): boolean; virtual;*

Are contents of these instances equal (same cursor)?

Overrides tObject.IsEqual.

IsValid *function tLinearIterator.IsValid(ADT: pADT): boolean; virtual;*

Is the specified ADT of a valid class?

Overrides tIterator.IsValid.

Last *procedure tLinearIterator.Last; virtual;*

Walks to the last element in this ADT.

Overrides tIterator.Last.

Position *function tLinearIterator.Position: word; virtual;*

Returns the current position inside the ADT.

Overrides tIterator.Position.

StreamLoad *constructor tLinearIterator.StreamLoad(Stream: pStream);*

Constructs and loads an instance from a stream.

Overrides tObject.StreamLoad.

StreamStore *procedure tLinearIterator.StreamStore(Stream: pStream); virtual;*

Stores an instance to a stream.

Overrides tObject.StreamStore.

Walk *procedure tLinearIterator.Walk(Steps: integer); virtual;*

Walks some steps relative to the current element.

Overrides tIterator.Walk.

WalkBackward *procedure tLinearIterator.WalkBackward; virtual;*

Walk backwards (to the predecessor element).

Overrides tIterator.WalkBackward.

WalkForward *procedure tLinearIterator.WalkForward; virtual;*

Walk forwards (to the successor element).

Overrides tIterator.WalkForward.

WalkTo *procedure tLinearIterator.WalkTo(Where: word); virtual;*

Walks to a position inside the associated ADT.

Overrides tIterator.WalkTo.

WalkToUsing *procedure tLinearIterator.WalkToUsing(Iterator: pIterator; Where: word); virtual;*

Use the specified iterator to walk somewhere.

Overrides tIterator.WalkToUsing.

WalkUsing *procedure tLinearIterator.WalkUsing(Iterator: pIterator; Steps: integer); virtual;*

Use the specified iterator to walk some steps.

Overrides tIterator.WalkUsing.

tLineStyle EFCONDRV

EFCONDRV.tLineStyle = object(tNamedElement)

This class defines a line style. It is abstract. A line style is a way of drawing a line in an arbitrary console. Descendants must at least override the *DrawLine* method. See tNamedElement for more information.

Fields and Methods:

public
constructor *Initialize...*
procedure *DrawLine...* virtual;
procedure *DrawFrame...* virtual;
procedure *DrawBranch...* virtual;
constructor *StreamLoad...*

Methods

DrawBranch *procedure tLineStyle.DrawBranch(Console: pConsole; X1, Y1, X2, Y2: word; Color: pColor); virtual;*

Draws a branch inside a frame.

DrawFrame *procedure tLineStyle.DrawFrame(Console: pConsole; X1, Y1, X2, Y2: word; Color: pColor); virtual;*

Draws a frame with the specified boundaries.

DrawLine *procedure tLineStyle.DrawLine(Console: pConsole; X1, Y1, X2, Y2: word; Color: pColor); virtual;*

Draws a line between two coordinates.

Initialize *constructor tLineStyle.Initialize(NameOfFont: string);*

Initializes a font with the specified name.

StreamLoad *constructor tLineStyle.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

tLinkage EFLIST

EFLIST.tLinkage = object(tCarrier)

Abstract class that imlements a container for elements (*tElement* descendants). tLinkage inherits methods for attachment and detachment in a doubly linked data structure.

Fields and Methods:

public
constructor *Initialize...*
function *Element...* virtual;
function *IsValid...* virtual;

Descendants:

EFTREE.tTreeLinkage
  EFTREE.tAVLLinkage

Methods

Element *function tLinkage.Element: pElement; virtual;*

Returns the element inside this linkage (or triggers error).

Initialize *constructor tLinkage.Initialize(HoldElement: pElement; Attach: pLinkage);*

Initializes container: attachs it before another.

Overrides tCarrier.Initialize.

Overrides tContainer.Initialize.

Overrides tObject.Initialize.

IsValid *function tLinkage.IsValid(Instance: pObject): boolean; virtual;*

Is the specified instance valid for this carrier?

Overrides tCarrier.IsValid.

tList EFLIST

EFLIST.tList = object(tLinearADT)

This class defines a general doubly linked list. Linked lists store elements in container instances (*tLinkage*). Containers are linked to each other, forming a doubly linked data structure:

—> | | —> | | —> HEAD [1] [n] HEAD <— | | <— | | <—

Linked lists have a head node that is linked from both the first and the last element. If a linked list is empty, the head node links only to itself.

A list is a data structure of choice when the number of elements in an ADT cannot be beounded. Unlike arrays, a list holds any number of elements. Links have virtual indexing. Elements must be examined one by one in sequence. For this reasion, the ammount of time required to access an element in a list depends upon the position the element holds in the list; accessing values near the head node of the list is performed more rapidly than accessing elements near the end.

Fields and Methods:

public
constructor *Initialize...*
constructor *Resemble...*
constructor *Duplicate...*
destructor *Intercept*; virtual;
procedure *Clear*; virtual;
procedure *Put...* virtual;
function *Get...* virtual;

procedure *SetElement...* virtual;
procedure *Update...* virtual;
procedure *Erase...* virtual;
procedure *EraseFirst*; virtual;
procedure *EraseLast*; virtual;
procedure *Insert...* virtual;
procedure *Reverse*; virtual;
procedure *Circulate*; virtual;
function *CreateIterator...* virtual;
procedure *PutNode...* virtual;
procedure *Touch...* virtual;
function *FirstNode...* virtual;
function *LastNode...* virtual;
function *IndexedNode...* virtual;
function *NodeIndex...* virtual;
constructor *StreamLoad...*
function *Elements...* virtual;
function *PackageSize...* virtual;
function *IsAllocated...* virtual;
function *IsEmpty...* virtual;
function *IsIntact...* virtual;
function *IsAttachNode...* virtual;
function *IsValidNode...* virtual;
private
fHead: *pLinkage*;

Descendants:

tCursorList
tReversedList
  tAdjustedList
  tOrderedList

Fields

fHead *tList.fHead: pLinkage;*

Dummy head node (also tail node).

Methods

Circulate *procedure tList.Circulate; virtual;*

Circulates elements, ie. moves last element to the front.

Clear *procedure tList.Clear; virtual;*

Clears all elements inside data type.

Overrides tADT.Clear.

CreateIterator *function tList.CreateIterator: pIterator; virtual;*

Creates an instance of the iterator class.

Overrides tADT.CreateIterator.

Duplicate *constructor tList.Duplicate(ADT: pADT);*

Initializes a duplicate ADT instance (with equal elements).

Elements *function tList.Elements: word; virtual;*

Returns the number of elements that currently are used.

Overrides tADT.Elements.

Erase *procedure tList.Erase(Index: word); virtual;*

Erases an element from the data structure.

Overrides tLinearADT.Erase.

EraseFirst *procedure tList.EraseFirst; virtual;*

Erases the first element in structure.

EraseLast *procedure tList.EraseLast; virtual;*

Erases the last element in structure.

FirstNode *function tList.FirstNode: pLinkage; virtual;*

Returns the first node in the structure (or NIL if empty).

Get *function tList.Get(Index: word): pElement; virtual;*

Gets the specified element instance.

Overrides tLinearADT.Get.

IndexedNode *function tList.IndexedNode(Index: word): pLinkage; virtual;*

Index ==> node; returns node with specified index.

Initialize *constructor tList.Initialize(SizeOfElements: word);*

Initializes an instance of this ADT class.

Overrides tLinearADT.Initialize.

Overrides tADT.Initialize.

Overrides tObject.Initialize.

Insert *procedure tList.Insert(var Data; Index: word); virtual;*

Inserts an element after the specified index.

Overrides tLinearADT.Insert.

Intercept *destructor tList.Intercept; virtual;*

Intercepts and destructs an instance of this type.

Overrides tADT.Intercept.

Overrides tObject.Intercept.

IsAllocated *function tList.IsAllocated: boolean; virtual;*

Is this ADT allocated, that is ready for use?

Overrides tADT.IsAllocated.

IsAttachNode *function tList.IsAttachNode(AttachNode, Node: pLinkage): boolean; virtual;*

Should the second node be inserted after the first node?

IsEmpty *function tList.IsEmpty: boolean; virtual;*

Is this ADT type empty, that is does not have any elements?

Overrides tADT.IsEmpty.

IsIntact *function tList.IsIntact: boolean; virtual;*

Is this data structure intact: are links preserved?

IsValidNode *function tList.IsValidNode(Node: pLinkage): boolean; virtual;*

Is node valid - assigned, intact and not the head node?

LastNode *function tList.LastNode: pLinkage; virtual;*

Returns the last node in the structure (or NIL if empty).

NodeIndex *function tList.NodeIndex(Node: pLinkage): word; virtual;*

Node ==> index; returns index of specified node.

PackageSize *function tList.PackageSize: word; virtual;*

Returns the size of element packaging (container).

Overrides tLinearADT.PackageSize.

Overrides tADT.PackageSize.

Put *procedure tList.Put(Element: pElement); virtual;*

Stores a new element (tElement instance) to this ADT.

Overrides tADT.Put.

PutNode *procedure tList.PutNode(Node: pLinkage); virtual;*

Stores a new node (container) in the list.

Resemble *constructor tList.Resemble(ADT: pADT);*

Initializes an ADT that resembles the specified ADT.

Reverse *procedure tList.Reverse; virtual;*

Reverses elements (moves them into reversed order).

SetElement *procedure tList.SetElement(Element: pElement; Index: word); virtual;*

Sets the element instance at some index.

Overrides tLinearADT.SetElement.

StreamLoad *constructor tList.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

Overrides tADT.StreamLoad.

Overrides tObject.StreamLoad.

Touch *procedure tList.Touch(Node: pLinkage; Index: word); virtual;*

Touchs a node, i.e. notifies that it was used (optional).

Update *procedure tList.Update(var Data; Index: word); virtual;*

Updates an element (specified by an index value).

Overrides tLinearADT.Update.

tLocalElement EFSYSTEM

EFSYSTEM.tLocalElement = object(tElement)

Fields and Methods:

fType: longint;
fAddress: pointer;

Fields

fAddress *tLocalElement.fAddress: pointer;*

Address to some local, far-declared constant or variable.

fType *tLocalElement.fType: longint;*

Integer mapping for the type classification.

tLocalMessage EFMSG

EFMSG.tLocalMessage = object(tMessage)

Local message: this event is associated to some local (far) procedure in your main program. This procedure is automatically called whenever the message is processed.

Fields and Methods:

public
constructor *Initialize...*
procedure *Action*; virtual;
private
fLocalProcedure: *tLocalProcedure*;

Fields

fLocalProcedure *tLocalMessage.fLocalProcedure: tLocalProcedure;*

Methods

Action *procedure tLocalMessage.Action; virtual;*

Does something when this message is processed.

Overrides tMessage.Action.

Initialize *constructor tLocalMessage.Initialize(MessageReceiver, MessageSender: pComponent; LocalProcedure: tLocalProcedure);*

Initializes a message to the specified receiver and sender.

Overrides tMessage.Initialize.

Overrides tObject.Initialize.

tLocalProcedure EFMSG

EFMSG.tLocalProcedure = procedure(Message: pMessage);

This type definition enables *tLocalMessage* to make use of an arbitrary local (far) procedure instead of the normal action method. The argument is calling message.

tManager EFCMAN

EFCMAN.tManager = object(tComponent)

Abstract manager class. This class defines a component that in addition to all properties defined in *tComponent* owns child components. It manages more components.

Managers must:

(1) Pass all messages to the right receiver among the child components, or ignore or reject an invalid message. Messages can be scheduled for later transmittance and placed in a message queue - they are posted. This behavior is implemeneted in *tCommunicator.Post*. *tManager* forces any posted message to be sent directly to its receiever if there is no manager for the manager.

(2) Know which of the components are focused. This is done by organizing the elements i Z-order (according to their priority order - normally the order they are shown on the screen).

(3) Know how to kill (intercept) an arbitrary component. Before the component is killed, it must be finished with the current task.

Fields and Methods:

public
constructor *Initialize*;
destructor *Intercept*; virtual;
procedure *Waiting*; virtual;
procedure *Update*; virtual;
procedure *Register...* virtual;
procedure *Focus...* virtual;

procedure *Kill...* virtual;
procedure *Handle...* virtual;
constructor *StreamLoad...*
procedure *StreamStore...* virtual;
function *FocusedComponent...* virtual;
function *IsVisual...* virtual;
function *IsRegistered...* virtual;
function *IsResponsible...* virtual;
private
fComponents: *pLinearADT*;
fPhase: *tProcessPhase*;

Descendants:

EFAPP.tApplication
tCommunicator

Fields

fComponents *tManager.fComponents: pLinearADT;*

The managed components ordered in Z-order.

fPhase *tManager.fPhase: tProcessPhase;*

The phase of the component processing.

Methods

Focus *procedure tManager.Focus(Component: pComponent); virtual;*

Attempts to activate the specified component.

FocusedComponent *function tManager.FocusedComponent: pComponent; virtual;*

Returns the focused component (or NIL if none).

Handle *procedure tManager.Handle(Message: pMessage); virtual;*

Handles a single (non-chained) message.

Overrides tComponent.Handle.

Initialize *constructor tManager.Initialize;*

Initializes this manager without any children.

Overrides tComponent.Initialize.

Overrides tMessage.Initialize.

Overrides tObject.Initialize.

Intercept *destructor tManager.Intercept; virtual;*

Intercepts this manager and all its owned components.

Overrides tElement.Intercept.

Overrides tObject.Intercept.

IsRegistered *function tManager.IsRegistered(Component: pComponent): boolean; virtual;*

Is the specified component registered?

IsResponsible *function tManager.IsResponsible(Component: pComponent; var Result: pComponent): boolean; virtual;*

Is this component responsible for the specified component?

Overrides tComponent.IsResponsible.

IsVisual *function tManager.IsVisual: boolean; virtual;*

Is this component visual, that is, uses a console device?

Overrides tComponent.IsVisual.

Kill *procedure tManager.Kill(Component: pComponent); virtual;*

Kills a component, ie. intercepts the component instance.

Register *procedure tManager.Register(Component: pComponent); virtual;*

Registers a new component and makes it focused.

StreamLoad *constructor tManager.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

Overrides tComponent.StreamLoad.

Overrides tStaticElement.StreamLoad.

Overrides tElement.StreamLoad.

Overrides tObject.StreamLoad.

StreamStore *procedure tManager.StreamStore(Stream: pStream); virtual;*

Stores an instance to a stream.

Overrides tComponent.StreamStore.

Overrides tStaticElement.StreamStore.

Overrides tElement.StreamStore.

Overrides tObject.StreamStore.

Update *procedure tManager.Update; virtual;*

Updates the component after its status has changed.

Overrides tComponent.Update.

Waiting *procedure tManager.Waiting; virtual;*

Permits this component to do something when it is waiting.

Overrides tComponent.Waiting.

tMapping EFSYSTEM

EFSYSTEM.tMapping = object(tNamedElement)

This class associates or maps some text to an arbitrary integer number with longint range. A mapping can be used to provide aliases for certain constants.

Fields and Methods:

public
private
fIdentity: longint;

Fields

fIdentity *tMapping.fIdentity: longint;*

Integer mapping for the specified name.

tMatchFilter EFSEARCH

EFSEARCH.tMatchFilter = object(tFilter)

This class defines a filter with an automatic search mechanism that compares your data in the background. tMatchFilter does not provide any advanced pattern matching mechanism. See *tFilter* for more information.

This filter searches data when it is read or written, and allows you to check for the last match at any time during the processing. It also counts the total number of matches since Reset was last called. See *Occurrences* and *LastMatch*.

Fields and Methods:

public
constructor *Initialize...*
destructor *Intercept*; virtual;
procedure *Read...* virtual;
procedure *Write...* virtual;
procedure *PutByte...* virtual;
function *GetByte...* virtual;
function *FilterByte...* virtual;
procedure *Seek...* virtual;
procedure *Found...* virtual;
function *Occurrences...* virtual;
function *LastMatch...* virtual;
function *IsFound...* virtual;
function *IsAllocated...* virtual;
private
fLastMatch: longint;
fOccurrences: longint;
fPattern: *pElement*;
fPatternIndex: word;

Descendants:

tSearchFilter
  tDelayedSearchFilter
    tReplaceFilter

Fields

fLastMatch *tMatchFilter.fLastMatch: longint;*

Position of the last match (first character).

fOccurrences *tMatchFilter.fOccurrences: longint;*

The total number of occurrences.

fPattern *tMatchFilter.fPattern: pElement;*

The pattern to search for (some text).

fPatternIndex *tMatchFilter.fPatternIndex: word;*

Number of matched characters in pattern.

Methods

FilterByte *function tMatchFilter.FilterByte(Data: byte): byte; virtual;*

Checks if the specified byte is a match.

Found *procedure tMatchFilter.Found(Where: longint); virtual;*

Handles a found match (sets data members).

GetByte *function tMatchFilter.GetByte: byte; virtual;*

Gets a byte from this stream.

Overrides tStream.GetByte.

Initialize *constructor tMatchFilter.Initialize(var Data; Length: word; Stream: pStream);*

Initializes a search filter for the specified data.

Overrides tFilter.Initialize.

Overrides tStream.Initialize.

Overrides tObject.Initialize.

Intercept *destructor tMatchFilter.Intercept; virtual;*

Intercepts and destructs an instance of this type.

Overrides tFilter.Intercept.

Overrides tStream.Intercept.

Overrides tObject.Intercept.

IsAllocated *function tMatchFilter.IsAllocated: boolean; virtual;*

Is the stream allocated, ie. ready to be used in some way?

Overrides tFilter.IsAllocated.

Overrides tStream.IsAllocated.

IsFound *function tMatchFilter.IsFound: boolean; virtual;*

Has the searched data been found at least once?

LastMatch *function tMatchFilter.LastMatch: longint; virtual;*

Returns the position of the last match.

Occurrences *function tMatchFilter.Occurrences: longint; virtual;*

Returns the total number of occurrences.

PutByte *procedure tMatchFilter.PutByte(Data: byte); virtual;*

Puts a byte into this stream.

Overrides tStream.PutByte.

Read *procedure tMatchFilter.Read(var Data; Count: word); virtual;*

Reads specified amount of data from this stream.

Overrides tFilter.Read.

Overrides tStream.Read.

Seek *procedure tMatchFilter.Seek(Where: longint); virtual;*

Seeks, ie. moves to, a position in the stream [0 .. n].

Overrides tFilter.Seek.

Overrides tStream.Seek.

Write *procedure tMatchFilter.Write(var Data; Count: word); virtual;*

Writes specified data to this stream.

Overrides tFilter.Write.

Overrides tStream.Write.

tMathFunction EFMEXP

EFMEXP.tMathFunction = object(tMathObject)

Fields and Methods:

public
constructor *Initialize*;
constructor *StreamLoad...*
private
fVariables: pOrderedADT;

EFMATH.tMathFunction = object(tObject)

Mathematical function in one variablet (X). This class used by *tSolver* for external functions (used by Newton Raphson′s method and Simpson′s formula).

Fields and Methods:

public
constructor *Initialize*;
function *F*...
constructor *StreamLoad*...

Fields

fVariables *tMathFunction.fVariables: pOrderedADT;*

Pointer to the variables.

Methods

F *function tMathFunction.F(X: tBaseReal): tBaseReal;*

Returns the function value F(X) where X is given.

Initialize *constructor tMathFunction.Initialize;*

Initializes an instance of this class.

Overrides tObject.Initialize.

Initialize *constructor tMathFunction.Initialize;*

Initializes an instance of this class.

Overrides tObject.Initialize.

StreamLoad *constructor tMathFunction.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

Overrides tStaticElement.StreamLoad.

Overrides tElement.StreamLoad.

Overrides tObject.StreamLoad.

StreamLoad *constructor tMathFunction.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

Overrides tObject.StreamLoad.

tMathObject EFMATH

EFMATH.tMathObject = object(tStaticElement)

This class defines a generic mathematical object, for example integers, rational numbers, real numbers, complex numbers, vectors and matrixes. All mathematical objects share a common interface, though not all objects may support all the operations that are defined here.

Fields and Methods:

public
procedure *Add...* virtual;
procedure *Subtract...* virtual;
procedure *Multiply...* virtual;
procedure *Divide...* virtual;
function *IsZero...* virtual;

Descendants:

tComplex
tInteger
tMatrix
  tPolynomial
tRational
tReal
tVector
EFMEXP.tExpression
EFMEXP.tMathFunction

Methods

Add *procedure tMathObject.Add(What: pMathObject); virtual;*

Adds an object to this object.

Divide *procedure tMathObject.Divide(What: pMathObject); virtual;*

Divides this object by another object.

IsZero *function tMathObject.IsZero: boolean; virtual;*

Is number equal to zero?

Multiply *procedure tMathObject.Multiply(What: pMathObject); virtual;*

Multiplies this object with another object.

Subtract *procedure tMathObject.Subtract(What: pMathObject); virtual;*

Subtracts an object from this object.

tMatrix EFMATH

EFMATH.tMatrix = object(tMathObject)

This class defines a arbitrary large 2-dimensional matrix and all arithmetical operations that are defined for these mathematical objects. Inversion, Gauss′ elimination and some statistical functions are included.

Fields and Methods:

public
constructor *Initialize...*
constructor *Duplicate...*
constructor *InitializeUnitary...*
destructor *Intercept*; virtual;
procedure *SetElement...*
function *Element...*
procedure *Add...* virtual;
procedure *Subtract...* virtual;
procedure *Multiply...* virtual;
procedure *Divide...* virtual;
procedure *MultiplyScalar...*
procedure *MultiplyRow...*
procedure *Transpose*;
function *Invert...*
function *Determinant...*
function *Gauss...*
function *Triangulate...*
function *Eliminate...*
function *BackSubstitute...*
procedure *Normalize...*
procedure *SwapRow...*
function *Average...*
function *Deviation...*
function *CreateMathObject...* virtual;
constructor *StreamLoad...*
procedure *StreamStore...* virtual;
procedure *StreamWrite...* virtual;
function *IsAllocated...* virtual;

function *IsValid...*
function *IsZero...* virtual;
function *IsSquare...* virtual;
function *IsResultMatrix...*
function *IsCompatible...* virtual;
function *Rows...*
function *Columns...*
private
fMatrix: *pArray*;
fRows: word;
function *Reference...*

Descendants:

tPolynomial

Fields

fMatrix *tMatrix.fMatrix: pArray;*

Matrix element data base.

fRows *tMatrix.fRows: word;*

Number of rows in matrix.

Methods

Add *procedure tMatrix.Add(What: pMathObject); virtual;*

Adds an object to this object.

Overrides tMathObject.Add.

Average *function tMatrix.Average: tBaseReal;*

Calculates the average value of all elements.

BackSubstitute *function tMatrix.BackSubstitute(Result: pMatrix): boolean;*

Back-substitutes a triangulated matrix (if possible).

Columns *function tMatrix.Columns: word;*

Returns the number of columns in this matrix.

CreateMathObject *function tMatrix.CreateMathObject: pMathObject; virtual;*

Creates a math object for this matrix.

Determinant *function tMatrix.Determinant: tBaseReal;*

Calculates the determinant for this matrix.

Deviation *function tMatrix.Deviation: tBaseReal;*

Returns the standard deviation for this matrix.

Divide *procedure tMatrix.Divide(What: pMathObject); virtual;*

Divides this object by another object.

Overrides tMathObject.Divide.

Duplicate *constructor tMatrix.Duplicate(What: pMathObject);*

Initializes an duplicated type-casted mathematical object.

Element *function tMatrix.Element(Row, Column: word): tBaseReal;*

Retrieves the value of an element in the matrix.

Eliminate *function tMatrix.Eliminate(Row, Column, UsingRow: word; Result: pMatrix): boolean;*

Elimates the specified element with another row.

Gauss *function tMatrix.Gauss(Result: pMatrix): boolean;*

Solves the equation system Matrix = X, returns status.

Initialize *constructor tMatrix.Initialize(nRows, nColumns: word);*

Initializes and constructs an instance of this type.

Overrides tObject.Initialize.

InitializeUnitary *constructor tMatrix.InitializeUnitary(nRows: word);*

Initializes an instance as the unitary matrix (Rows*Rows).

Intercept *destructor tMatrix.Intercept; virtual;*

Intercepts and destructs an instance of this type. #E Element handling methods

Overrides tElement.Intercept.

Overrides tObject.Intercept.

Invert *function tMatrix.Invert: boolean;*

Inverts this matrix.

IsAllocated *function tMatrix.IsAllocated: boolean; virtual;*

Is a matrix allocated for this instance?

Overrides tElement.IsAllocated.

IsCompatible *function tMatrix.IsCompatible(What: pObject): boolean; virtual;*

Has this matrix the same size as the specified matrix?

Overrides tStaticElement.IsCompatible.

Overrides tElement.IsCompatible.

Overrides tObject.IsCompatible.

IsResultMatrix *function tMatrix.IsResultMatrix(Result: pMatrix): boolean;*

Returns TRUE if the specified matrix is valid for results.

IsSquare *function tMatrix.IsSquare: boolean; virtual;*

Is the number of rows equal to the number of columns?

IsValid *function tMatrix.IsValid(Row, Column: word): boolean;*

Is the specified element valid for this matrix?

IsZero *function tMatrix.IsZero: boolean; virtual;*

Is number equal to zero?

Overrides tMathObject.IsZero.

Multiply *procedure tMatrix.Multiply(What: pMathObject); virtual;*

Multiplies this object with another object.

Overrides tMathObject.Multiply.

MultiplyRow *procedure tMatrix.MultiplyRow(Row: word; What: tBaseReal);*

Multiplies the specified row with a scalar (real number).

MultiplyScalar *procedure tMatrix.MultiplyScalar(What: tBaseReal);*

Multiplies this matrix with a scalar (real number).

Normalize *procedure tMatrix.Normalize(Result: pMatrix);*

Normalizes the matrix (creates ones in the diagonal).

Reference *function tMatrix.Reference(Row, Column: word): word;*

Returns the data type element for the matrix element.

Rows *function tMatrix.Rows: word;*

Returns the number of rows in this matrix.

SetElement *procedure tMatrix.SetElement(Row, Column: word; Value: tBaseReal);*

Sets the value of an element in the matrix.

StreamLoad *constructor tMatrix.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

Overrides tStaticElement.StreamLoad.

Overrides tElement.StreamLoad.

Overrides tObject.StreamLoad.

StreamStore *procedure tMatrix.StreamStore(Stream: pStream); virtual;*

Stores an instance to a stream.

Overrides tStaticElement.StreamStore.

Overrides tElement.StreamStore.

Overrides tObject.StreamStore.

StreamWrite *procedure tMatrix.StreamWrite(Stream: pStream); virtual;*

Writes this math object as formatted text, e.g. "2+2i".

Overrides tElement.StreamWrite.

Overrides tObject.StreamWrite.

Subtract *procedure tMatrix.Subtract(What: pMathObject); virtual;*

Subtracts an object from this object.

Overrides tMathObject.Subtract.

SwapRow *procedure tMatrix.SwapRow(FirstRow, SecondRow: word; Result: pMatrix);*

Swaps the locations of two rows in this matrix.

Transpose *procedure tMatrix.Transpose;*

Transposes this matrix (shifts rows and columns).

Triangulate *function tMatrix.Triangulate(Result: pMatrix): boolean;*

tMemoryStream EFMEMORY

EFMEMORY.tMemoryStream = object(tStream)

Fields and Methods:

tMergeSortPlug EFSORT

EFSORT.tMergeSortPlug = object(tSortPlug)

Merge sort plug class derived from *tSortPlug*. This class provides the Merge sort algorithm. Merge sort is fast, even in the worst-case scenario, but may require at great deal of memory. Consider *tQuickSortPlug* if the number of elements is large. Complexity: O(N * 2-log N).

Fields and Methods:

public
constructor *Initialize...*
procedure *Sort...* virtual;
procedure *Merge...*
constructor *StreamLoad...*

Methods

Initialize *constructor tMergeSortPlug.Initialize(PlugManager: pPlugManager);*

Constructs a plug instance for the specified manager.

Overrides tPlug.Initialize.

Overrides tObject.Initialize.

Merge *procedure tMergeSortPlug.Merge(Start, Middle, Stop: word; Order: tSortOrder);*

Merges two segments (used by the algorithm).

Sort *procedure tMergeSortPlug.Sort(Start, Stop: word; Order: tSortOrder); virtual;*

Sorts elements within the specified interval.

Overrides tSortPlug.Sort.

StreamLoad *constructor tMergeSortPlug.StreamLoad(Stream: pStream);*

Constructs and loads an instance from a stream.

Overrides tSortPlug.StreamLoad.

Overrides tPlug.StreamLoad.

Overrides tStaticElement.StreamLoad.

Overrides tElement.StreamLoad.

Overrides tObject.StreamLoad.

tMessage EFCMAN

EFCMAN.tMessage = object(tStaticElement)

This is the abstract message class. Messages are actions or requests of actions that are sent to components: they are the communication link between different classes in EFLIB.

Messages must know what actions should be performed when the message is received. Actions can be defined in the *Action* method. Messages that do not provide any actions are called strict event carriers or just events. The receiver of such a message is itself responsible for triggering the associated actions. For instance, a key press results in an event, that is recognized by a component. The component then decide what to do with the key press.

Message transmittance principles:

(1) When the receiver has processed the message or recognized an event, it is resposible for the destruction of the message instance.

(2) If the receiver does not recognize an event, it must be redirected or if there is nowhere to redirect it, it must be rejected (bounce back to the sender marked as undelivered).

In addition, messages can must know the receiver, and can now the sender - both are identified with pointers to *tComponent* instances.

Messages can connect to each other, resulting in message queues. When a component receives an entire structure of messages, it must process them one-by-one until all events have been processed. While the component handles these queued messages, other components must wait.

Messages can do several things, e.g. activate a component, invoke an external component, teminate a process or cancel an operation. The basic set of messages are defined in the *EFMSG* unit.

Fields and Methods:

public
constructor *Initialize...*
procedure *SetSender...* virtual;
procedure *SetReceiver...* virtual;
procedure *Action*; virtual;
procedure *Reject*; virtual;
function *Receiver...* virtual;
function *Sender...* virtual;
function *IsReceiver...* virtual;
function *IsDependent...* virtual;
function *IsUndelivered...* virtual;
private
fSender: *pComponent*;
fReceiver: *pComponent*;
fUndelivered: boolean;

Descendants:

- *tComponent*
  - *tManager*
    - *EFAPP.tApplication*
    - *tCommunicator*
  - *EFCONDRV.tConsole*
    - *EFCONDRV.tBoundedConsole*
      - *EFCONDRV.tCRT*
        - *EFSCREEN.tScreen*
      - *EFCONDRV.tVirtualConsole*
        - *EFCONDRV.tImage*
        - *EFSCREEN.tBufferedScreen*
  - *EFDEVICE.tDevice*
    - *EFDEVICE.tSoundDevice*
    - *EFMOUSE.tMouseDevice*
  - *EFVIEWS.tView*
    - *EFVIEWS.tGroup*

tServiceMessage
EFDEVICE.tInstallMessage
EFKEYBRD.tKeyEvent
EFMOUSE.tMouseEvent
EFMSG.tFocusedEvent
EFMSG.tLocalMessage

Fields

fReceiver *tMessage.fReceiver: pComponent;*

The receiver of this message (required).

fSender *tMessage.fSender: pComponent;*

The sender of this message (optional).

fUndelivered *tMessage.fUndelivered: boolean;*

Is this message undelivered, i.e. not reached receiver?

Methods

Action *procedure tMessage.Action; virtual;*

Does something when this message is processed.

Initialize *constructor tMessage.Initialize(MessageReceiver, MessageSender: pComponent);*

Initializes a message to the specified receiver and sender.

Overrides tObject.Initialize.

IsDependent *function tMessage.IsDependent: boolean; virtual;*

Returns TRUE if the receiver can destruct the message.

IsReceiver *function tMessage.IsReceiver(Component: pComponent): boolean; virtual;*

Is the specified component the receiver of this message?

IsUndelivered *function tMessage.IsUndelivered: boolean; virtual;*

Returns TRUE if this message could not be delivered.

Receiver *function tMessage.Receiver: pComponent; virtual;*

Returns the receiver of this message (required).

Reject *procedure tMessage.Reject; virtual;*

Rejects this message and classifies it as undelivered.

Sender *function tMessage.Sender: pComponent; virtual;*

Returns the sender of this message or NIL if unknown.

SetReceiver *procedure tMessage.SetReceiver(Component: pComponent); virtual;*

Sets the receiver of this message.

SetSender *procedure tMessage.SetSender(Component: pComponent); virtual;*

Sets the sender of this message (optional).

tMouseDevice EFMOUSE

EFMOUSE.tMouseDevice = object(tDevice)

Fields and Methods:

public
constructor *Initialize*;
destructor *Intercept*; virtual;
procedure *EnableDriver*;
procedure *DisableDriver*;
procedure *SetSensitivity*...
procedure *SetDoubleSpeedTreshold*...
procedure *DisableTreshold*;
procedure *SetHardwareCursor*...
procedure *SetSoftwareCursor*...
procedure *SetEventHandler*...
procedure *SetDefaultEventHandler*...
procedure *Reset*;
procedure *ShowCursor*;
procedure *HideCursor*;
procedure *GotoXY*...
procedure *CursorBox*...
procedure *ConditionalOff*...
function *WaitForButtonPress*...
function *WaitForButtonRelease*...
function *WhereX*...
function *WhereY*...
function *LastPressedX*...
function *LastPressedY*...
function *LastReleasedX*...
function *LastReleasedY*...

function *ButtonPressed...*
function *LeftButtonPressed...*
function *RightButtonPressed...*
function *MiddleButtonPressed...*
function *ThisButtonPressed...*
function *ButtonPresses...*
function *ButtonReleases...*
function *XMotion...*
function *YMotion...*
function *IsInstalled...* virtual;
function *IsPressedInside...*
function *IsReleasedInside...*
function *IsMovedUp...*
function *IsMovedDown...*
function *IsMovedLeft...*
function *IsMovedRight...*
function *IsInMotion...*
function *IsInside...*

Methods

ButtonPressed *function tMouseDevice.ButtonPressed: boolean;*

Is a button pressed?

ButtonPresses *function tMouseDevice.ButtonPresses(Button: word): word;*

How many button presses since last call?

ButtonReleases *function tMouseDevice.ButtonReleases(Button: word): word;*

How many button releases since last call?

ConditionalOff *procedure tMouseDevice.ConditionalOff(X1, Y1, X2, Y2: word);*

Hide cursor if it enters the specified region.

CursorBox *procedure tMouseDevice.CursorBox(X1, Y1, X2, Y2: word);*

Set cursor region, ie. the allowed coordinates for mouse.

DisableDriver *procedure tMouseDevice.DisableDriver;*

Disables the mouse driver.

DisableTreshold *procedure tMouseDevice.DisableTreshold;*

Disbable treshold,

EnableDriver *procedure tMouseDevice.EnableDriver;*

Enables the mouse driver.

GotoXY *procedure tMouseDevice.GotoXY(X, Y: word);*

Change cursor position to (X,Y).

HideCursor *procedure tMouseDevice.HideCursor;*

Hide the mouse cursor.

Initialize *constructor tMouseDevice.Initialize;*

Initializes and constructs an instance of this type.

Overrides tComponent.Initialize.

Overrides tMessage.Initialize.

Overrides tObject.Initialize.

Intercept *destructor tMouseDevice.Intercept; virtual;*

Intercepts and destructs an instance of this type.

Overrides tElement.Intercept.

Overrides tObject.Intercept.

IsInMotion *function tMouseDevice.IsInMotion: boolean;*

Is mouse in motion, ie. moved in any direction?

IsInside *function tMouseDevice.IsInside(X1, Y1, X2, Y2: byte): boolean;*

Is the mouse cursor inside the specified boundaries?

IsInstalled *function tMouseDevice.IsInstalled: boolean; virtual;*

Is driver installed?

Overrides tDevice.IsInstalled.

IsMovedDown *function tMouseDevice.IsMovedDown: boolean;*

Is mouse moved down?

IsMovedLeft *function tMouseDevice.IsMovedLeft: boolean;*

Is mouse moved left?

IsMovedRight *function tMouseDevice.IsMovedRight: boolean;*

Is mouse moved right?

IsMovedUp *function tMouseDevice.IsMovedUp: boolean;*

Is mouse moved up?

IsPressedInside *function tMouseDevice.IsPressedInside(Button: word; X1, Y1, X2, Y2: byte): boolean;*

Is the specified button pressed inside boundaries?

IsReleasedInside *function tMouseDevice.IsReleasedInside(Button: word; X1, Y1, X2, Y2: byte): boolean;*

Is the specified button released inside boundaries?

LastPressedX *function tMouseDevice.LastPressedX(Button: word): word;*

Returns the X coordinate for last button press.

LastPressedY *function tMouseDevice.LastPressedY(Button: word): word;*

Returns the Y coordinate for last button press.

LastReleasedX *function tMouseDevice.LastReleasedX(Button: word): word;*

Returns the X coordinate for last button release.

LastReleasedY *function tMouseDevice.LastReleasedY(Button: word): word;*

Returns the Y coordinate for last button release.

LeftButtonPressed *function tMouseDevice.LeftButtonPressed: boolean;*

Is left button pressed?

MiddleButtonPressed *function tMouseDevice.MiddleButtonPressed: boolean;*

Is middle button pressed?

Reset *procedure tMouseDevice.Reset;*

Reset mouse driver status.

RightButtonPressed *function tMouseDevice.RightButtonPressed: boolean;*

Is right button pressed?

SetDefaultEventHandler *procedure tMouseDevice.SetDefaultEventHandler(Mask: word);*

Enable the default event handler.

SetDoubleSpeedTreshold *procedure tMouseDevice.SetDoubleSpeedTreshold(Treshold: word);*

Enable / set treshold .

SetEventHandler *procedure tMouseDevice.SetEventHandler(Mask: word; Handler: pointer);*

Set event handler procedure.

SetHardwareCursor *procedure tMouseDevice.SetHardwareCursor(Start, Stop: word);*

Set hardware text cursor to specified size.

SetSensitivity *procedure tMouseDevice.SetSensitivity(X, Y: word);*

Set mickey / pixel ratio (mouse speed).

SetSoftwareCursor *procedure tMouseDevice.SetSoftwareCursor(ScreenMask, CursorMask: word);*

Set software text cursor to specified mask.

ShowCursor *procedure tMouseDevice.ShowCursor;*

Show the mouse cursor.

ThisButtonPressed *function tMouseDevice.ThisButtonPressed(Button: word): boolean;*

Is specified button pressed?

WaitForButtonPress *function tMouseDevice.WaitForButtonPress(Button: byte; Timeout: word): boolean;*

Wait for button press or until specified time passes.

WaitForButtonRelease *function tMouseDevice.WaitForButtonRelease(Button: byte; Timeout: word): boolean;*

Wait for button release or until specified time passes.

WhereX *function tMouseDevice.WhereX: word;*

Returns the current mouse cursor X coordinate.

WhereY *function tMouseDevice.WhereY: word;*

Returns the curernt mouse cursor Y coordinate.

XMotion *function tMouseDevice.XMotion: integer;*

X motion in mickeys (the mouse X speed).

YMotion *function tMouseDevice.YMotion: integer;*

Y motion in mickeys (the mouse Y speed).

tMouseEvent EFMOUSE

EFMOUSE.tMouseEvent = object(tMessage)

Fields and Methods:

public
constructor *Initialize*;
destructor *Intercept*; virtual;
procedure *Reset*; virtual;
procedure *RegisterPress*; virtual;
function *IsMoved...* virtual;
function *IsDoubleClicked...* virtual;
function *IsDoubleClickedInField...* virtual;
function *LastX...* virtual;
function *LastY...* virtual;
private
DoubleClickTimer: *tTimer*;
DoubleClickIndex: word;
Coordinates: *tCoordinates*;

Fields

Coordinates *tMouseEvent.Coordinates: tCoordinates;*

Coordinate for the first button press.

DoubleClickIndex *tMouseEvent.DoubleClickIndex: word;*

Click counter (number of button presses).

DoubleClickTimer *tMouseEvent.DoubleClickTimer: tTimer;*

Time in milliseconds since last click.

Methods

Initialize *constructor tMouseEvent.Initialize;*

Initializes and constructs an instance of this type.

Overrides tMessage.Initialize.

Overrides tObject.Initialize.

Intercept *destructor tMouseEvent.Intercept; virtual;*

Intercepts and destructs an instance of this type.

Overrides tElement.Intercept.

Overrides tObject.Intercept.

IsDoubleClicked *function tMouseEvent.IsDoubleClicked: boolean; virtual;*

Is mouse double clicked?

IsDoubleClickedInField *function tMouseEvent.IsDoubleClickedInField(Field: tField): boolean; virtual;*

Is mouse clicked inside the specified field?

IsMoved *function tMouseEvent.IsMoved: boolean; virtual;*

Has the mouse moved?

LastX *function tMouseEvent.LastX: word; virtual;*

Last X coordinate with button press.

LastY *function tMouseEvent.LastY: word; virtual;*

Last Y coordinate with button press.

RegisterPress *procedure tMouseEvent.RegisterPress; virtual;*

Registers a button click event.

Reset *procedure tMouseEvent.Reset; virtual;*

Reset click status.

tNamedElement EFSYSTEM

EFSYSTEM.tNamedElement = object(tStaticElement)

This class defines a named and static element. The element is named since it is associated to some string - the name. It is static since the element is itself an independent class. However, named elements are not entirely static since the name may vary between different classes. tNamedElements are ordered according to their names.

This is the parent class for classes like tFont and tColor. For more information, see *tStaticElement*.

Fields and Methods:

public
constructor *Initialize...*
function *Data...* virtual;
function *Size...* virtual;
procedure *Dispose*; virtual;
function *Name...* virtual;
constructor *StreamLoad...*
procedure *StreamStore...* virtual;
private
fName: *pString*;

Descendants:

tFlag
tMapping
tProfile
  tEnvironment
  tSystemProfile
tSetting
  tComponentSelector

Fields

fName *tNamedElement.fName: pString;*

The name of this static element.

Methods

Data *function tNamedElement.Data(Position: word): pointer; virtual;*

Returns a pointer to contents at a position within [0,n].

Overrides tElement.Data.

Dispose *procedure tNamedElement.Dispose; virtual;*

Disposes the string that holds the name of the element.

Overrides tElement.Dispose.

Initialize *constructor tNamedElement.Initialize(NameOfElement: string);*

Initializes a named element with the specified name.

Overrides tObject.Initialize.

Name *function tNamedElement.Name: string; virtual;*

Returns the name of this element.

Size *function tNamedElement.Size: word; virtual;*

Returns the size of the contents of this element in bytes.

Overrides tElement.Size.

StreamLoad *constructor tNamedElement.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

Overrides tStaticElement.StreamLoad.

Overrides tElement.StreamLoad.

Overrides tObject.StreamLoad.

StreamStore *procedure tNamedElement.StreamStore(Stream: pStream); virtual;*

Stores an instance to a stream.

Overrides tStaticElement.StreamStore.

Overrides tElement.StreamStore.

Overrides tObject.StreamStore.

tNodeIterator EFLIST

EFLIST.tNodeIterator = object(tLinearIterator)

Node iterator class that provides a fast mechanism of walking through elements in a doubly linked data structure, that is, a descendant of the *tList* class.

Fields and Methods:

public
constructor *Initialize...*
procedure *First*; virtual;
procedure *Last*; virtual;
function *Node...* virtual;
function *Element...* virtual;
procedure *Walk...* virtual;
procedure *WalkTo...* virtual;
procedure *WalkUsing...* virtual;
procedure *WalkForward*; virtual;

procedure *WalkBackward*; virtual;
constructor *StreamLoad...*
function *IsEnd...* virtual;
private
fCursorNode: *pLinkage*;

Fields

fCursorNode *tNodeIterator.fCursorNode: pLinkage;*

Cursor node (ie. the current position).

Methods

Element *function tNodeIterator.Element: pElement; virtual;*

Returns a pointer to the current element instance.

Overrides tLinearIterator.Element.

Overrides tIterator.Element.

First *procedure tNodeIterator.First; virtual;*

Walks to the first element in this ADT.

Overrides tLinearIterator.First.

Overrides tIterator.First.

Initialize *constructor tNodeIterator.Initialize(ADT: pList);*

Initializes an iterator for the specified ADT.

Overrides tLinearIterator.Initialize.

Overrides tObject.Initialize.

IsEnd *function tNodeIterator.IsEnd: boolean; virtual;*

Is iterator at boundaries (beginning or end)?

Overrides tLinearIterator.IsEnd.

Overrides tIterator.IsEnd.

Last *procedure tNodeIterator.Last; virtual;*

Walks to the last element in this ADT.

Overrides tLinearIterator.Last.

Overrides tIterator.Last.

Node *function tNodeIterator.Node: pLinkage; virtual;*

Returns the container for the current element (tLinkage).

StreamLoad *constructor tNodeIterator.StreamLoad(Stream: pStream);*

Constructs and loads an instance from a stream.

Overrides tLinearIterator.StreamLoad.

Overrides tObject.StreamLoad.

Walk *procedure tNodeIterator.Walk(Steps: integer); virtual;*

Walks some steps relative to the current element.

Overrides tLinearIterator.Walk.

Overrides tIterator.Walk.

WalkBackward *procedure tNodeIterator.WalkBackward; virtual;*

Walk backwards (to the predecessor element).

Overrides tLinearIterator.WalkBackward.

Overrides tIterator.WalkBackward.

WalkForward *procedure tNodeIterator.WalkForward; virtual;*

Walk forwards (to the successor element).

Overrides tLinearIterator.WalkForward.

Overrides tIterator.WalkForward.

WalkTo *procedure tNodeIterator.WalkTo(Where: word); virtual;*

Walks to a position inside the associated ADT.

Overrides tLinearIterator.WalkTo.

Overrides tIterator.WalkTo.

WalkUsing *procedure tNodeIterator.WalkUsing(Iterator: pIterator; Steps: integer); virtual;*

Use the specified iterator to walk some steps.

Overrides tLinearIterator.WalkUsing.

Overrides tIterator.WalkUsing.

tNullStream EFSTREAM

EFSTREAM.tNullStream = object(tStream)

This stream acts a null device, that is it pretends to read and write data without actually doing it. All properties of streams are inherited.

Fields and Methods:

public
constructor *Initialize*;
procedure *Read*... virtual;
procedure *Write*... virtual;

Methods

Initialize *constructor tNullStream.Initialize;*

Initializes a stream that operates acts a null device .

Overrides tStream.Initialize.

Overrides tObject.Initialize.

Read *procedure tNullStream.Read(var Data; Count: word); virtual;*

Reads data from stream into a variable.

Overrides tStream.Read.

Write *procedure tNullStream.Write(var Data; Count: word); virtual;*

Writes data to stream from a variable.

Overrides tStream.Write.

tNumberElement EFSYSTEM

EFSYSTEM.tNumberElement = object(tElement)

Fields and Methods:

private
fNumber: *tBaseReal*;

Fields

fNumber *tNumberElement.fNumber: tBaseReal;*

tObject EFKERNEL

EFKERNEL.tObject = object

This is an abstract parent class. All other classes are derived from the tObject class. There are no instances of tObject. However, this class provides a basic mechanism and structure for type checking and encapsulation: a minimal definition for what a derived class must do. An object must:

(1) Construct and destruct with *Initialize* and *Intercept.*

(2) Define when two instances are considered equal. The *IsEqual* method does this.

(3) Define when two instances are compatible, ie. when data can be exchanged. The method *IsCompatible* returns the compatibility of two instances.

Fields and Methods:

public
constructor *Initialize*;
destructor *Intercept*; virtual;
destructor *Free*; virtual;
constructor *StreamLoad...*
procedure *StreamStore...* virtual;
procedure *StreamWrite...* virtual;
procedure *Assert...* virtual;
procedure *Error...* virtual;
procedure *Abstract*;
function *IsInitialized...*
function *IsValidStream...*
function *IsEqual...* virtual;
function *IsCompatible...* virtual;
function *IsParent...*
function *IsDescendant...*
function *IsDerived...*
function *IsEqualType...*
function *ClassIdentity...*
function *NameOfType...*
function *Identify...*

Descendants:

EFADT.tADT
EFADT.tLinearADT
EFARRAY.tArray
EFARRAY.tBoundedArray

        EFARRAY.tVirtualArray
    EFLIST.tList
      EFLIST.tCursorList
      EFLIST.tReversedList
        EFLIST.tAdjustedList
        EFLIST.tOrderedList
  EFADT.tOrderedADT
    EFDATA.tCompositeADT
      EFDATA.tQueue
        EFDATA.tCircularQueue
        EFDATA.tPriorityQueue
      EFDATA.tStack
      EFTABLE.tHashTable
        EFTABLE.tBucketHashTable
        EFTABLE.tOpenHashTable
          EFTABLE.tDoubleHashTable
          EFTABLE.tProbeHashTable
    EFTREE.tBinaryTree
      EFTREE.tAVLTree
      EFTREE.tSplayTree
    EFTREE.tTree
EFADT.tADTSelector
EFADT.tIterator
  EFADT.tLinearIterator
    EFARRAY.tSparceIterator
    EFLIST.tNodeIterator
  EFTABLE.tBucketIterator
  EFTREE.tTreeIterator
EFBASIC.tSet
  EFBASIC.tCharacters
  EFSTYLES.tStyle
EFBASIC.tTime
EFBASIC.tTimer
EFCONDRV.tConsoleSelector
EFELEM.tElement
  EFELEM.tCarrierElement
  EFELEM.tGenericElement
    EFMEMORY.tAllocation
      EFMEMORY.tBuffer
    EFSTRING.tString
      EFPATRN.tPatternString
      EFSTRING.tToken
  EFELEM.tStaticElement
    EFCMAN.tMessage

  - *EFCMAN.tComponent*
    - *EFCMAN.tManager*
      - *EFAPP.tApplication*
      - *EFCMAN.tCommunicator*
    - *EFCONDRV.tConsole*
      - *EFCONDRV.tBoundedConsole*
        - *EFCONDRV.tCRT*
          - *EFSCREEN.tScreen*
        - *EFCONDRV.tVirtualConsole*
          - *EFCONDRV.tImage*
          - *EFSCREEN.tBufferedScreen*
    - *EFDEVICE.tDevice*
      - *EFDEVICE.tSoundDevice*
      - *EFMOUSE.tMouseDevice*
    - *EFVIEWS.tView*
      - *EFVIEWS.tGroup*
  - *EFCMAN.tServiceMessage*
  - *EFDEVICE.tInstallMessage*
  - *EFKEYBRD.tKeyEvent*
  - *EFMOUSE.tMouseEvent*
  - *EFMSG.tFocusedEvent*
  - *EFMSG.tLocalMessage*
- *EFELEM.tIdentityElement*
- *EFMATH.tMathObject*
  - *EFMATH.tComplex*
  - *EFMATH.tInteger*
  - *EFMATH.tMatrix*
    - *EFMATH.tPolynomial*
  - *EFMATH.tRational*
  - *EFMATH.tReal*
  - *EFMATH.tVector*
  - *EFMEXP.tExpression*
  - *EFMEXP.tMathFunction*
- *EFPLUG.tPlug*
  - *EFADT.tADTPlug*
    - *EFADT.tKeyPlug*
      - *EFADT.tBoundedKeyPlug*
      - *EFADT.tReversedKeyPlug*
    - *EFADT.tSortPlug*
      - *EFADT.tQuickSortPlug*
      - *EFSORT.tBubbleSortPlug*
      - *EFSORT.tInsertionSortPlug*
      - *EFSORT.tMergeSortPlug*
    - *EFTABLE.tHashPlug*

          - *EFTABLE.tTextHashPlug*
        - *EFTREE.tPathPlug*
    - *EFSYSTEM.tNamedElement*
      - *EFSYSTEM.tFlag*
      - *EFSYSTEM.tMapping*
      - *EFSYSTEM.tProfile*
        - *EFSYSTEM.tEnvironment*
        - *EFSYSTEM.tSystemProfile*
      - *EFSYSTEM.tSetting*
        - *EFSYSTEM.tComponentSelector*
    - *EFTABLE.tBucket*
    - *EFTIME.tTiming*
      - *EFTIME.tDate*
        - *EFTIME.tTimeDate*
      - *EFTIME.tTime*
  - *EFELEM.tStreamElement*
  - *EFMEMORY.tStreamHandle*
  - *EFSYSTEM.tLocalElement*
  - *EFSYSTEM.tNumberElement*
- *EFIO.tDirectory*
- *EFIO.tFilename*
  - *EFIO.tPath*
- *tClassManager*
- *tClassRegister*
  - *tCarrierRegister*
    - *tErrorHandler*
  - *EFREG.tRegister*
- *tContainer*
  - *EFBASIC.tCoordinates*
  - *EFBASIC.tField*
  - *tCarrier*
    - *EFLIST.tLinkage*
      - *EFTREE.tTreeLinkage*
        - *EFTREE.tAVLLinkage*
  - *tClassIdentity*
  - *EFSYSTEM.tClassSelector*
- *tError*
  - *tErrorMessage*
- *tHandle*
- *tSetup*
- *tStream*
  - *EFADT.tADTStream*
  - *EFFILTER.tFilter*
    - *EFCONV.tConverter*

EFFILTER.tBufferFilter
EFFILE.tFile
EFFILE.tIFile
EFFILE.tOFile
EFFILE.tTemporaryFile
EFRES.tStreamableFile
EFFILTER.tDuplicateFilter
EFFILTER.tEncryptFilter
EFFILTER.tSequentialFilter
EFFILTER.tCompressFilter
EFFILTER.tCRC16Filter
EFFILTER.tCRC32Filter
EFFILTER.tTextFilter
EFRES.tResource
EFRES.tArchive
EFSEARCH.tMatchFilter
EFSEARCH.tSearchFilter
EFSEARCH.tDelayedSearchFilter
EFSEARCH.tReplaceFilter
EFSYSTEM.tSwapFilter
EFMEMORY.tMemoryStream
EFSTREAM.tDataStream
EFSTREAM.tFileStream
EFSTREAM.tNullStream
EFSTREAM.tStandardStream
EFKEYBRD.tKeyboard
EFMATH.tCaster
EFMATH.tMathFunction
EFMATH.tSolver
EFPLUG.tPlugManager
EFREG.tHandle
EFSTRING.tStringMatcher
EFSTRING.tTranslation
EFTEXT.tTextResource
EFTEXT.tText
EFTEXT.tHyperText
EFTEXT.tParser
EFCLINE.tCommandLine
EFTEXT.tTextFile
EFTIME.tTimer
EFVIEWS.tOldView

Methods

Abstract *procedure tObject.Abstract;*

Error due to abstract method call.

Assert *procedure tObject.Assert(Condition: boolean; Identity: word); virtual;*

Asserts the condition or reports an error.

ClassIdentity *function tObject.ClassIdentity: word;*

Returns the class identity number (see *tClassManager*).

Error *procedure tObject.Error(Identity: word); virtual;*

Handles a given error.

Free *destructor tObject.Free; virtual;*

Calls Intercept and releases the memory for this instance.

Identify *function tObject.Identify: pClassIdentity;*

Returns the class identity instance for this class.

Initialize *constructor tObject.Initialize;*

Initializes and constructs an instance of this type.

Intercept *destructor tObject.Intercept; virtual;*

Intercepts and destructs an instance of this type.

IsCompatible *function tObject.IsCompatible(Instance: pObject): boolean; virtual;*

Are classes type-compatible (fully replacable).

IsDerived *function tObject.IsDerived(Instance: pObject): boolean;*

Is the specified instance any subclass to this class?

IsDescendant *function tObject.IsDescendant(Instance: pObject): boolean;*

Is the specified instance an immediate subclass?

IsEqual *function tObject.IsEqual(Instance: pObject): boolean; virtual;*

Are contents of these instances equal (bitwise compare)?

IsEqualType *function tObject.IsEqualType(Instance: pObject): boolean;*

Are instances of the same class type?

IsInitialized *function tObject.IsInitialized: boolean;*

Is this instance initialized?

IsParent *function tObject.IsParent(Instance: pObject): boolean;*

Is the specified instance any superclass to this class?

IsValidStream *function tObject.IsValidStream(Stream: pStream): boolean;*

Is the specified stream valid for storage?

NameOfType *function tObject.NameOfType: string;*

Returns the full class type name (eg. "tObject").

StreamLoad *constructor tObject.StreamLoad(Stream: pStream);*

Constructs and loads an instance from a stream.

StreamStore *procedure tObject.StreamStore(Stream: pStream); virtual;*

Stores an instance to a stream.

StreamWrite *procedure tObject.StreamWrite(Stream: pStream); virtual;*

Writes formatted text contents to a stream (optional).

tOFile EFFILE

EFFILE.tOFile = object(tFile)

This class implements a write only *tFile*. Any existing file is automatically overwritten.

Fields and Methods:

public
constructor *Initialize...*

Methods

Initialize *constructor tOFile.Initialize(Filename: string; SizeOfBuffer: word);*

Initializes a file stream, optionally with an buffer.

Overrides tFile.Initialize.

Overrides tBufferFilter.Initialize.

Overrides tFilter.Initialize.

Overrides tStream.Initialize.

Overrides tObject.Initialize.

tOldView EFVIEWS

EFVIEWS.tOldView = object(tObject)

Abstract view base class. A view is a output region with all properties of a screen. Visual components don't know the details of the view it operates on. Hence, the user can implement visual components that act on a invisible view as well as directly on the screen.

Views are buffered, ie. they use small memory regions to store the contents. When a buffer flush is requested, the view dumps it's contents to the assigned screen driver (tScreen in EFSCREEN). Normally, the dump is done immediately after the view has been updated. It is however possible to delay the flush. Use the *Wait* method and specify the number of updates that are permitted before the view is automatically flushed (0 = flush immediately).

If the update is delayed, the view stores all updated regions in a structure. When the flush method is called, all the updated regions are updated. If *fMaximumUpdates* number of updates has occurred, the view automatically redraws all of it's contents and clears the update queue (*fUpdates*).

Fields and Methods:

public
procedure *Flush*; virtual;
procedure *Update...* virtual;
procedure *Wait...* virtual;
procedure *DrawLine...* virtual;
procedure *Put...* virtual;
function *Get...* virtual;
procedure *PutAttribute...* virtual;
function *GetAttribute...*
constructor *StreamLoad...*
procedure *StreamStore...* virtual;
procedure *FetchDriver*; virtual;

procedure *UpdateDriver*; virtual;
procedure *FetchStatus*... virtual;
procedure *UpdateStatus*... virtual;
private
fBuffer: *pAllocation*;
fUpdates: *pField*;
fUpdatesCounter: word;
fMaximumUpdates: word;
fWait: word;
function *Buffer*...

Fields

fBuffer *tOldView.fBuffer: pAllocation;*

View buffer: text and graphics in the view.

fMaximumUpdates *tOldView.fMaximumUpdates: word;*

Maximum update requests for full view transfer to driver.

fUpdates *tOldView.fUpdates: pField;*

Update request queue passed to flush (carriers of tField).

fUpdatesCounter *tOldView.fUpdatesCounter: word;*

Number of update requests.

fWait *tOldView.fWait: word;*

Number of remaining updates before buffer flush.

Methods

Buffer *function tOldView.Buffer(nX, nY: word): pointer;*

Returns a pointer to the character at (X,Y) in the buffer.

DrawLine *procedure tOldView.DrawLine(X1, Y1: word; X2, Y2: word; Character: char; nAttribute: byte); virtual;*

Draws a horizontal, vertical or diagonal line.

FetchDriver *procedure tOldView.FetchDriver; virtual;*

Fetchs status from the driver (position, attribute).

FetchStatus *procedure tOldView.FetchStatus(View: pView); virtual;*

Fetchs the status of another view.

Flush *procedure tOldView.Flush; virtual;*

Flushes the view buffer immediately.

Get *function tOldView.Get(nX, nY: word): char; virtual;*

Gets a character from the specified coordinate.

GetAttribute *function tOldView.GetAttribute(nX, nY: word): byte;*

Gets an attribute from the specified coordinate.

Put *procedure tOldView.Put(nX, nY: word; Character: char); virtual;*

Puts a character on the specified coordinate.

PutAttribute *procedure tOldView.PutAttribute(nX, nY: word; nAttribute: byte); virtual;*

Puts an attribute on the specified coordinate.

StreamLoad *constructor tOldView.StreamLoad(Stream: pStream);*

Constructs and loads an instance from a stream.

Overrides tObject.StreamLoad.

StreamStore *procedure tOldView.StreamStore(Stream: pStream); virtual;*

Stores an instance to a stream.

Overrides tObject.StreamStore.

Update *procedure tOldView.Update(Field: pField); virtual;*

Request an update of the specified field (schedule flush).

UpdateDriver *procedure tOldView.UpdateDriver; virtual;*

Updates driver status (position, attribute).

UpdateStatus *procedure tOldView.UpdateStatus(View: pView); virtual;*

Updates the status of another view from this view.

Wait *procedure tOldView.Wait(Updates: word); virtual;*

Ignores buffer flush for a specified number of updates.

EFTABLE.tOpenHashTable = object(tHashTable)

This class defines an open hash table. This class is abstract and you can therefor not derive any instances from this class. Instead, you must make use one of the specialized classes: *tProbeHashTable* or *tDoubleHashTable*.

A hash table is a generalization of an array into an ordered ADT. For details, see *tHashTable*.

An open hash table is basically an array. However, indexes are calculated according to the content of the inserted element - a hash function (defined in *tHashPlug*) is used. Since there can only be one element for a certain hash code, a open hash table must define some mechanism for handling collisions.

Open hash tables must know if an element has been erased since the rehashing may be obstructed if a rehash chain is broken. Therefor, erased elements are automatically set to an abstract tElement instance - that is, an instance that you may not use. Most users will never have to worry about this mechanism since it is handled internally by *tOpenHashTable*. The implementation is hidden.

Fields and Methods:

public
procedure *Erase...* virtual;
procedure *Put...* virtual;
function *Get...* virtual;
function *Rehash...* virtual;
function *Elements...* virtual;
function *IsUsed...* virtual;
function *IsDeleted...* virtual;

Descendants:

tDoubleHashTable
tProbeHashTable

Methods

Elements *function tOpenHashTable.Elements: word; virtual;*

Returns the number of elements that currently are used.

Overrides tCompositeADT.Elements.

Overrides tADT.Elements.

Erase *procedure tOpenHashTable.Erase(var Key; KeySize: word); virtual;*

Erases element(s) with the specified search key.

Overrides tCompositeADT.Erase.

Overrides tOrderedADT.Erase.

Get *function tOpenHashTable.Get(var Key; KeySize: word): pElement; virtual;*

Gets the specified element instance.

Overrides tCompositeADT.Get.

Overrides tOrderedADT.Get.

IsDeleted *function tOpenHashTable.IsDeleted(Index: word): boolean; virtual;*

Is element deleted?

IsUsed *function tOpenHashTable.IsUsed(Index: word): boolean; virtual;*

Is element used?

Put *procedure tOpenHashTable.Put(Element: pElement); virtual;*

Stores a new element (tElement instance) to this ADT.

Overrides tCompositeADT.Put.

Overrides tADT.Put.

Rehash *function tOpenHashTable.Rehash(Index, Collisions: word): word; virtual;*

Rehashes a hash code to resolve a collision.

tOperator EFMEXP

EFMEXP.tOperator = object(tNamedElement)

This class defines an abstract operator plug. This plug is called whenever an operator is located in an expression, and operators with higher priority report that they do not match the current substring. Then, if this operator match the substring, it may execeute. Otherwise, operators with lower priority are enabled to check the substring.

This class operates directly on the RPN expressions two stacks: the operator and operand stack.

Fields and Methods:

public
procedure *Execute...*
function *IsPattern...*
function *Patternize...*

Methods

Execute *procedure tOperator.Execute(Operands: pStack; Operators: pStack);*

IsPattern *function tOperator.IsPattern: boolean;*

Patternize *function tOperator.Patternize: pStringStub;*

tOrderedADT EFADT

EFADT.tOrderedADT = object(tADT)

This class defines the properties of a ordered data structure, that is an ADT that itself decide where elements are put. Examples on ordered ADT's are hash tables and trees. Elements in ordered data structures are accessed using keys.

Common behavior for tOrderedADT descendants:

(1) Elements are referenced with search keys, generated from the content of an element (the mechanism is provided by a *tKeyPlug* instance).

(2) Special iterator are used with ordered ADT′s. You may have to write new iterators for new ordered ADT′s.

Fields and Methods:

public
procedure *Retrieve...* virtual;
procedure *Update...* virtual;
procedure *Erase...* virtual;
function *Get...* virtual;
function *IsValid...* virtual;
function *IsCompatible...* virtual;

Descendants:

EFDATA.tCompositeADT
  EFDATA.tQueue
    EFDATA.tCircularQueue
    EFDATA.tPriorityQueue
  EFDATA.tStack
  EFTABLE.tHashTable
    EFTABLE.tBucketHashTable
    EFTABLE.tOpenHashTable
      EFTABLE.tDoubleHashTable
      EFTABLE.tProbeHashTable
EFTREE.tBinaryTree
  EFTREE.tAVLTree
  EFTREE.tSplayTree
EFTREE.tTree

Methods

Erase *procedure tOrderedADT.Erase(var Key; KeySize: word); virtual;*

Erases element(s) with the specified search key.

Get *function tOrderedADT.Get(var Key; KeySize: word): pElement; virtual;*

Gets the specified element instance.

IsCompatible *function tOrderedADT.IsCompatible(Instance: pObject): boolean; virtual;*

Are ADT′s compatible, that is has compatible elements.

Overrides tADT.IsCompatible.

Overrides tObject.IsCompatible.

IsValid *function tOrderedADT.IsValid(var Key; KeySize: word): boolean; virtual;*

Is the specified search key valid (existing)?

Retrieve *procedure tOrderedADT.Retrieve(var Key; KeySize: word; var Data); virtual;*

Retrieves element with the specified search key.

Update *procedure tOrderedADT.Update(var Key; KeySize: word; var Data); virtual;*

Updates an element (specified by a search key).

tOrderedList EFLIST

EFLIST.tOrderedList = object(tReversedList)

This class defines an ordered list, that is, a list where elements automatically are inserted in sorted order.

Fields and Methods:

public
constructor *Initialize...*
constructor *Resemble...*
constructor *Duplicate...*
procedure *SetSearchMethod...*
procedure *PutNode...* virtual;
function *LinearAttachNode...* virtual;
function *BinaryAttachNode...* virtual;
constructor *StreamLoad...*
function *IsAttachNode...* virtual;
function *IsIntact...* virtual;
private
fSearchMethod: *tSearchMethod*;

Fields

fSearchMethod *tOrderedList.fSearchMethod: tSearchMethod;*

Search method: linear or binary search for attach nodes.

Methods

BinaryAttachNode *function tOrderedList.BinaryAttachNode(Node: pLinkage): pLinkage; virtual;*

Find attach node using a binary search (fast average-case).

Duplicate *constructor tOrderedList.Duplicate(ADT: pADT);*

Initializes a duplicate ADT instance (with equal elements).

Overrides tReversedList.Duplicate.

Overrides tList.Duplicate.

Initialize *constructor tOrderedList.Initialize(SizeOfElements: word);*

Initializes an instance of this ADT class.

Overrides tReversedList.Initialize.

Overrides tList.Initialize.

Overrides tLinearADT.Initialize.

Overrides tADT.Initialize.

Overrides tObject.Initialize.

IsAttachNode *function tOrderedList.IsAttachNode(AttachNode, Node: pLinkage): boolean; virtual;*

Should the second node be inserted after the first node?

Overrides tList.IsAttachNode.

IsIntact *function tOrderedList.IsIntact: boolean; virtual;*

Is this data structure intact: are links preserved?

Overrides tList.IsIntact.

LinearAttachNode *function tOrderedList.LinearAttachNode(Node: pLinkage): pLinkage; virtual;*

Find attach node using a linear search (good worst-case).

PutNode *procedure tOrderedList.PutNode(Node: pLinkage); virtual;*

Stores a new node (container) in the list.

Overrides tReversedList.PutNode.

Overrides tList.PutNode.

Resemble *constructor tOrderedList.Resemble(ADT: pADT);*

Initializes an ADT that resembles the specified ADT.

Overrides tReversedList.Resemble.

Overrides tList.Resemble.

SetSearchMethod *procedure tOrderedList.SetSearchMethod(SearchMethod: tSearchMethod);*

Sets the search method: linear or binary (default) search.

StreamLoad *constructor tOrderedList.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

Overrides tReversedList.StreamLoad.

Overrides tList.StreamLoad.

Overrides tADT.StreamLoad.

Overrides tObject.StreamLoad.

tParser EFTEXT

EFTEXT.tParser = object(tText)

Fields and Methods:

Descendants:

EFCLINE.tCommandLine

tPath EFIO

EFIO.tPath = object(tFilename)

Fields and Methods:

public

private

EFTREE.tPathPlug = object(tADTPlug)

This is class defines a path-finder for an arbitrary *tTree* data structure. The path-finder determines the location of a certain element by providing a path. The path plug both provide a search mechanism and determines where new elements should be inserted.

Fields and Methods:

public
constructor *Initialize...*
procedure *Install...* virtual;
function *Path...* virtual;
constructor *StreamLoad...*
function *TypeOfPlug...* virtual;

Methods

Initialize

constructor tPathPlug.Initialize(PlugManager: pPlugManager);

Initializes a hash function plug and attempts to install.

Overrides tPlug.Initialize.

Overrides tObject.Initialize.

Install

procedure tPathPlug.Install(PlugManager: pPlugManager); virtual;

Installs the plug in the specified plug manager.

Overrides tADTPlug.Install.

Overrides tPlug.Install.

Path

function tPathPlug.Path(From: pTreeLinkage; var Next: pTreeLinkage; Destination: pElement): boolean; virtual;

Returns the path toward the specified destination element.

StreamLoad

constructor tPathPlug.StreamLoad(Stream: pStream);

Constructs and loads an instance from a stream.

Overrides tPlug.StreamLoad.

Overrides tStaticElement.StreamLoad.

Overrides tElement.StreamLoad.

Overrides tObject.StreamLoad.

TypeOfPlug *function tPathPlug.TypeOfPlug: string; virtual;*

The classification of this plug, e.g. "tSortPlug".

Overrides tADTPlug.TypeOfPlug.

Overrides tPlug.TypeOfPlug.

tPattern EFPATRN

EFPATRN.tPattern = object(tParser)

This class defines a pattern, that is a ordered set of elements that can be compared with other elements (or patterns). Basically, a pattern is a way of transforming text data into instances and vice versa. To determine if a pattern matches some text data (an element), the pattern must normally make several comparisons before it knows if a match is possible.

DEFINITION A pattern P match another element E if and only if:

(1) All the elements in P is equal to some sequence in E.

(2) The elements in P occur in E in the same orderer as they are stored in P.

(3) There is no gap between the matches in E of the elements in P.

The criterias that decide when an element A inside the pattern match a sequence inside another element is defined by the IsPattern method in A.

Fields and Methods:

public
function *IsEqual...* virtual;
private
fADT: *pADT*;

Fields

fADT *tPattern.fADT: pADT;*

Methods

IsEqual *function tPattern.IsEqual(Instance: pObject): boolean; virtual;*

Are the contents of these elements equal?

tPatternManager EFPATRN

EFPATRN.tPatternManager = object(tRegister)

This class maintains a register with patterns and their priority. Classes can ask tPatternManager to attempt to identify what class a pattern is associated to, for instance tInteger or tString.

tPatternManager knows the priority of its patterns. Patterns must have a priority number, since certain patterns should be checked before other: for example, all integers can be converted into strings, but not all strings can become integers. Thus, an integer must be checked before a string. Integers should be given higher priority.

Fields and Methods:

tPatternNumber EFPATRN

EFPATRN.tPatternNumber = object(tElement)

Fields and Methods:

public
function *IsEqual...* virtual;
private
fLow,
fHigh: tBaseReal;
fDecimals: byte;

Fields

fDecimals *tPatternNumber.fDecimals: byte;*

Number of decimals in the number (0 = integer number).

fHigh *See fLow*

fLow *tPatternNumber.fLow,*

fHigh: tBaseReal;

Lower and higher limits for the number: fLow <= N <= fHigh.

Methods

IsEqual *function tPatternNumber.IsEqual(Instance: pObject): boolean; virtual;*

Are the contents of these elements equal?

tPatternString EFPATRN

EFPATRN.tPatternString = object(tString)

This is the most fundamental pattern class. It defines a set of characters and a length. This pattern matches another string S if and only if all characters in S is inside the pattern, and the length of S is equal to the length of the pattern.

Fields and Methods:

public
function *IsEqual...* virtual;
private
fPatternLength: word;
fCharacters: pCharacters;

Fields

fCharacters *tPatternString.fCharacters: pCharacters;*

Valid characters in this pattern (see tCharacters).

fPatternLength *tPatternString.fPatternLength: word;*

Length of this pattern (number of requried characters.

Methods

IsEqual *function tPatternString.IsEqual(Instance: pObject): boolean; virtual;*

Are the contents of these elements equal?

Overrides tElement.IsEqual.

Overrides tObject.IsEqual.

tPlug EFPLUG

EFPLUG.tPlug = object(tStaticElement)

This class defines an a plug, that is a well-defined module that can be attached to any *tPlugManager* instance.

Definition of a plug:

(1) Plugs are owned by a plug manager.

(2) Plugs can ask the plug manager on which class they operate. That class is the owner of the plug manager, for instance an ADT. The owner cannot change.

(3) Plugs are capable of installing on a given class via its plug manager. After installation, the plug is ready to be used on that class whenever the class request a service.

(4) Plugs know what other plugs that are considered compatible, that is what plugs that can replace the plug with respect to the owner class. Compatible plugs are derived from the same specialized plug class as returned by the *TypeOfPlug* method, for example "tSortPlug" or "tKeyPlug". The method *IsCompatible* performs a compatibility check. Plugs are compatible if:

(i) They operate on the same class, or either of them is not assigned to a owner class thus not being installed. (ii) They are inherited from the same, specialized plug class.

(5) A plug must be able to load and store to and from external streams. After loading the plug must be installed (via a manager) to be operable.

Plugs provide high extendibility as well as optimized memory usage since you never install more plugs than you actually will need. Plugs are often only activated when the user requires so. For example, ADTs provide a built-in mechanism for element comparisons. This mechanism is however bypassed whenever a compare plug is installed. ADTs automatically detects an installed plug that provides a compare mechanism that override the built-in mechanism.

Fields and Methods:

public
constructor *Initialize...*
procedure *Install...* virtual;
constructor *StreamLoad...*
function *Manager...* virtual;
function *Owner...* virtual;
function *TypeOfPlug...* virtual;
function *IsInstalled...* virtual;
function *IsCompatible...* virtual;
private
fManager: *pPlugManager*;

Descendants:

EFADT.tADTPlug
- *EFADT.tKeyPlug*
  - *EFADT.tBoundedKeyPlug*
  - *EFADT.tReversedKeyPlug*
- *EFADT.tSortPlug*
  - *EFADT.tQuickSortPlug*
  - *EFSORT.tBubbleSortPlug*
  - *EFSORT.tInsertionSortPlug*
  - *EFSORT.tMergeSortPlug*
- *EFTABLE.tHashPlug*
  - *EFTABLE.tTextHashPlug*
- *EFTREE.tPathPlug*

Fields

fManager *tPlug.fManager: pPlugManager;*

Plug manager class: this class controls all plugs.

Methods

Initialize *constructor tPlug.Initialize(PlugManager: pPlugManager);*

Initializes a container that carries the given instance.

Overrides tObject.Initialize.

Install *procedure tPlug.Install(PlugManager: pPlugManager); virtual;*

Installs the plug in the specified plug manager.

IsCompatible *function tPlug.IsCompatible(Instance: pObject): boolean; virtual;*

Are plugs compatible (see *TypeOfPlug*)?

Overrides tStaticElement.IsCompatible.

Overrides tElement.IsCompatible.

Overrides tObject.IsCompatible.

IsInstalled *function tPlug.IsInstalled: boolean; virtual;*

Returns TRUE if this plug have been installed.

Manager *function tPlug.Manager: pPlugManager; virtual;*

Returns the manager of this plug.

Owner *function tPlug.Owner: pObject; virtual;*

The owner of the plug manager - class that uses the plugs.

StreamLoad *constructor tPlug.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

Overrides tStaticElement.StreamLoad.

Overrides tElement.StreamLoad.

Overrides tObject.StreamLoad.

TypeOfPlug *function tPlug.TypeOfPlug: string; virtual;*

The classification of this plug, e.g. "tSortPlug".

tPlugManager EFPLUG

EFPLUG.tPlugManager = object(tObject)

The plug manager is responsible for several other classes: the plugs (see *tPlug*). tPlugManager is capable of installing new plugs during run-time, and it must detect plug conflicts.

The plug technology is a extentible solution that enables you to package certain operations in a manager. This manager tells the owner what plug it should use for a certain task.

(1) The plug manager must automatically install all its plugs after they have been loaded from a stream.

(2) There can never be two compatible plugs in the same plug manager since the choice of the required plug cannot be ambiguous.

(3) The owner of the manager may not change once the manager have been constructed.

The plug manager can only be assigned to one owner class, a class on which all its plugs must act. The owner is identified with an address to its instance.

Fields and Methods:

public
constructor *Initialize...*
destructor *Intercept*; virtual;
procedure *Install...* virtual;
procedure *InstallAll...* virtual;
function *FindPlug...* virtual;
constructor *StreamLoad...*
procedure *StreamStore...* virtual;
function *Register...* virtual;
function *IsValid...* virtual;
private
fOwnerInstance: *pObject*;
fPlugs: *pClassRegister*;

Fields

fOwnerInstance *tPlugManager.fOwnerInstance: pObject;*

The owner of the plug manager with plugs.

fPlugs *tPlugManager.fPlugs: pClassRegister;*

Plug register instance that maintains the plugs.

Methods

FindPlug *function tPlugManager.FindPlug(TypeOfPlug: pointer; var Plug: pPlug): boolean; virtual;*

Returns a certain plug class or FALSE if it was not found.

Initialize *constructor tPlugManager.Initialize(Owner: pObject);*

Initializes a plug manager for the specified class.

Overrides tObject.Initialize.

Install *procedure tPlugManager.Install(Plug: pPlug); virtual;*

Installs a new plug and removes existing compatible plug.

InstallAll *procedure tPlugManager.InstallAll(Owner: pObject); virtual;*

Installs all plugs (calls their install methods).

Intercept *destructor tPlugManager.Intercept; virtual;*

Intercepts the plug manager and all its plugs.

Overrides tObject.Intercept.

IsValid *function tPlugManager.IsValid(What: pObject): boolean; virtual;*

Can this instance (class) be registered - is it valid?

Register *function tPlugManager.Register: pClassRegister; virtual;*

Provides direct access to the class register instance.

StreamLoad *constructor tPlugManager.StreamLoad(Stream: pStream);*

Constructs and loads an instance from a stream.

Overrides tObject.StreamLoad.

StreamStore *procedure tPlugManager.StreamStore(Stream: pStream); virtual;*

Stores an instance to a stream.

Overrides tObject.StreamStore.

tPolynomial EFMATH

EFMATH.tPolynomial = object(tMatrix)

This class defines a polynomial object with complex coefficients. Polynomials are expressions of the type "ax^3 + bx^2 + cx +d". Polynomes support most of the common operations, such as division, multiplication and addition.

Fields and Methods:

public
constructor *Initialize...*
constructor *Duplicate...*
procedure *SetCoefficient...*
function *Coefficient...*
procedure *Add...* virtual;
procedure *Subtract...* virtual;
procedure *Multiply...* virtual;
procedure *Divide...* virtual;
procedure *Evaluate...*
procedure *GCD...*
constructor *StreamLoad...*
procedure *StreamWrite...* virtual;
function *IsConstant...*
function *Order...*
private
procedure *Adjust*; virtual;

Methods

Add *procedure tPolynomial.Add(What: pMathObject); virtual;*

Adds an object to this object.

Overrides tMatrix.Add.

Overrides tMathObject.Add.

Adjust *procedure tPolynomial.Adjust; virtual;*

Adjusts the dimension of this polynomial according to the highest non-zero term (internal only).

Coefficient *function tPolynomial.Coefficient(Index: word): tBaseReal;*

Retrieves the (real part) value of a coefficient.

Divide *procedure tPolynomial.Divide(What: pMathObject); virtual;*

Divides this object by another object.

Overrides tMatrix.Divide.

Overrides tMathObject.Divide.

Duplicate *constructor tPolynomial.Duplicate(What: pMathObject);*

Initializes an duplicated type-casted mathematical object. #E Element handling methods

Overrides tMatrix.Duplicate.

Evaluate *procedure tPolynomial.Evaluate(Z: pMathObject; var Result: pComplex);*

Evaluates the polynomial P(x) with specified x value.

GCD *procedure tPolynomial.GCD(Nominator, Denominator: pPolynomial);*

Makes this polynomial the GCD of the specified polynomials.

Initialize *constructor tPolynomial.Initialize(nOrder: word);*

Initializes and constructs an instance of this type.

Overrides tMatrix.Initialize.

Overrides tObject.Initialize.

IsConstant *function tPolynomial.IsConstant: boolean;*

Is this a constant polynomial?

Multiply *procedure tPolynomial.Multiply(What: pMathObject); virtual;*

Multiplies this object with another object.

Overrides tMatrix.Multiply.

Overrides tMathObject.Multiply.

Order *function tPolynomial.Order: word;*

Returns the highest orded coefficient in the polynomial.

SetCoefficient *procedure tPolynomial.SetCoefficient(Index: word; Value: tBaseReal);*

Sets the (real) value of a coefficient in the polynomial.

StreamLoad *constructor tPolynomial.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

Overrides tMatrix.StreamLoad.

Overrides tStaticElement.StreamLoad.

Overrides tElement.StreamLoad.

Overrides tObject.StreamLoad.

StreamWrite *procedure tPolynomial.StreamWrite(Stream: pStream); virtual;*

Writes this math object as formatted text, e.g. "2+2i".

Overrides tMatrix.StreamWrite.

Overrides tElement.StreamWrite.

Overrides tObject.StreamWrite.

Subtract *procedure tPolynomial.Subtract(What: pMathObject); virtual;*

Subtracts an object from this object.

Overrides tMatrix.Subtract.

Overrides tMathObject.Subtract.

tPresentationMethod EFDEF

EFDEF.tPresentationMethod = (NoEffect, Float, Push, Sweep, Explode, Shrink, Fade);

This type contains the presentation method (effect) that are used when a view or a visual item is shown on the screen. The effects are: floating (rolls in over the old contents), pushing (pushs away the old contents), sweeping, exploding (from center and out), shrinking (from corners to center) and fade (random).

tPriorityQueue EFDATA

EFDATA.tPriorityQueue = object(tQueue)

This class implements a priority queue. A priority queue is a queue that does not return dequeue elements in a strict FIFO order. Instead a priority queue dequeues elements according to their search key values. See *tQueue.*

The priority queue inserts new elements in sorted order, and dequeus the element with the highest priority. Use *tKeyPlug* to define how the priority is calculated from your data.

Technically, a priority queue is a tQueue ADT based on an ordered list instead of a common reversed list.

Fields and Methods:

public
constructor *Initialize...*
constructor *Resemble...*
constructor *Duplicate...*
function *CreateComposite...* virtual;
constructor *StreamLoad...*

Methods

CreateComposite *function tPriorityQueue.CreateComposite(NumberOfElements: word): pLinearADT; virtual;*

Creates an instance of the composite ADT.

Overrides tQueue.CreateComposite.

Overrides tCompositeADT.CreateComposite.

Duplicate *constructor tPriorityQueue.Duplicate(ADT: pADT);*

Initializes a duplicate ADT instance (with equal elements).

Overrides tQueue.Duplicate.

Initialize *constructor tPriorityQueue.Initialize(SizeOfElements: word);*

Initializes an instance of this ADT class.

Overrides tQueue.Initialize.

Overrides tCompositeADT.Initialize.

Overrides tADT.Initialize.

Overrides tObject.Initialize.

Resemble *constructor tPriorityQueue.Resemble(ADT: pADT);*

Initializes an ADT that resembles the specified ADT.

Overrides tQueue.Resemble.

StreamLoad *constructor tPriorityQueue.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

Overrides tQueue.StreamLoad.

Overrides tCompositeADT.StreamLoad.

Overrides tADT.StreamLoad.

Overrides tObject.StreamLoad.

EFTABLE.tProbeHashTable = object(tOpenHashTable)

This class defines an open hash table that uses linear probing to handle collisions. A hash table is a generalization of an array into an ordered ADT (see *tHashTable*).

This is an open hash table with linear probing. Collisions are resolved by walking forward in the table until an empty slot is found.

inserted at positions calculated by hash function (see *tHashPlug*). If there occur a collision (a calculated slot is used), the *Rehash* method is called until an empty slot is found. The double hash mechanism uses the collision index (the wrong hash code) to calculate a new hash code. Double hashing is generally more powerful than linear probing (see *tProbeHashTable*), but it is not as rapid as the bucket mechanism in *tBucketHashTable*.

Fields and Methods:

public
constructor *Initialize...*
function *Rehash...* virtual;
constructor *StreamLoad...*

Methods

Initialize

constructor tProbeHashTable.Initialize(CapacityOfElements, SizeOfElements: word);

Initializes an instance of this ADT class.

Overrides tHashTable.Initialize.

Overrides tCompositeADT.Initialize.

Overrides tADT.Initialize.

Overrides tObject.Initialize.

Rehash

function tProbeHashTable.Rehash(Index, Collisions: word): word; virtual;

Rehashes a hash code to resolve a collision.

Overrides tOpenHashTable.Rehash.

StreamLoad *constructor tProbeHashTable.StreamLoad(Stream: pStream);*

Loads an ADT from a stream.

Overrides tCompositeADT.StreamLoad.

Overrides tADT.StreamLoad.

Overrides tObject.StreamLoad.

tProcessPhase EFDEF

EFDEF.tProcessPhase = (Preprocess, FocusProcess, Postprocess);

tProfile EFSYSTEM

EFSYSTEM.tProfile = object(tNamedElement)

This class defines a profile, that is a registration hierarchy with arbitrary settings (elements), divided into sections or sub-profiles.

Settings must have a lifetime: they are either persistent or temporary. Persistent settings are static settings that affect how EFLIB looks and feels. Temporary settings are settings that control internal states in EFLIB, for example memory usage or number of open windows.

Both kinds of settings can be stored either as local constants or as internal instances owned by the profile.

Fields and Methods:

public
private
fSettings: *pLinearADT*;

Descendants:

tEnvironment
tSystemProfile

Fields

fSettings *tProfile.fSettings: pLinearADT;*

Settings in this profile (or other profiles).

tQueue EFDATA

EFDATA.tQueue = object(tCompositeADT)

This class implements a queue, that is, a specialized form of list that obey the first-in, first-out (FIFO) protocol. Elements are inserted in the back of the queue and removed from the front. Thus, an element removed from the the queue is the element that has been held by the queue for the longest amount of time.

Fields and Methods:

public
constructor *Initialize...*
constructor *Resemble...*
constructor *Duplicate...*
procedure *Enqueue...* virtual;
procedure *Dequeue...* virtual;
procedure *Skip*; virtual;
function *CreateComposite...* virtual;
constructor *StreamLoad...*

Descendants:

tCircularQueue
tPriorityQueue

Methods

CreateComposite *function tQueue.CreateComposite(NumberOfElements: word): pLinearADT; virtual;*

Creates an instance of the composite ADT.

Overrides tCompositeADT.CreateComposite.

Dequeue *procedure tQueue.Dequeue(var Data); virtual;*

Returns the last (oldest) element in the queue.

Duplicate *constructor tQueue.Duplicate(ADT: pADT);*

Initializes a duplicate ADT instance (with equal elements).

Enqueue *procedure tQueue.Enqueue(var Data); virtual;*

Inserts a new element in the queue.

Initialize *constructor tQueue.Initialize(SizeOfElements: word);*

Initializes an instance of this ADT class.

Overrides tCompositeADT.Initialize.

Overrides tADT.Initialize.

Overrides tObject.Initialize.

Resemble *constructor tQueue.Resemble(ADT: pADT);*

Initializes an ADT that resembles the specified ADT.

Skip *procedure tQueue.Skip; virtual;*

Erases the last (oldest) element in the queue.

StreamLoad *constructor tQueue.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

Overrides tCompositeADT.StreamLoad.

Overrides tADT.StreamLoad.

Overrides tObject.StreamLoad.

tQuickSortPlug EFADT

EFADT.tQuickSortPlug = object(tSortPlug)

Quick sort plug class derived from *tSortPlug*. This class provides the Quick sort algorithm. Quick sort is fast in most cases, but has a bad worst-case scenario (when elements are already in sorted order). Use the tMergeSortPlug if you require good performance even in the worst-case scenario (see *EFSORT*). Complexity: O(N * 2-log N) - O(N^2).

Fields and Methods:

public
constructor *Initialize...*
procedure *Sort...* virtual;

procedure *Partition...*
constructor *StreamLoad...*

Methods

Initialize

constructor tQuickSortPlug.Initialize(PlugManager: pPlugManager);

Constructs a plug instance for the specified manager.

Overrides tPlug.Initialize.

Overrides tObject.Initialize.

Partition

procedure tQuickSortPlug.Partition(Start, Stop: word; var Pivot: word; Order: tSortOrder);

Partitions a segment (used by the algorithm).

Sort

procedure tQuickSortPlug.Sort(Start, Stop: word; Order: tSortOrder); virtual;

Sorts elements within the specified interval.

Overrides tSortPlug.Sort.

StreamLoad

constructor tQuickSortPlug.StreamLoad(Stream: pStream);

Constructs and loads an instance from a stream.

Overrides tSortPlug.StreamLoad.

Overrides tPlug.StreamLoad.

Overrides tStaticElement.StreamLoad.

Overrides tElement.StreamLoad.

Overrides tObject.StreamLoad.

tRational EFMATH

EFMATH.tRational = object(tMathObject)

This class defines a rational number, ie. a mathematical object that can be written as P/Q, where P and Q are arbitrary integer numbers. All rational numbers are automatically simplified so that GCD (P,Q) = 1.

Fields and Methods:

public
constructor *Initialize...*
constructor *Duplicate...*
procedure *SetValue...*
procedure *SetReal...*
procedure *Add...* virtual;
procedure *Subtract...* virtual;
procedure *Multiply...* virtual;
procedure *Divide...* virtual;
constructor *StreamLoad...*
procedure *StreamStore...* virtual;
procedure *StreamWrite...* virtual;
function *Value...* virtual;
function *P...*
function *Q...*
function *IsZero...* virtual;
private
fP,
fQ: longint;
procedure *Normalize*; virtual;

Fields

fP *tRational.fP,*

fQ: longint;

Nominator and denominator for this rational number (P/Q).

fQ *See fP*

Methods

Add *procedure tRational.Add(What: pMathObject); virtual;*

Adds an object to this object.

Overrides tMathObject.Add.

Divide *procedure tRational.Divide(What: pMathObject); virtual;*

Divides this object by another object.

Overrides tMathObject.Divide.

Duplicate *constructor tRational.Duplicate(What: pMathObject);*

Initializes an duplicated type-casted mathematical object.

Initialize *constructor tRational.Initialize(nP, nQ: longint);*

Initializes an instance with the specified content.

Overrides tObject.Initialize.

IsZero *function tRational.IsZero: boolean; virtual;*

Is number equal to zero?

Overrides tMathObject.IsZero.

Multiply *procedure tRational.Multiply(What: pMathObject); virtual;*

Multiplies this object with another object.

Overrides tMathObject.Multiply.

Normalize *procedure tRational.Normalize; virtual;*

Normalizes P/Q so that GCD (P,Q) = 1.

P *function tRational.P: longint;*

Returns the nominator in P/Q.

Q *function tRational.Q: longint;*

Returns the denominator in P/Q.

SetReal *procedure tRational.SetReal(nX: tBaseReal);*

Sets the number to (approximate) real value.

SetValue *procedure tRational.SetValue(nP, nQ: longint);*

Sets the number.

StreamLoad *constructor tRational.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

Overrides tStaticElement.StreamLoad.

Overrides tElement.StreamLoad.

Overrides tObject.StreamLoad.

StreamStore *procedure tRational.StreamStore(Stream: pStream); virtual;*

Stores an instance to a stream.

Overrides tStaticElement.StreamStore.

Overrides tElement.StreamStore.

Overrides tObject.StreamStore.

StreamWrite *procedure tRational.StreamWrite(Stream: pStream); virtual;*

Writes this math object as formatted text, e.g. "2+2i".

Overrides tElement.StreamWrite.

Overrides tObject.StreamWrite.

Subtract *procedure tRational.Subtract(What: pMathObject); virtual;*

Subtracts an object from this object.

Overrides tMathObject.Subtract.

Value *function tRational.Value: tBaseReal; virtual;*

Returns the value of this real number.

tReal EFMATH

EFMATH.tReal = object(tMathObject)

This class defines a real number, ie. number with theoretically infinently many decimals. The default range of the tReal number is [-1.1e4932, 1.1e4932] with a precision of 19 digits. However, if the EXTENDEDREAL directive is disabled in FLAGS.INC, the range is reduced to [-1.7e38, 1.7e38].

Fields and Methods:

public
constructor *Initialize...*
constructor *Duplicate...*
procedure *SetValue...*
procedure *Add...* virtual;
procedure *Subtract...* virtual;
procedure *Multiply...* virtual;
procedure *Divide...* virtual;
constructor *StreamLoad...*
procedure *StreamStore...* virtual;
procedure *StreamWrite...* virtual;
function *Value...* virtual;
function *IsZero...* virtual;
private
fX: *tBaseReal*;

Fields

fX *tReal.fX: tBaseReal;*

Floating-point number.

Methods

Add *procedure tReal.Add(What: pMathObject); virtual;*

Adds an object to this object.

Overrides tMathObject.Add.

Divide *procedure tReal.Divide(What: pMathObject); virtual;*

Divides this object by another object.

Overrides tMathObject.Divide.

Duplicate *constructor tReal.Duplicate(What: pMathObject);*

Initializes an duplicated type-casted mathematical object.

Initialize *constructor tReal.Initialize(nX: tBaseReal);*

Initializes an instance with the specified content.

Overrides tObject.Initialize.

IsZero *function tReal.IsZero: boolean; virtual;*

Is number equal to zero?

Overrides tMathObject.IsZero.

Multiply *procedure tReal.Multiply(What: pMathObject); virtual;*

Multiplies this object with another object.

Overrides tMathObject.Multiply.

SetValue *procedure tReal.SetValue(nX: tBaseReal);*

Sets the number.

StreamLoad *constructor tReal.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

Overrides tStaticElement.StreamLoad.

Overrides tElement.StreamLoad.

Overrides tObject.StreamLoad.

StreamStore *procedure tReal.StreamStore(Stream: pStream); virtual;*
Stores an instance to a stream.
Overrides tStaticElement.StreamStore.
Overrides tElement.StreamStore.
Overrides tObject.StreamStore.

StreamWrite *procedure tReal.StreamWrite(Stream: pStream); virtual;*
Writes this math object as formatted text, e.g. "2+2i".
Overrides tElement.StreamWrite.
Overrides tObject.StreamWrite.

Subtract *procedure tReal.Subtract(What: pMathObject); virtual;*
Subtracts an object from this object.
Overrides tMathObject.Subtract.

Value *function tReal.Value: tBaseReal; virtual;*
Returns the value of this real number.

tRegister EFREG

EFREG.tRegister = object(tClassRegister)

This class defines a class register that uses an ordered ADT for registrations. See *tClassRegister* for details.

tRegister require all the registered classes to be descendants of *tElement*.

Fields and Methods:

public
procedure *Register...* virtual;
function *Release...* virtual;
function *ReleaseClass...* virtual;
function *First...* virtual;
function *Last...* virtual;
function *Instances...* virtual;
function *IsRegistered...* virtual;
function *IsClassRegistered...* virtual;
function *IsValid...* virtual;
private
fRegister: *pOrderedADT*;

Fields

fRegister *tRegister.fRegister: pOrderedADT;*

Methods

First *function tRegister.First: word; virtual;*

Returns the lowest registered identity (zero if empty).

Overrides tClassRegister.First.

Instances *function tRegister.Instances: word; virtual;*

Returns the number of registered instances.

Overrides tClassRegister.Instances.

IsClassRegistered *function tRegister.IsClassRegistered(Class: pointer): boolean; virtual;*

Is an instance of this class or a subclass registered?

Overrides tClassRegister.IsClassRegistered.

IsRegistered *function tRegister.IsRegistered(IdentityOfInstance: word): boolean; virtual;*

Is there an instance with the specified identity?

Overrides tClassRegister.IsRegistered.

IsValid *function tRegister.IsValid(What: pObject): boolean; virtual;*

Can this instance (class) be registered - is it valid?

Overrides tClassRegister.IsValid.

Last *function tRegister.Last: word; virtual;*

Returns the highest registered identity (zero if empty).

Overrides tClassRegister.Last.

Register *procedure tRegister.Register(What: pObject); virtual;*

Registers an instance, that is, puts it into the register.

Overrides tClassRegister.Register.

Release *function tRegister.Release(IdentityOfInstance: word): pObject; virtual;*

Releases the instance with this identity - or returns NIL.

Overrides tClassRegister.Release.

ReleaseClass *function tRegister.ReleaseClass(Class: pointer): pObject; virtual;*

Releases a descendant from this class - or returns NIL.

Overrides tClassRegister.ReleaseClass.

tReplaceFilter EFSEARCH

EFSEARCH.tReplaceFilter = object(tDelayedSearchFilter)

This filter replaces any occurrences of some text whenever data is written to the base stream. Reading is not filtered. The search data and the replacement data must be specified when the filter constructs.

Fields and Methods:

public
constructor *Initialize*...
destructor *Intercept*; virtual;
procedure *Match*; virtual;
private
fReplace: *pElement*;

Fields

fReplace *tReplaceFilter.fReplace: pElement;*

Replacement element to write when a match is found.

Methods

Initialize *constructor tReplaceFilter.Initialize(var Data; Length: word; var ReplaceWith; ReplaceLength: word; Stream: pStream);*

Initializes a replace filter for the specified data.

Overrides tSearchFilter.Initialize.

Overrides tMatchFilter.Initialize.

Overrides tFilter.Initialize.

Overrides tStream.Initialize.

Overrides tObject.Initialize.

Intercept *destructor tReplaceFilter.Intercept; virtual;*

Intercepts and destructs an instance of this type.

Overrides tSearchFilter.Intercept.

Overrides tMatchFilter.Intercept.

Overrides tFilter.Intercept.

Overrides tStream.Intercept.

Overrides tObject.Intercept.

Match *procedure tReplaceFilter.Match; virtual;*

Write the replacement data whenever a match is found.

Overrides tDelayedSearchFilter.Match.

tResource EFRES

EFRES.tResource = object(tFilter)

This class implements a resource handler. It handles an external collection of arbitrary classes. Resources have the following format:

<tString> Name of the resource entry. <longint> 32-bit checksum for the tObject instance. <tObject> An arbitrary tObject instance.

Resources are automatically compressed using the tCompressFilter class and certified with a CRC32 checksum.

Fields and Methods:

public
constructor *Initialize...*
procedure *PutEntry...* virtual;
function *GetEntry...* virtual;
function *IsIntact...* virtual;
function *IsInside...* virtual;

Descendants:

tArchive

Methods

GetEntry *function tResource.GetEntry(Name: string; var Entry: pObject): boolean; virtual;*

Gets an entry and returns TRUE for success.

Initialize *constructor tResource.Initialize(Filename: string);*

Constructs a resource handler for the specified stream.

Overrides tFilter.Initialize.

Overrides tStream.Initialize.

Overrides tObject.Initialize.

IsInside *function tResource.IsInside(Name: string): boolean; virtual;*

Returns TRUE if an entry with the specified name exists.

IsIntact *function tResource.IsIntact: boolean; virtual;*

Returns TRUE if this resource is intact (CRC check).

PutEntry *procedure tResource.PutEntry(Name: string; Instance: pObject); virtual;*

Puts a new instance into the resource.

tReversedKeyPlug EFADT

EFADT.tReversedKeyPlug = object(tKeyPlug)

Key plug class: this class provides a mechanism of assigning search keys to elements. This class reverses search keys and element comparisons, forcing an ADT to reverse its sort order.

tReversedKeyPlug forces an ADT to ignore *tElement.Compare* and instead use the Compare method in this class.

Compare reverses the result of an comparison. If an element is smaller than another, Compare pretends that it is bigger. Use *tReversedKeyPlug* to reverse sort orders, etc.

Fields and Methods:

public
constructor *Initialize...*
function *CompareElements...* virtual;
function *CompareKey...* virtual;
constructor *StreamLoad...*

Methods

CompareElements *function tReversedKeyPlug.CompareElements(A, B: pElement): shortint; virtual;*

Compares element A with element B and returns [-1, 0, 1].

Overrides tKeyPlug.CompareElements.

CompareKey *function tReversedKeyPlug.CompareKey(var A; B: pElement): shortint; virtual;*

Compares key A with element B and returns [-1, 0, 1].

Overrides tKeyPlug.CompareKey.

Initialize *constructor tReversedKeyPlug.Initialize(PlugManager: pPlugManager);*

Initializes a bounded search key for an ADTs plug manager.

Overrides tKeyPlug.Initialize.

Overrides tPlug.Initialize.

Overrides tObject.Initialize.

StreamLoad *constructor tReversedKeyPlug.StreamLoad(Stream: pStream);*

Constructs and loads an instance from a stream.

Overrides tKeyPlug.StreamLoad.

Overrides tPlug.StreamLoad.

Overrides tStaticElement.StreamLoad.

Overrides tElement.StreamLoad.

Overrides tObject.StreamLoad.

EFLIST.tReversedList = object(tList)

This class defines a reversed doubly linked list. Elements are inserted in reversed order, that is, Store and Put inserts the element before the first node.

Fields and Methods:

public
constructor *Initialize...*
constructor *Resemble...*
constructor *Duplicate...*
procedure *PutNode...* virtual;
constructor *StreamLoad...*

Descendants:

tAdjustedList
tOrderedList

Methods

Duplicate *constructor tReversedList.Duplicate(ADT: pADT);*

Initializes a duplicate ADT instance (with equal elements).

Overrides tList.Duplicate.

Initialize *constructor tReversedList.Initialize(SizeOfElements: word);*

Initializes an instance of this ADT class.

Overrides tList.Initialize.

Overrides tLinearADT.Initialize.

Overrides tADT.Initialize.

Overrides tObject.Initialize.

PutNode *procedure tReversedList.PutNode(Node: pLinkage); virtual;*

Stores a new node (container) in the list.

Overrides tList.PutNode.

Resemble *constructor tReversedList.Resemble(ADT: pADT);*

Initializes an ADT that resembles the specified ADT.

Overrides tList.Resemble.

StreamLoad *constructor tReversedList.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

Overrides tList.StreamLoad.

Overrides tADT.StreamLoad.

Overrides tObject.StreamLoad.

tScreen EFSCREEN

EFSCREEN.tScreen = object(tCRT)

Fields and Methods:

tSearchFilter EFSEARCH

EFSEARCH.tSearchFilter = object(tMatchFilter)

This class defines an efficient Knuth-Morris-Pratt (KMP) algorithm for searching in streams. See *tMatchFilter* for details.

Since the KMP algorithm require an auxilary (prefix) array, you should remember that very large search texts may require a lot of memory. The KMP algorithm is one of the fastest algorithms when it comes to sequential searches.

Fields and Methods:

public
constructor *Initialize...*
destructor *Intercept*; virtual;
function *FilterByte...* virtual;
function *IsAllocated...* virtual;
private
fArray: *pArray*;

Descendants:

tDelayedSearchFilter
tReplaceFilter

Fields

fArray *tSearchFilter.fArray: pArray;*

Auxiliary array with information about the search string.

Methods

FilterByte *function tSearchFilter.FilterByte(Data: byte): byte; virtual;*

Checks if the specified byte is a match.

Overrides tMatchFilter.FilterByte.

Initialize *constructor tSearchFilter.Initialize(var Data; Length: word; Stream: pStream);*

Initializes a search filter for the specified data.

Overrides tMatchFilter.Initialize.

Overrides tFilter.Initialize.

Overrides tStream.Initialize.

Overrides tObject.Initialize.

Intercept *destructor tSearchFilter.Intercept; virtual;*

Intercepts and destructs an instance of this type.

Overrides tMatchFilter.Intercept.

Overrides tFilter.Intercept.

Overrides tStream.Intercept.

Overrides tObject.Intercept.

IsAllocated *function tSearchFilter.IsAllocated: boolean; virtual;*

Is the stream allocated, ie. ready to be used in some way?

Overrides tMatchFilter.IsAllocated.

Overrides tFilter.IsAllocated.

Overrides tStream.IsAllocated.

tSearchMethod EFDEF

EFDEF.tSearchMethod = (LinearSearchMethod, BinarySearchMethod);

This type stores the search method for an ADT: either a linear search (through the entire structure) or a binary search that is much faster but require that elements are in sorted order.

tSequentialFilter EFFILTER

EFFILTER.tSequentialFilter = object(tFilter)

This class is an abstract filter. It overrides the method *Seek* to disable random-access, thus forcing all descendants only to provide sequential access.

Fields and Methods:

public
procedure *Seek*... virtual;

Descendants:

tCompressFilter
tCRC16Filter
  tCRC32Filter

Methods

Seek *procedure tSequentialFilter.Seek(Where: longint); virtual;*

Seeks, ie. moves to, a position in the stream [0 .. n].

Overrides tFilter.Seek.

Overrides tStream.Seek.

tServiceMessage EFCMAN

EFCMAN.tServiceMessage = object(tMessage)

Service message: this message contains one of the service identities (defined in *EFDEF*). These identities forces the receiver component to change one of its standard state flags (such as focusing, locking or protection).

Fields and Methods:

public
constructor *Initialize...*
function *Service...* virtual;
private
fService: word;

Fields

fService *tServiceMessage.fService: word;*

The identity of the requested service (see *EFDEF*).

Methods

Initialize *constructor tServiceMessage.Initialize(MessageReceiver, MessageSender: pComponent; ServiceIdentity: word);*

Initializes a message to the specified receiver and sender.

Overrides tMessage.Initialize.

Overrides tObject.Initialize.

Service *function tServiceMessage.Service: word; virtual;*

Returns the service identity number as defined in *EFDEF*.

tSet EFBASIC

EFBASIC.tSet = object(tObject)

Extended set class. This class operates similar to the Borland Pascal set type but uses a bit-level organization, hence being much more memory efficient. It also handles up to 524,280 elements.

Fields and Methods:

public
constructor *Initialize...*
constructor *Duplicate...*
destructor *Intercept*; virtual;

procedure *Clear*; virtual;
procedure *Copy...* virtual;
procedure *SetElement...*
function *Element...*
procedure *Union...*
procedure *Difference...*
procedure *Intersection...*
constructor *StreamLoad...*
procedure *StreamStore...* virtual;
function *IsSubset...*
function *Size...*
private
fBits: *tAllocation*;

Descendants:

tCharacters
EFSTYLES.tStyle

Fields

fBits *tSet.fBits: tAllocation;*

Bit allocation (0 = none, 1 = exist).

Methods

Clear *procedure tSet.Clear; virtual;*

Clears the set, ie. resets all elements to FALSE.

Copy *procedure tSet.Copy(What: pSet); virtual;*

Copies the specified set into this set..

Difference *procedure tSet.Difference(What: pSet);*

Makes this set the differnce of itself and another set.

Duplicate *constructor tSet.Duplicate(Instance: pSet);*

Initializes and duplicates an instance of this class.

Element *function tSet.Element(Index: longint): boolean;*

Returns the existance of this element (TRUE or FALSE).

Initialize *constructor tSet.Initialize(Elements: longint);*

Initializes and constructs an instance of this class.

Overrides tObject.Initialize.

Intercept *destructor tSet.Intercept; virtual;*

Intercepts and destructs this instance.

Overrides tObject.Intercept.

Intersection *procedure tSet.Intersection(What: pSet);*

Makes this set the intersection of itself and another set.

IsSubset *function tSet.IsSubset(What: pSet): boolean;*

Is the specified set a subset to this set (A [B).

SetElement *procedure tSet.SetElement(Index: longint; State: boolean);*

Sets the state of this element (existance).

Size *function tSet.Size: longint;*

Maximum element capacity (number of bits).

StreamLoad *constructor tSet.StreamLoad(Stream: pStream);*

Constructs and loads an instance from a stream.

Overrides tObject.StreamLoad.

StreamStore *procedure tSet.StreamStore(Stream: pStream); virtual;*

Stores an instance to a stream.

Overrides tObject.StreamStore.

Union *procedure tSet.Union(What: pSet);*

Makes this set the unary of itself and another set.

tSetting EFSYSTEM

EFSYSTEM.tSetting = object(tNamedElement)

This class defines an abstract setting for profile classes. A setting is a named element associated to some value. Settings can be read or written to streams using the *StreamRead* and *StreamWrite*.

Fields and Methods:

public
constructor *Initialize...*
destructor *Intercept*; virtual;
procedure *SetValue...* virtual;

procedure *EnableDependence*; virtual;
procedure *DisableDependence*; virtual;
procedure *EnablePersistence*; virtual;
procedure *DisablePersistence*; virtual;
procedure *Put*... virtual;
function *Get*... virtual;
constructor *StreamLoad*...
procedure *StreamStore*... virtual;
procedure *StreamRead*... virtual;
procedure *StreamWrite*... virtual;
function *Value*... virtual;
function *HasValue*... virtual;
function *IsDependent*... virtual;
function *IsPersistent*... virtual;
private
fValue: *pElement*;
fDependent: boolean;
fPersistent: boolean;

Descendants:

tComponentSelector

Fields

fDependent *tSetting.fDependent: boolean;*

Is the value dependent on the setting?

fPersistent *tSetting.fPersistent: boolean;*

Is the setting persistent (saved when program ends)?

fValue *tSetting.fValue: pElement;*

Value of this setting (text string).

Methods

DisableDependence *procedure tSetting.DisableDependence; virtual;*

Disables value dependence (value never destructed).

DisablePersistence *procedure tSetting.DisablePersistence; virtual;*

Disables persistence for status settings for run-time only.

EnableDependence *procedure tSetting.EnableDependence; virtual;*

Enables value dependence (value owned).

EnablePersistence *procedure tSetting.EnablePersistence; virtual;*

Enables persistence (saves the setting when program ends).

Get *function tSetting.Get: string; virtual;*

Gets the value of this setting (text string).

HasValue *function tSetting.HasValue: boolean; virtual;*

Has this setting a value element?

Initialize *constructor tSetting.Initialize(Name, Value: string);*

Initializes a setting with a name and value.

Overrides tNamedElement.Initialize.

Overrides tObject.Initialize.

Intercept *destructor tSetting.Intercept; virtual;*

Intercepts this setting and any owned value.

Overrides tElement.Intercept.

Overrides tObject.Intercept.

IsDependent *function tSetting.IsDependent: boolean; virtual;*

Returns TRUE if the setting owns its value.

IsPersistent *function tSetting.IsPersistent: boolean; virtual;*

Returns TRUE if the setting is persistent (saved).

Put *procedure tSetting.Put(Value: string); virtual;*

Puts a value into this setting (text string).

SetValue *procedure tSetting.SetValue(ValueOfSetting: pElement); virtual;*

Sets value element assigned to this setting.

StreamLoad *constructor tSetting.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

Overrides tNamedElement.StreamLoad.

Overrides tStaticElement.StreamLoad.

Overrides tElement.StreamLoad.

Overrides tObject.StreamLoad.

StreamRead *procedure tSetting.StreamRead(Stream: pStream); virtual;*

Reads this setting from a text stream.

Overrides tElement.StreamRead.

StreamStore *procedure tSetting.StreamStore(Stream: pStream); virtual;*

Stores an instance to a stream.

Overrides tNamedElement.StreamStore.

Overrides tStaticElement.StreamStore.

Overrides tElement.StreamStore.

Overrides tObject.StreamStore.

StreamWrite *procedure tSetting.StreamWrite(Stream: pStream); virtual;*

Writes this setting to a text stream.

Overrides tElement.StreamWrite.

Overrides tObject.StreamWrite.

Value *function tSetting.Value: pElement; virtual;*

Returns the associated value element or NIL if none.

tSetup EFKERNEL

EFKERNEL.tSetup = object(tObject)

This class contains the kernel setup, that is the system settings in EFLIB. Additional settings are added by units like *EFKEYBRD* and *EFMOUSE*.

Fields and Methods:

public
fIniUpdate: boolean;
fIdleTime: word;
fStreamBuffer: word;
fReportAllErrors: boolean;
fSound: boolean;
fRestoreStartup: boolean;
fNationalTable: string;
fGeneralError: string;
fUndefinedError: string;

fUnknownComponent: string;
fDefaultStringSize: word;
fStringPrecision: byte;
fHeapPrecision: byte;
fHeapMaximumSize: word;
fStartupMemory: longint;
fIniFile: *pStream*;
constructor *Initialize*;
procedure *SetDefaults*;
procedure *GetSetting...* virtual;
constructor *StreamLoad...*
procedure *StreamStore...* virtual;

Fields

fDefaultStringSize *tSetup.fDefaultStringSize: word;*

The size (in bytes) to assign to text strings by default.

fGeneralError *tSetup.fGeneralError: string;*

Message to display if a general kernel error occur.

fHeapMaximumSize *tSetup.fHeapMaximumSize: word;*

Maximum size of a single heap allocation in bytes. Borland Pascal limits the size of memory that can be safely allocated to 65528 bytes. EFLIB limits the size to 65520 bytes by default.

fHeapPrecision *tSetup.fHeapPrecision: byte;*

Precision in heap allocations for heap usage estimations. This precision is normally set to 16 bytes (word alignment), but can in some systems be changed to 8 or 32 bytes.

fIdleTime *tSetup.fIdleTime: word;*

Maximum idle time in milliseconds for components, that is the time before the application must allow other components to do their job.

fIniFile *tSetup.fIniFile: pStream;*

Initialization file (optional). This file is set by EFINIT when present.

fIniUpdate *tSetup.fIniUpdate: boolean;*

This setting enables or desables an automatic update of any INI file when the program terminates.

fNationalTable *tSetup.fNationalTable: string;*

National case translations, eg. ''. This string should contain pairs of characters, where the first in the pair always specify the lower-case representation and the second the upper-case representation.

fReportAllErrors *tSetup.fReportAllErrors: boolean;*

This setting can force EFLIB to terminate the program whenever an error is reported, even if that error is not classified as critical (fatal flag). Default is FALSE.

fRestoreStartup *tSetup.fRestoreStartup: boolean;*

Toggles restoring of the start-up screen and settings when your program terminates.

fSound *tSetup.fSound: boolean;*

Toggles system sounds. This setting should be TRUE if you want EFLIB to generate warning sounds.

fStartupMemory *tSetup.fStartupMemory: longint;*

Number of bytes memory available when EFLIB started.

fStreamBuffer *tSetup.fStreamBuffer: word;*

Maximum buffer size for temporary stream transfers. This setting affect the memory buffer for CopyIn and CopyOut operations in *tStream*, and some filter processes. Higher values may give you better performance.

fStringPrecision *tSetup.fStringPrecision: byte;*

Precision in string allocations (sizes must be multiples of this value). Higher precision values may result in fewer reallocation due to insuffcient space in strings, and may thus increase speed. See tString.

fUndefinedError *tSetup.fUndefinedError: string;*

Standard message that describes an undefined error (an error that does not have an associated message but must be displayed).

fUnknownComponent *tSetup.fUnknownComponent: string;*

Message to display if a components name is unknown and an error occur.

Methods

GetSetting

procedure tSetup.GetSetting(Name: string; var Data); virtual;

Gets the value of a certain setting.

Initialize

constructor tSetup.Initialize;

Initializes a basic system setup instance with defaults.

Overrides tObject.Initialize.

SetDefaults

procedure tSetup.SetDefaults;

Sets the default system settings.

StreamLoad

constructor tSetup.StreamLoad(Stream: pStream);

Loads an instance from a stream.

Overrides tObject.StreamLoad.

StreamStore

procedure tSetup.StreamStore(Stream: pStream); virtual;

Stores an instance to a stream.

Overrides tObject.StreamStore.

tSolver EFMATH

EFMATH.tSolver = object(tObject)

Generic numerical solver class. This class provides a set of common numerical methods: Newton-Raphson's method for solving f(x) = 0, Simpson's formula for integral calculus, derivate approximation, Euler's and Heun's methods for solving y' = f(x,y). The results are automatically optimized using Richardson's extrapolation where possible. This class is designed to plug-in with the tEvaluator class.

Fields and Methods:

public
constructor *Initialize...*
function *F...* virtual;
function *G...* virtual;
function *H...* virtual;
function *Newton...*
procedure *Simpson...*

procedure *Euler...*
procedure *Heun...*
function *Richardson...*
function *Result...*
function *IsFinished...* virtual;
private
fIterations: longint;
fMaximum: longint;
fPrecision: *tBaseReal*;
fCurrent: *tBaseReal*;
fPrevious: *tBaseReal*;
fExternalF: *pExternalF*;
fExternalG: *pExternalF*;
fExternalH: *pExternalH*;

Fields

fCurrent *tSolver.fCurrent: tBaseReal;*

Current value (in this iteration).

fExternalF *tSolver.fExternalF: pExternalF;*

External f(x) function (optional).

fExternalG *tSolver.fExternalG: pExternalF;*

External f(x) function (optional).

fExternalH *tSolver.fExternalH: pExternalH;*

External f(x) function (optional).

fIterations *tSolver.fIterations: longint;*

Iteration counter (1 .. n).

fMaximum *tSolver.fMaximum: longint;*

Maximum number of iterations before the solver halts.

fPrecision *tSolver.fPrecision: tBaseReal;*

User-defined precision in answer.

fPrevious *tSolver.fPrevious: tBaseReal;*

Previous value (in this iteration).

Methods

Euler *procedure tSolver.Euler(X: tBaseReal);*

Differntial equation solving with Euler's step method.

F *function tSolver.F(X: tBaseReal): tBaseReal; virtual;*

The first function for the iteration.

G *function tSolver.G(X: tBaseReal): tBaseReal; virtual;*

The second function for the iteration (optional).

H *function tSolver.H(X, Y: tBaseReal): tBaseReal; virtual;*

The third function for the iteration (optional).

Heun *procedure tSolver.Heun(X: tBaseReal);*

Differntial equation solving with Heun's method.

Initialize *constructor tSolver.Initialize(Start, Precision: tBaseReal; Maximum: longint; ExternalF, ExternalG: pExternalF; ExternalH: pExternalH);*

Initializes an instance with the specified property.

Overrides tObject.Initialize.

IsFinished *function tSolver.IsFinished: boolean; virtual;*

Has the iteration finished, ie. desired precision reached.

Newton *function tSolver.Newton: boolean;*

Solve f(x)=0 with Newton-Raphson's method.

Result *function tSolver.Result: tBaseReal;*

Returns the result from the numerical method.

Richardson *function tSolver.Richardson(Y1, Y2: tBaseReal; P: byte): tBaseReal;*

Optimizes approximations using Richardson extrapolation.

Simpson *procedure tSolver.Simpson(Lower, Upper: tBaseReal);*

Integrale approximation with Simpson's formula.

tSortMethod EFDEF

EFDEF.tSortMethod = (MergeSortAlgorithm, QuickSortAlgorithm);

This type stores the sort method for an ADT. The sort method can be either Mergesort or Quicksort, the latter faster in most cases. The worst case scenario is better with Mergesort.

tSortOrder EFDEF

EFDEF.tSortOrder = (AscendingOrder, DescendingOrder, UnsortedOrder);

This type stores the sort order of the elements in an ADT. The elements can be sorted in ascending order, descending order or not sorted at all.

tSortPlug EFADT

EFADT.tSortPlug = object(tADTPlug)

Abstract sort plug class. Instances of this class can be connected to linear data structures (*tLinearADT*). They provide a mechanism of sorting elements. You may wish to create sort plugs for your own sorting algorithms.

Fields and Methods:

public
procedure *Install...* virtual;
procedure *Sort...* virtual;
constructor *StreamLoad...*
function *TypeOfPlug...* virtual;

Descendants:

tQuickSortPlug
EFSORT.tBubbleSortPlug
EFSORT.tInsertionSortPlug
EFSORT.tMergeSortPlug

Methods

Install *procedure tSortPlug.Install(PlugManager: pPlugManager); virtual;*

Installs the plug in the specified plug manager.

Overrides tADTPlug.Install.

Overrides tPlug.Install.

Sort *procedure tSortPlug.Sort(Start, Stop: word; Order: tSortOrder); virtual;*

Sorts elements within the specified interval.

StreamLoad *constructor tSortPlug.StreamLoad(Stream: pStream);*

Constructs and loads an instance from a stream.

Overrides tPlug.StreamLoad.

Overrides tStaticElement.StreamLoad.

Overrides tElement.StreamLoad.

Overrides tObject.StreamLoad.

TypeOfPlug *function tSortPlug.TypeOfPlug: string; virtual;*

The classification of this plug, e.g. "tSortPlug".

Overrides tADTPlug.TypeOfPlug.

Overrides tPlug.TypeOfPlug.

tSoundDevice EFDEVICE

EFDEVICE.tSoundDevice = object(tDevice)

Device object that handles the internal speaker.

Fields and Methods:

public
procedure *SetVolume...* virtual;
procedure *Sound...* virtual;
procedure *NoSound*; virtual;
procedure *Beep*;
procedure *SoundBell*;
procedure *SoundEffect...*

Methods

Beep *procedure tSoundDevice.Beep;*

Generates beep sound.

NoSound *procedure tSoundDevice.NoSound; virtual;*

Disables all sounds.

SetVolume *procedure tSoundDevice.SetVolume(Level: byte); virtual;*

Sets the sound output volume.

Sound *procedure tSoundDevice.Sound(Frequence: word); virtual;*

Generates a sound of given frequence.

SoundBell *procedure tSoundDevice.SoundBell;*

Generates a bell sound.

SoundEffect *procedure tSoundDevice.SoundEffect(Start, Stop, FrequenceMultiplier, SoundDelay: word);*

Generates a sound effect.

tSparceIterator EFARRAY

EFARRAY.tSparceIterator = object(tLinearIterator)

This class defines a sparce iterator, that is an iterator that only proceses elements that are used. Elements that are NIL are automatically skipped.

Fields and Methods:

constructor *Initialize...*
procedure *First*; virtual;
procedure *Last*; virtual;
procedure *WalkForward*; virtual;
procedure *WalkBackward*; virtual;
constructor *StreamLoad...*
function *IsEnd...* virtual;

Methods

First

procedure tSparceIterator.First; virtual;

Walks to the first element in this ADT.

Overrides tLinearIterator.First.

Overrides tIterator.First.

Initialize

constructor tSparceIterator.Initialize(ADT: pLinearADT);

Initializes an iterator for the specified ADT.

Overrides tLinearIterator.Initialize.

Overrides tObject.Initialize.

IsEnd

function tSparceIterator.IsEnd: boolean; virtual;

Is iterator at boundaries (beginning or end)?

Overrides tLinearIterator.IsEnd.

Overrides tIterator.IsEnd.

Last

procedure tSparceIterator.Last; virtual;

Walks to the last element in this ADT.

Overrides tLinearIterator.Last.

Overrides tIterator.Last.

StreamLoad

constructor tSparceIterator.StreamLoad(Stream: pStream);

Constructs and loads an instance from a stream.

Overrides tLinearIterator.StreamLoad.

Overrides tObject.StreamLoad.

WalkBackward

procedure tSparceIterator.WalkBackward; virtual;

Walk backwards (to the predecessor element).

Overrides tLinearIterator.WalkBackward.

Overrides tIterator.WalkBackward.

WalkForward

procedure tSparceIterator.WalkForward; virtual;

Walk forwards (to the successor element).

Overrides tLinearIterator.WalkForward.

Overrides tIterator.WalkForward.

EFTREE.tSplayTree = object(tBinaryTree)

This class defines a self-organizing binary tree known as a splay tree. The splay tree automatically balances its nodes according to the frequency in which they are accessed. Thus, nodes that are often accessed are moved closer to the root node, and elements that are rarely retrieved are kept in the bottom leaves. Splay trees does not change the element order.

Fields and Methods:

public
procedure *Touch...* virtual;

Methods

Touch *procedure tSplayTree.Touch(Node: pTreeLinkage); virtual;*

Touches the specified node after it have been accessed.

tStack EFDATA

EFDATA.tStack = object(tCompositeADT)

This class implements a stack, that is, a specialized form of list that obey the last-in, first-out (LIFO) protocol. Elements can be added (pushed into the stack) and removed (popped from the stack) only from the front of the stack. Thus, an element removed from a stack is the element that has been held for the least amount of time.

Fields and Methods:

public
constructor *Initialize...*
constructor *Resemble...*
constructor *Duplicate...*
procedure *Push...*
procedure *Pop...*
procedure *Top...*
procedure *Skip*; virtual;

constructor *StreamLoad...*

Methods

Duplicate *constructor tStack.Duplicate(ADT: pADT);*

Initializes a duplicate ADT instance (with equal elements).

Initialize *constructor tStack.Initialize(SizeOfElements: word);*

Initializes an instance of this ADT class.

Overrides tCompositeADT.Initialize.

Overrides tADT.Initialize.

Overrides tObject.Initialize.

Pop *procedure tStack.Pop(var Data);*

Pops an element from a non-empty stack (topmost element).

Push *procedure tStack.Push(var Data);*

Push a new element into the stack.

Resemble *constructor tStack.Resemble(ADT: pADT);*

Initializes an ADT that resembles the specified ADT.

Skip *procedure tStack.Skip; virtual;*

Erases the topmost element in the stack.

StreamLoad *constructor tStack.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

Overrides tCompositeADT.StreamLoad.

Overrides tADT.StreamLoad.

Overrides tObject.StreamLoad.

Top *procedure tStack.Top(var Data);*

Returns the topmost element in the stack.

EFSTREAM.tStandardStream = object(tStream)

This class defines a stream that operates on the standard input and output device (console and keyboard if not redirceted). The instance *StdIO* in EFSTREAM provides access to this stream. You can replace *StdIO* with your own instance to redirect I/O.

Fields and Methods:

public
constructor *Initialize...*
procedure *Read...* virtual;
procedure *Write...* virtual;
function *GetLine...* virtual;
private
fEcho: boolean;

Fields

fEcho *tStandardStream.fEcho: boolean;*

Shall character be written after they have been read?

Methods

GetLine *function tStandardStream.GetLine: string; virtual;*

Gets a text line from the stream (read until CR/LF).

Overrides tStream.GetLine.

Initialize *constructor tStandardStream.Initialize(Echo: boolean);*

Initializes a stream that operates acts a null device .

Overrides tStream.Initialize.

Overrides tObject.Initialize.

Read *procedure tStandardStream.Read(var Data; Count: word); virtual;*

Reads data from stream into a variable.

Overrides tStream.Read.

Write *procedure tStandardStream.Write(var Data; Count: word); virtual;*

Writes data to stream from a variable.

Overrides tStream.Write.

tStaticElement EFELEM

EFELEM.tStaticElement = object(tElement)

This is an abstract class that defines the properties of a static element. A static element is a *tElement* with contents that cannot be updated or swapped. A static element is an element that is fully defined by its class belonging, that is when you know the class of the element, you also know exactly what kind of element it is, what it contain, etc.

Definition of static elements:

(1) Static elements cannot be updated or swapped.

(2) Static elements are themself some kind of component.

(3) Static elements are only compatible with other static elements of precisely the same class.

(4) Static elements are normally stored in streams only as class identities - the property of a static element is fully defined by its class belonging.

However, a static element may have some properties that are not static: a property that maps the static element to some other classes, etc. See tNamedElement and *tIdentityElement*.

Fields and Methods:

public
constructor *StreamLoad...*
procedure *StreamStore...* virtual;
function *IsStatic...* virtual;
function *IsCompatible...* virtual;

Descendants:

EFCMAN.tMessage
  EFCMAN.tComponent
    EFCMAN.tManager
      EFAPP.tApplication
      EFCMAN.tCommunicator

    - *EFCONDRV.tConsole*
      - *EFCONDRV.tBoundedConsole*
        - *EFCONDRV.tCRT*
          - *EFSCREEN.tScreen*
        - *EFCONDRV.tVirtualConsole*
          - *EFCONDRV.tImage*
          - *EFSCREEN.tBufferedScreen*
    - *EFDEVICE.tDevice*
      - *EFDEVICE.tSoundDevice*
      - *EFMOUSE.tMouseDevice*
    - *EFVIEWS.tView*
      - *EFVIEWS.tGroup*
  - *EFCMAN.tServiceMessage*
  - *EFDEVICE.tInstallMessage*
  - *EFKEYBRD.tKeyEvent*
  - *EFMOUSE.tMouseEvent*
  - *EFMSG.tFocusedEvent*
  - *EFMSG.tLocalMessage*
- *tIdentityElement*
- *EFMATH.tMathObject*
  - *EFMATH.tComplex*
  - *EFMATH.tInteger*
  - *EFMATH.tMatrix*
    - *EFMATH.tPolynomial*
  - *EFMATH.tRational*
  - *EFMATH.tReal*
  - *EFMATH.tVector*
  - *EFMEXP.tExpression*
  - *EFMEXP.tMathFunction*
- *EFPLUG.tPlug*
  - *EFADT.tADTPlug*
    - *EFADT.tKeyPlug*
      - *EFADT.tBoundedKeyPlug*
      - *EFADT.tReversedKeyPlug*
    - *EFADT.tSortPlug*
      - *EFADT.tQuickSortPlug*
      - *EFSORT.tBubbleSortPlug*
      - *EFSORT.tInsertionSortPlug*
      - *EFSORT.tMergeSortPlug*
    - *EFTABLE.tHashPlug*
      - *EFTABLE.tTextHashPlug*
    - *EFTREE.tPathPlug*
- *EFSYSTEM.tNamedElement*
  - *EFSYSTEM.tFlag*

EFSYSTEM.tMapping
EFSYSTEM.tProfile
EFSYSTEM.tEnvironment
EFSYSTEM.tSystemProfile
EFSYSTEM.tSetting
EFSYSTEM.tComponentSelector
EFTABLE.tBucket
EFTIME.tTiming
EFTIME.tDate
EFTIME.tTimeDate
EFTIME.tTime

Methods

IsCompatible *function tStaticElement.IsCompatible(Instance: pObject): boolean; virtual;*

Are these elements compatible?

Overrides tElement.IsCompatible.

Overrides tObject.IsCompatible.

IsStatic *function tStaticElement.IsStatic: boolean; virtual;*

Overrides tElement.IsStatic.

StreamLoad *constructor tStaticElement.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

Overrides tElement.StreamLoad.

Overrides tObject.StreamLoad.

StreamStore *procedure tStaticElement.StreamStore(Stream: pStream); virtual;*

Stores an instance to a stream.

Overrides tElement.StreamStore.

Overrides tObject.StreamStore.

tStream EFKERNEL

EFKERNEL.tStream = object(tObject)

tStream is a general abstract class providing polymorphic I/O to and from a storage device. Streams provide both sequential and random access, quite similar to the Pascal file handling.

You can create your own descendant stream by overriding the virtual methods *Read* and *Write*. EFLIB itself does this to derive tBufferedStream, tFile and many other streams in *EFSTREAM*.

Fields and Methods:

public
fIndex: longint;
fLastTransfer: longint;
fMode: *tAccess*;
constructor *Initialize*;
destructor *Intercept*; virtual;
procedure *SetMode*... virtual;
procedure *Read*... virtual;
procedure *Write*... virtual;
procedure *Put*... virtual;
function *Get*... virtual;
procedure *PutString*... virtual;
function *GetString*... virtual;
procedure *PutLine*... virtual;
function *GetLine*... virtual;
procedure *PutByte*... virtual;
function *GetByte*... virtual;
procedure *CopyIn*... virtual;
procedure *CopyOut*... virtual;
procedure *Reset*; virtual;
procedure *Flush*; virtual;
procedure *Truncate*; virtual;
procedure *Seek*... virtual;
function *Mode*... virtual;
function *Size*... virtual;
function *Position*... virtual;
function *LastTransfer*... virtual;
function *IsAllocated*... virtual;
function *IsReadOnly*... virtual;
function *IsWriteOnly*... virtual;
function *IsEmpty*... virtual;
function *IsEnd*... virtual;
function *IsValid*... virtual;
function *IsEqual*... virtual;
function *IsEqualSize*...
function *IsCompatible*... virtual;

Descendants:

EFADT.tADTStream
EFFILTER.tFilter
  EFCONV.tConverter
  EFFILTER.tBufferFilter
    EFFILE.tFile
      EFFILE.tIFile
      EFFILE.tOFile
      EFFILE.tTemporaryFile
      EFRES.tStreamableFile
  EFFILTER.tDuplicateFilter
  EFFILTER.tEncryptFilter
  EFFILTER.tSequentialFilter
    EFFILTER.tCompressFilter
    EFFILTER.tCRC16Filter
      EFFILTER.tCRC32Filter
  EFFILTER.tTextFilter
  EFRES.tResource
    EFRES.tArchive
  EFSEARCH.tMatchFilter
    EFSEARCH.tSearchFilter
      EFSEARCH.tDelayedSearchFilter
        EFSEARCH.tReplaceFilter
  EFSYSTEM.tSwapFilter
EFMEMORY.tMemoryStream
EFSTREAM.tDataStream
EFSTREAM.tFileStream
EFSTREAM.tNullStream
EFSTREAM.tStandardStream

Fields

fIndex *tStream.fIndex: longint;*

Current position in the stream within [0, Size].

fLastTransfer *tStream.fLastTransfer: longint;*

Size of last transfer in bytes.

fMode *tStream.fMode: tAccess;*

Access mode (read only, write only or full).

Methods

CopyIn *procedure tStream.CopyIn(Stream: pStream; Count: longint); virtual;*

Copies data from this stream to another stream.

CopyOut *procedure tStream.CopyOut(Stream: pStream; Count: longint); virtual;*

Copies data into this stream from another stream.

Flush *procedure tStream.Flush; virtual;*

Flushes any buffer that is associated to this stream.

Get *function tStream.Get: pObject; virtual;*

Constructs the instance and loads data members.

GetByte *function tStream.GetByte: byte; virtual;*

Gets a byte from this stream.

GetLine *function tStream.GetLine: string; virtual;*

Gets a text line from the stream (read until CR/LF).

GetString *function tStream.GetString(Delimeter: string): string; virtual;*

Gets a text string from the stream, separated by delimeter.

Initialize *constructor tStream.Initialize;*

Initializes and constructs an instance of this type.

Overrides tObject.Initialize.

Intercept *destructor tStream.Intercept; virtual;*

Intercepts and destructs an instance of this type.

Overrides tObject.Intercept.

IsAllocated *function tStream.IsAllocated: boolean; virtual;*

Is the stream allocated, ie. ready to be used in some way?

IsCompatible *function tStream.IsCompatible(Instance: pObject): boolean; virtual;*

Are objects compatible, ie. can they replace each other?

Overrides tObject.IsCompatible.

IsEmpty *function tStream.IsEmpty: boolean; virtual;*

Is the stream empty, ie. has it′s size equal to zero.

IsEnd *function tStream.IsEnd: boolean; virtual;*

Is the current position at the end of stream?

IsEqual *function tStream.IsEqual(Instance: pObject): boolean; virtual;*

Are contents of these instances equal?

Overrides tObject.IsEqual.

IsEqualSize *function tStream.IsEqualSize(Instance: pStream): boolean;*

Is the specified stream of the same length as this stream?

IsReadOnly *function tStream.IsReadOnly: boolean; virtual;*

Is only reading allowed?

IsValid *function tStream.IsValid(Where: longint): boolean; virtual;*

Is the specified position valid for this stream?

IsWriteOnly *function tStream.IsWriteOnly: boolean; virtual;*

Is only writing allowed?

LastTransfer *function tStream.LastTransfer: longint; virtual;*

Size of last transfer, ie. number of bytes last transfered.

Mode *function tStream.Mode: tAccess; virtual;*

Returns the access mode for this stream.

Position *function tStream.Position: longint; virtual;*

Returns current position in this stream.

Put *procedure tStream.Put(Instance: pObject); virtual;*

Stores the instance at the current position.

PutByte *procedure tStream.PutByte(Data: byte); virtual;*

Puts a byte into this stream.

PutLine *procedure tStream.PutLine(Data: string); virtual;*

Puts a text line into the stream (with CR/LF).

PutString *procedure tStream.PutString(Data: string); virtual;*

Puts a text string into the stream without CR/LF.

Read *procedure tStream.Read(var Data; Count: word); virtual;*

Reads specified amount of data from this stream.

Reset *procedure tStream.Reset; virtual;*

Resets stream and moves to the first position.

Seek *procedure tStream.Seek(Where: longint); virtual;*

Seeks, ie. moves to, a position in the stream [0 .. n].

SetMode *procedure tStream.SetMode(Access: tAccess); virtual;*

Sets access mode, ie. the way the stream can be accessed.

Size *function tStream.Size: longint; virtual;*

Returns the streams size (length) in bytes.

Truncate *procedure tStream.Truncate; virtual;*

Truncates the stream, ie. deletes remaining data.

Write *procedure tStream.Write(var Data; Count: word); virtual;*

Writes specified data to this stream.

tStreamableFile EFRES

EFRES.tStreamableFile = object(tFile)

This class defines a streamable file, that is a file that can be stored in a stream. When the file is restored from the stream, it is automatically created in the current directory. This class is used by *tResource* to store external files in compressed archives.

tStreamableFile restores filename with optional path, attributes and file date and time.

Fields and Methods:

public
private
fName: *pString*;
fTime: *pTime*;
fDate: pDate;
fAttribute: byte;

Fields

fAttribute *tStreamableFile.fAttribute: byte;*

File attributes such as read-only marking.

fDate *tStreamableFile.fDate: pDate;*

Date for last modification.

fName *tStreamableFile.fName: pString;*

Name of this entry (maximum 65,535 characters).

fTime *tStreamableFile.fTime: pTime;*

Time for last modification.

tStreamElement EFELEM

EFELEM.tStreamElement = object(tElement)

This class handles a stream element for data structures. Stream elements leave the content control to external stream(s). A stream element knows where in the stream the element data begins and ends (offset and size of the data), and also knows which stream the data is stored in. When a stream element is stored to a stream, it is converted to a tGenericElement instance, that is, when it is retrieved it will actually be a tGenericElement instance.

Fields and Methods:

public
constructor *Initialize...*
constructor *InitializeEmpty*;
procedure *CopyIn...* virtual;
procedure *CopyOut...* virtual;
function *Data...* virtual;
function *Size...* virtual;
procedure *Dispose*; virtual;
constructor *StreamLoad...*
procedure *StreamStore...* virtual;
function *Compare...* virtual;
function *IsAllocated...* virtual;
private
fStream: *pStream*;

fOffset: longint;
fSize: word;

Fields

fOffset *tStreamElement.fOffset: longint;*

Offset of this element (position in stream).

fSize *tStreamElement.fSize: word;*

Size of this element in bytes.

fStream *tStreamElement.fStream: pStream;*

Pointer to an arbitrary stream instance (element contents).

Methods

Compare *function tStreamElement.Compare(What: pElement): shortint; virtual;*

Compares elements and returns [1, 0, -1] as result.

Overrides tElement.Compare.

CopyIn *procedure tStreamElement.CopyIn(Source: pointer; Count, Position: word); virtual;*

Copies the specified contents into this element.

Overrides tElement.CopyIn.

CopyOut *procedure tStreamElement.CopyOut(Target: pointer; Count, Position: word); virtual;*

Copies (parts of the) contents to some external address.

Overrides tElement.CopyOut.

Data *function tStreamElement.Data(Position: word): pointer; virtual;*

Returns a pointer to contents at some position.

Overrides tElement.Data.

Dispose *procedure tStreamElement.Dispose; virtual;*

Disposes any data allocation or erases the contents.

Overrides tElement.Dispose.

Initialize *constructor tStreamElement.Initialize(Stream: pStream; Offset: longint; SizeOfData: word);*

Initializes an element with the specified contents.

Overrides tObject.Initialize.

InitializeEmpty *constructor tStreamElement.InitializeEmpty;*

Initializes an empty element (not allocated).

IsAllocated *function tStreamElement.IsAllocated: boolean; virtual;*

Is this element allocated, ie. has some contents?

Overrides tElement.IsAllocated.

Size *function tStreamElement.Size: word; virtual;*

Returns the size of the contents of this element in bytes.

Overrides tElement.Size.

StreamLoad *constructor tStreamElement.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

Overrides tElement.StreamLoad.

Overrides tObject.StreamLoad.

StreamStore *procedure tStreamElement.StreamStore(Stream: pStream); virtual;*

Stores an instance to a stream.

Overrides tElement.StreamStore.

Overrides tObject.StreamStore.

tStreamHandle EFMEMORY

EFMEMORY.tStreamHandle = object(tElement)

This class handles a stream reference or an ownership of some data block inside an arbitrary stream.

Fields and Methods:

private
fIndex: longint;
fSize: longint;
fOwner: *pHandle*;

Fields

fIndex *tStreamHandle.fIndex: longint;*

fOwner *tStreamHandle.fOwner: pHandle;*

fSize *tStreamHandle.fSize: longint;*

tString EFSTRING

EFSTRING.tString = object(tGenericElement)

This class defines a null-terminated text string and complete set of string operations. The tString class differ from the built in Pascal string convention in several ways:

(1) tStrings are dynamically allocated and hence memory efficient.

(2) tStrings can be up to 65,534 character long.

(3) tString can contain end of line sequences.

(4) tString is derived from tElement, thus being an element that you can store in any of EFLIB′s data structures.

(5) tStrings have an extended set of string operation.

Fields and Methods:

public
constructor *Initialize...*
constructor *InitializeEmpty*;
constructor *Duplicate...*
destructor *Intercept*; virtual;
procedure *Assign...* virtual;
procedure *AssignPascal...* virtual;
procedure *Append...* virtual;
procedure *AppendPascal...* virtual;
procedure *AppendSection...* virtual;
procedure *CopyString...* virtual;
procedure *CopyPascal...* virtual;
procedure *Insert...* virtual;
procedure *InsertPascal...* virtual;
procedure *Delete...* virtual;
procedure *Truncate...* virtual;
procedure *Linefeed*; virtual;

procedure *LowerCase*; virtual;
procedure *UpperCase*; virtual;
function *Size...* virtual;
function *Capacity...* virtual;
function *Pascal...* virtual;
function *Data...* virtual;
function *Text...* virtual;
function *Allocate...* virtual;
procedure *Dispose*; virtual;
procedure *Resize...* virtual;
procedure *Adjust...* virtual;
procedure *Minimize*; virtual;
function *CreateString...* virtual;
function *CreateMatcher...* virtual;
constructor *StreamLoad...*
procedure *StreamStore...* virtual;
procedure *StreamRead...* virtual;
procedure *StreamWrite...* virtual;
function *Locate...* virtual;
function *Search...* virtual;
function *SearchPascal...* virtual;
function *SearchForward...* virtual;
function *SearchBackward...* virtual;
function *SearchClosest...* virtual;
function *SearchOccurrence...* virtual;
function *Occurrence...* virtual;
procedure *Replace...* virtual;
procedure *ReplacePascal...* virtual;
procedure *Fill...* virtual;
procedure *FillChar...* virtual;
procedure *Space...* virtual;
procedure *Strip...* virtual;
procedure *DeleteLeading...* virtual;
procedure *DeleteTrailing...* virtual;
function *Successor...* virtual;
function *IsToken...* virtual;
function *IsInside...* virtual;
function *IsCharactersInside...* virtual;

Descendants:

EFPATRN.tPatternString
tToken

Methods

Adjust *procedure tString.Adjust(AssertLength: word); virtual;*

Asserts that the string has room for the text (adjusts).

Allocate *function tString.Allocate(SizeOfData: word): boolean; virtual;*

Attempts to allocate memory for element contents.

Overrides tGenericElement.Allocate.

Overrides tElement.Allocate.

Append *procedure tString.Append(What: pString); virtual;*

Appends the specified string to this string.

AppendPascal *procedure tString.AppendPascal(What: string); virtual;*

Appends the specified Pascal string to this string.

AppendSection *procedure tString.AppendSection(What: pString; Start: word; Count: word); virtual;*

Appends a section of anothjer string to this string.

Assign *procedure tString.Assign(What: pString); virtual;*

Copies the specified string into this string.

AssignPascal *procedure tString.AssignPascal(What: string); virtual;*

Copies the specified Pascal string into this string.

Capacity *function tString.Capacity: word; virtual;*

Returns the allocation size in bytes (maximum length).

CopyPascal *procedure tString.CopyPascal(What: string; Start: word; Count: word); virtual;*

Returns a substring of a string (0 = first).

CopyString *procedure tString.CopyString(What: pString; Start: word; Count: word); virtual;*

Returns a substring of a string (0 = first).

CreateMatcher *function tString.CreateMatcher(Match: pString): pStringMatcher; virtual;*

Creates an instance of the string matcher class.

CreateString *function tString.CreateString: pString; virtual;*

Creates an instance of the string class (tString).

Data *function tString.Data(Position: word): pointer; virtual;*

Returns a pointer to contents at some position.

Overrides tGenericElement.Data.

Overrides tElement.Data.

Delete *procedure tString.Delete(Where: word; Count: word); virtual;*

Deletes a substring from a string (0 = first).

DeleteLeading *procedure tString.DeleteLeading(What: pString); virtual;*

Deletes all leading occurrences of a substring.

DeleteTrailing *procedure tString.DeleteTrailing(What: pString); virtual;*

Deletes all trailing occurrences of a substring.

Dispose *procedure tString.Dispose; virtual;*

Disposes any data allocation or erases the contents.

Overrides tGenericElement.Dispose.

Overrides tElement.Dispose.

Duplicate *constructor tString.Duplicate(What: pString);*

Initializes an duplicated string instance.

Overrides tGenericElement.Duplicate.

Fill *procedure tString.Fill(What: pString; Copies: word); virtual;*

Erases and fills the string with the a repeated substring.

FillChar *procedure tString.FillChar(Character: char; Copies: word); virtual;*

Fills the string with repeated copies of a character.

Initialize *constructor tString.Initialize(PascalString: string);*

Initializes a string with the specified text content.

Overrides tGenericElement.Initialize.

Overrides tObject.Initialize.

InitializeEmpty *constructor tString.InitializeEmpty;*

Initializes a string without text content.

Overrides tGenericElement.InitializeEmpty.

Insert *procedure tString.Insert(What: pString; Where: word); virtual;*

Inserts a substring into a string (0 = first).

InsertPascal *procedure tString.InsertPascal(What: string; Where: word); virtual;*

Inserts a substring into a string (0 = first).

Intercept *destructor tString.Intercept; virtual;*

Intercepts and destructs a string.

Overrides tElement.Intercept.

Overrides tObject.Intercept.

IsCharactersInside *function tString.IsCharactersInside(Table: tChars): boolean; virtual;*

Returns TRUE if any of the specified characters is inside.

IsInside *function tString.IsInside(What: pString): boolean; virtual;*

Returns TRUE if a substring is inside the string.

IsToken *function tString.IsToken: boolean; virtual;*

Returns TRUE if this string can connect to other strings.

Linefeed *procedure tString.Linefeed; virtual;*

Adds a line feed sequence at the end of this string.

Locate *function tString.Locate(What: pString; Start, Stop: word; var Match: word; var Result: pString): boolean; virtual;*

Locates a substring by searching in a direction (1 or -1).

LowerCase *procedure tString.LowerCase; virtual;*

Converts this string to lower-case characters.

Minimize *procedure tString.Minimize; virtual;*

Minimizes the size of this string (according to length).

Occurrence *function tString.Occurrence(What: pString): word; virtual;*

Returns the number of occurrences of a substring.

Pascal *function tString.Pascal: string; virtual;*

Returns the first 255 characters in the string (maximum).

Replace *procedure tString.Replace(Match: pString; What: pString); virtual;*

Replaces any occurrences of a substring with something.

ReplacePascal *procedure tString.ReplacePascal(Match: string; What: string); virtual;*

Replaces any occurrences of a substring with something.

Resize *procedure tString.Resize(MaximumLength: word); virtual;*

Resizes the string to the specified maximum length.

Search *function tString.Search(What: pString; var Match: word): boolean; virtual;*

Searchs for a substring. Returns index and TRUE if found.

SearchBackward *function tString.SearchBackward(What: pString; Start: word; var Match: word): boolean; virtual;*

Searchs backwards for a substring and returns its position.

SearchClosest *function tString.SearchClosest(What: pString; Start: word; var Match: word): boolean; virtual;*

Searchs the closest match of the specified substring.

SearchForward *function tString.SearchForward(What: pString; Start: word; var Match: word): boolean; virtual;*

Searchs forwards for a substring and returns its position.

SearchOccurrence *function tString.SearchOccurrence(What: pString; Start, Index: word; var Match: word): boolean; virtual;*

Searchs the n:th occurrence of a substring.

SearchPascal *function tString.SearchPascal(What: string; var Match: word): boolean; virtual;*

Searchs for a substring. Returns index and TRUE if found.

Size *function tString.Size: word; virtual;*

Returns the length of this string (not allocation size).

Overrides tGenericElement.Size.

Overrides tElement.Size.

Space *procedure tString.Space(Width: word); virtual;*

Fills the string with some blank spaces.

StreamLoad *constructor tString.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

Overrides tGenericElement.StreamLoad.

Overrides tElement.StreamLoad.

Overrides tObject.StreamLoad.

StreamRead *procedure tString.StreamRead(Stream: pStream); virtual;*

Reads formatted text from a stream and creates an element.

Overrides tElement.StreamRead.

StreamStore *procedure tString.StreamStore(Stream: pStream); virtual;*

Stores an instance to a stream.

Overrides tGenericElement.StreamStore.

Overrides tElement.StreamStore.

Overrides tObject.StreamStore.

StreamWrite *procedure tString.StreamWrite(Stream: pStream); virtual;*

Writes formatted text that describe the element contents.

Overrides tElement.StreamWrite.

Overrides tObject.StreamWrite.

Strip *procedure tString.Strip(What: pString); virtual;*

Removes all leading and trailing substrings.

Successor *function tString.Successor: pString; virtual;*

Returns the next chained string (optional), or NIL.

Text *function tString.Text(Position: word): pChar; virtual;*

Returns a pointer to a character (0 .. n) in the string.

Truncate *procedure tString.Truncate(Where: word); virtual;*

Removes character after and including a position.

UpperCase *procedure tString.UpperCase; virtual;*

Converts this string to upper-case characters.

tStringMatcher EFSTRING

EFSTRING.tStringMatcher = object(tObject)

This class defines a string matcher, that is a string that rapidly knows if it is equal to some substring in an arbitrary string. This matcher uses common character matching.

The method *Match* optionally returns a mismatch position. This position (0 = unknown) is added to the current search position (in the user of the string matcher): it guarantees that there cannot be a match within an interval.

Fields and Methods:

public
constructor *Initialize...*
procedure *AssignString...* virtual;
function *Match...* virtual;
constructor *StreamLoad...*
private
fMatch: *pString*;
fLength: word;

Fields

fLength *tStringMatcher.fLength: word;*

fMatch *tStringMatcher.fMatch: pString;*

Methods

AssignString *procedure tStringMatcher.AssignString(What: pString); virtual;*

Assigns a new string (owned by some external instance).

Initialize *constructor tStringMatcher.Initialize(What: pString);*

Initializes and constructs a string matcher instance.

Overrides tObject.Initialize.

Match *function tStringMatcher.Match(Inside: pString; MaximumLength, Start, Stop: word; var Mismatch: word): boolean; virtual;*

Checks if there is a match in the specified string segment.

StreamLoad *constructor tStringMatcher.StreamLoad(Stream: pStream);*

Constructs and loads an instance from a stream.

Overrides tObject.StreamLoad.

tStyle EFSTYLES

EFSTYLES.tStyle = object(tSet)

Fields and Methods:

public
constructor *Initialize*;
procedure *Enable...* virtual;
procedure *Disable...* virtual;
private
fSettings: longint;

Fields

fSettings *tStyle.fSettings: longint;*

Methods

Disable *procedure tStyle.Disable(Setting: byte); virtual;*

Enable *procedure tStyle.Enable(Setting: byte); virtual;*

Initialize *constructor tStyle.Initialize;*

Overrides tSet.Initialize.

Overrides tObject.Initialize.

tSwapFilter EFSYSTEM

EFSYSTEM.tSwapFilter = object(tFilter)

Fields and Methods:

public
private

tSystemProfile EFSYSTEM

EFSYSTEM.tSystemProfile = object(tProfile)

Fields and Methods:

public
constructor *Initialize*;
destructor *Intercept*; virtual;

Methods

Initialize *constructor tSystemProfile.Initialize;*

Initializes the default system profile.

Overrides tNamedElement.Initialize.

Overrides tObject.Initialize.

Intercept *destructor tSystemProfile.Intercept; virtual;*

Intercepts the system profile.

Overrides tElement.Intercept.

Overrides tObject.Intercept.

tTemporaryFile EFFILE

EFFILE.tTemporaryFile = object(tFile)

This class defines a temporary file stream, that is a tFile that is automatically created as a new, unique file in the current directory. The temporary file is automatically deleted when the instance destructs. TemporaryFile can be used as a swap file.

Fields and Methods:

public
constructor *Initialize...*

destructor *Intercept*; virtual;
private
fFilename: string;

Fields

fFilename *tTemporaryFile.fFilename: string;*

Name of the opened file (deleted when destructed).

Methods

Initialize *constructor tTemporaryFile.Initialize(SizeOfBuffer: word);*

Initializes a temporary file stream with some buffer.

Overrides tFile.Initialize.

Overrides tBufferFilter.Initialize.

Overrides tFilter.Initialize.

Overrides tStream.Initialize.

Overrides tObject.Initialize.

Intercept *destructor tTemporaryFile.Intercept; virtual;*

Intercepts and destructs an instance of this class.

Overrides tBufferFilter.Intercept.

Overrides tFilter.Intercept.

Overrides tStream.Intercept.

Overrides tObject.Intercept.

tText EFTEXT

EFTEXT.tText = object(tTextResource)

This class defines a text structure, that is a set of text strings accessed according to some index number (the line number). A text owns a linear ADT with string elements (see *tString*).

Fields and Methods:

public
constructor *Initialize*;
destructor *Intercept*; virtual;

function *Get...* virtual;
procedure *Put...* virtual;
procedure *Insert...* virtual;
procedure *Update...* virtual;
procedure *Erase...* virtual;
constructor *StreamLoad...*
procedure *StreamStore...* virtual;
function *Lines...* virtual;
private
fADT: *pLinearADT*;

Descendants:

tHyperText
tParser
  EFCLINE.tCommandLine

Fields

fADT *tText.fADT: pLinearADT;*

The linear ADT with text contents.

Methods

Erase *procedure tText.Erase(Index: word); virtual;*

Erases an existing line in the text resource.

Overrides tTextResource.Erase.

Get *function tText.Get(Index: word): string; virtual;*

Gets a line from the text resource.

Overrides tTextResource.Get.

Initialize *constructor tText.Initialize;*

Overrides tObject.Initialize.

Insert *procedure tText.Insert(Text: string; AfterIndex: word); virtual;*

Inserts a line after the specified line.

Overrides tTextResource.Insert.

Intercept *destructor tText.Intercept; virtual;*

Overrides tObject.Intercept.

Lines *function tText.Lines: word; virtual;*

Returns the total number of lines in this text resource.

Overrides tTextResource.Lines.

Put *procedure tText.Put(Text: string); virtual;*

Puts a new line to the end of the text resource.

Overrides tTextResource.Put.

StreamLoad *constructor tText.StreamLoad(Stream: pStream);*

Constructs and loads an instance from a stream.

Overrides tObject.StreamLoad.

StreamStore *procedure tText.StreamStore(Stream: pStream); virtual;*

Stores an instance to a stream.

Overrides tObject.StreamStore.

Update *procedure tText.Update(Text: string; Index: word); virtual;*

Updates an existing line in the text resource.

Overrides tTextResource.Update.

tTextFile EFTEXT

EFTEXT.tTextFile = object(tTextResource)

This is a text resource (see #tTextFile.) that operates directly on an arbitrary *tStream* instance via a text filter (see *tTextFilter*).

Fields and Methods:

public
constructor *Initialize...*
destructor *Intercept*; virtual;
procedure *SetStream...* virtual;
function *Get...* virtual;
procedure *Put...* virtual;
procedure *Insert...* virtual;
procedure *Update...* virtual;
procedure *Erase...* virtual;
constructor *StreamLoad...*
procedure *StreamStore...* virtual;

function *Stream...* virtual;
function *Lines...* virtual;
private
fFilename: *pString*;
fBase: *pTextFilter*;

Fields

fBase *tTextFile.fBase: pTextFilter;*

Text filter with some base stream.

fFilename *tTextFile.fFilename: pString;*

Filename of the text file.

Methods

Erase *procedure tTextFile.Erase(Index: word); virtual;*

Erases an existing line in the text resource.

Overrides tTextResource.Erase.

Get *function tTextFile.Get(Index: word): string; virtual;*

Gets a line from the text resource.

Overrides tTextResource.Get.

Initialize *constructor tTextFile.Initialize(Filename: string);*

Initializes a text file with the specified name.

Overrides tObject.Initialize.

Insert *procedure tTextFile.Insert(Text: string; AfterIndex: word); virtual;*

Inserts a line after the specified line.

Overrides tTextResource.Insert.

Intercept *destructor tTextFile.Intercept; virtual;*

Overrides tObject.Intercept.

Lines *function tTextFile.Lines: word; virtual;*

Returns the total number of lines in this text resource.

Overrides tTextResource.Lines.

Put *procedure tTextFile.Put(Text: string); virtual;*

Puts a new line to the end of the text resource.

Overrides tTextResource.Put.

SetStream *procedure tTextFile.SetStream(TextStream: pStream); virtual;*

Sets the text stream on which this class operates.

Stream *function tTextFile.Stream: pStream; virtual;*

Returns the text stream on which this class operates.

StreamLoad *constructor tTextFile.StreamLoad(Stream: pStream);*

Constructs and loads an instance from a stream.

Overrides tObject.StreamLoad.

StreamStore *procedure tTextFile.StreamStore(Stream: pStream); virtual;*

Stores an instance to a stream.

Overrides tObject.StreamStore.

Update *procedure tTextFile.Update(Text: string; Index: word); virtual;*

Updates an existing line in the text resource.

Overrides tTextResource.Update.

tTextFilter EFFILTER

EFFILTER.tTextFilter = object(tFilter)

tTextFilter behaves does not modify the data in the base stream. It merely maintains an index of the current line by checking all EOL sequences. tTextFilter also supply a seek method (*SeekLine*) that provides a fast way of seeking other lines using the current line number.

Fields and Methods:

public
constructor *Initialize...*
procedure *Read...* virtual;
procedure *Write...* virtual;
procedure *PutLine...* virtual;
function *GetLine...* virtual;
procedure *Reset*; virtual;
procedure *Seek...* virtual;
procedure *SeekLine...* virtual;
function *CountEOL...* virtual;

function *Line...* virtual;
function *Lines...* virtual;
private
fCurrentLine: longint;
fEOLPosition: byte;

Fields

fCurrentLine *tTextFilter.fCurrentLine: longint;*

Line number for the current base stream index.

fEOLPosition *tTextFilter.fEOLPosition: byte;*

Current position inside the EOL sequence (internal).

Methods

CountEOL *function tTextFilter.CountEOL(var Data; Count: word): word; virtual;*

Counts occurrences of EOL in the specified data.

GetLine *function tTextFilter.GetLine: string; virtual;*

Gets a text line from the stream (read until CR/LF).

Overrides tStream.GetLine.

Initialize *constructor tTextFilter.Initialize(Stream: pStream);*

Initializes an instance as dependent on a base stream.

Overrides tFilter.Initialize.

Overrides tStream.Initialize.

Overrides tObject.Initialize.

Line *function tTextFilter.Line: longint; virtual;*

Returns the current line number (1..n).

Lines *function tTextFilter.Lines: longint; virtual;*

Returns the total number of text lines (EOL sequences).

PutLine *procedure tTextFilter.PutLine(Data: string); virtual;*

Puts a text line into the stream (with CR/LF).

Overrides tStream.PutLine.

Read *procedure tTextFilter.Read(var Data; Count: word); virtual;*

Reads specified amount of data from this stream.

Overrides tFilter.Read.

Overrides tStream.Read.

Reset *procedure tTextFilter.Reset; virtual;*

Resets stream and moves to the first position.

Overrides tStream.Reset.

Seek *procedure tTextFilter.Seek(Where: longint); virtual;*

Seeks, ie. moves to, a position in the stream [0 .. n].

Overrides tFilter.Seek.

Overrides tStream.Seek.

SeekLine *procedure tTextFilter.SeekLine(LineToSeek: word); virtual;*

Seeks the specified line number (counting EOL sequences).

Write *procedure tTextFilter.Write(var Data; Count: word); virtual;*

Writes specified data to this stream.

Overrides tFilter.Write.

Overrides tStream.Write.

tTextHashPlug EFTABLE

EFTABLE.tTextHashPlug = object(tHashPlug)

This is a hash function class that have been optimized for text elements. See *tHashPlug* for more information.

Fields and Methods:

public
constructor *Initialize...*
function *Hash...* virtual;
constructor *StreamLoad...*

Methods

Hash *function tTextHashPlug.Hash(var Key; KeySize: word): word; virtual;*

Returns a virtual index for the specified key.

Overrides tHashPlug.Hash.

Initialize *constructor tTextHashPlug.Initialize(PlugManager: pPlugManager);*

Initializes a hash function plug and attempts to install.

Overrides tHashPlug.Initialize.

Overrides tPlug.Initialize.

Overrides tObject.Initialize.

StreamLoad *constructor tTextHashPlug.StreamLoad(Stream: pStream);*

Constructs and loads an instance from a stream.

Overrides tHashPlug.StreamLoad.

Overrides tPlug.StreamLoad.

Overrides tStaticElement.StreamLoad.

Overrides tElement.StreamLoad.

Overrides tObject.StreamLoad.

tTextResource EFTEXT

EFTEXT.tTextResource = object(tObject)

This is an abstract class that define a text resource. A text resource is either a descendant of *tText* or *tTextFile* that provides access to its text lines.

The first line has index number "1". The highest permitted index is the last line in the text resource.

Fields and Methods:

public
function *Get...* virtual;
procedure *Put...* virtual;
procedure *Insert...* virtual;
procedure *Update...* virtual;
procedure *Erase...* virtual;

function *LongestLine...* virtual;
function *ShortestLine...* virtual;
function *Lines...* virtual;

Descendants:

tText
  tHyperText
  tParser
    EFCLINE.tCommandLine
tTextFile

Methods

Erase *procedure tTextResource.Erase(Index: word); virtual;*

Erases an existing line in the text resource.

Get *function tTextResource.Get(Index: word): string; virtual;*

Gets a line from the text resource.

Insert *procedure tTextResource.Insert(Text: string; AfterIndex: word); virtual;*

Inserts a line after the specified line.

Lines *function tTextResource.Lines: word; virtual;*

Returns the total number of lines in this text resource.

LongestLine *function tTextResource.LongestLine: word; virtual;*

Returns the index of the longest line.

Put *procedure tTextResource.Put(Text: string); virtual;*

Puts a new line to the end of the text resource.

ShortestLine *function tTextResource.ShortestLine: word; virtual;*

Returns the index of the shortest line.

Update *procedure tTextResource.Update(Text: string; Index: word); virtual;*

Updates an existing line in the text resource.

EFTIME.tTime = object(tTiming)

This class handles a time: hours, minutes, seconds and milli- seconds. It provides access to the system timer and can perform simple arithmetics with other tTime instances.

Fields and Methods:

public
constructor *Initialize*;
constructor *Duplicate...*
procedure *Reset*; virtual;
procedure *SetTime...* virtual;
procedure *SetDate...* virtual;
procedure *Assign...*
procedure *AssignSystemTime*;
procedure *UpdateSystemTime*;
procedure *GetTime...*
procedure *GetDate...*
function *StringTime...*
function *StringDate...*
function *StringDay...*
function *StringMidnightTime...*
constructor *StreamLoad...*
procedure *StreamStore...* virtual;
private
fHour: byte;
fMinute: byte;
fSecond: byte;
fSec100: word;
fMS: word;

EFBASIC.tTime = object(tObject)

This object handles time and date including conversions and calculation of week and year day number.

Fields and Methods:

public
constructor *Initialize*;
constructor *Duplicate...*

procedure *Reset*;
procedure *SetTime*...
procedure *SetDate*...
procedure *Assign*...
procedure *AssignSystemTime*;
procedure *UpdateSystemTime*;
procedure *GetTime*...
procedure *GetDate*...
function *StringTime*...
function *StringDate*...
function *StringDay*...
function *StringMidnightTime*...
constructor *StreamLoad*...
procedure *StreamStore*... virtual;
function *DayOfWeek*...
function *DayOfYear*...
function *IsValid*...
function *IsLeapYear*...
function *IsEqual*... virtual;
private
fYear,
fMonth,
fDay,
fHour,
fMinute,
fSecond,
fSec100: word;

Fields

fDay *See fYear*

fHour *See fYear*

fHour *tTime.fHour: byte;*

Hour within (1..24).

fMinute *See fYear*

fMinute *tTime.fMinute: byte;*

Minute within (0..60).

fMonth *See fYear*

fMS *tTime.fMS: word;*

* Millisecond (thousands of seconds) within (0..1000). New!

fSec100 *tTime.fSec100: word;*

* Hundredths of seconds within (0..99). Outdated!

fSec100 *See fYear*

fSecond *See fYear*

fSecond *tTime.fSecond: byte;*

Second within (0..60).

fYear *tTime.fYear,*

fMonth,

fDay,

fHour,

fMinute,

fSecond,

fSec100: word;

Time and date fields.

Methods

Assign *procedure tTime.Assign(Time: pTime);*

Assigns the properties of an instance to this instance.

Assign *procedure tTime.Assign(Time: pTime);*

Assigns the properties of an instance to this instance.

AssignSystemTime *procedure tTime.AssignSystemTime;*

Assigns the current system time to this instance.

AssignSystemTime *procedure tTime.AssignSystemTime;*

Assigns the current system time to this instance.

DayOfWeek *function tTime.DayOfWeek: byte;*

Returns the day of the week (0 .. 7).

DayOfYear *function tTime.DayOfYear: word;*

Returns the day of the year (0 .. 365).

Duplicate *constructor tTime.Duplicate(Time: pTime);*

Initializes a duplicate instance.

Duplicate *constructor tTime.Duplicate(Time: pTime);*

Initializes a duplicate instance.

GetDate *procedure tTime.GetDate(var Year, Month, Day: word);*

Gets the date from instance.

GetDate *procedure tTime.GetDate(var Year, Month, Day: word);*

Gets the date from instance.

GetTime *procedure tTime.GetTime(var Hour, Minute, Second, Sec100: word);*

Gets the time from instance.

GetTime *procedure tTime.GetTime(var Hour, Minute, Second, Sec100: word);*

Gets the time from instance.

Initialize *constructor tTime.Initialize;*

Initializes and constructs an instance of this type.

Overrides tObject.Initialize.

Initialize *constructor tTime.Initialize;*

Initializes and constructs an instance of this type.

Overrides tObject.Initialize.

IsEqual *function tTime.IsEqual(What: pObject): boolean; virtual;*

Are objects data equal?

Overrides tObject.IsEqual.

IsLeapYear *function tTime.IsLeapYear: boolean;*

Is this a leap year?

IsValid *function tTime.IsValid: boolean;*

Is time and date valid?

Reset *procedure tTime.Reset; virtual;*

Resets the time to 00:00:00.00.

Overrides tTiming.Reset.

Reset *procedure tTime.Reset;*

Resets time and date.

SetDate *procedure tTime.SetDate(Year, Month, Day: word);*

Sets the date stored in the instance.

SetDate *procedure tTime.SetDate(Year, Month, Day: word); virtual;*

SetTime *procedure tTime.SetTime(Hour, Minute, Second, Sec100: word); virtual;*

Sets the time.

SetTime *procedure tTime.SetTime(Hour, Minute, Second, Sec100: word);*

Sets the time stored in the instance.

StreamLoad *constructor tTime.StreamLoad(Stream: pStream);*

Loads the object from a stream.

Overrides tObject.StreamLoad.

StreamLoad *constructor tTime.StreamLoad(Stream: pStream);*

Loads the object from a stream.

Overrides tStaticElement.StreamLoad.

Overrides tElement.StreamLoad.

Overrides tObject.StreamLoad.

StreamStore *procedure tTime.StreamStore(Stream: pStream); virtual;*

Stores the object to a stream.

Overrides tStaticElement.StreamStore.

Overrides tElement.StreamStore.

Overrides tObject.StreamStore.

StreamStore *procedure tTime.StreamStore(Stream: pStream); virtual;*

Stores the object to a stream.

Overrides tObject.StreamStore.

StringDate *function tTime.StringDate: string;*

Returns a string with the date.

StringDate *function tTime.StringDate: string;*

Returns a string with the date.

StringDay *function tTime.StringDay: string;*

Returns a string with the name of the day.

StringDay *function tTime.StringDay: string;*

Returns a string with the name of the day.

StringMidnightTime *function tTime.StringMidnightTime: string;*

Returns a string with the time passed since midnight.

StringMidnightTime *function tTime.StringMidnightTime: string;*

Returns a string with the time passed since midnight.

StringTime *function tTime.StringTime: string;*

Returns a string with the full time and date.

StringTime *function tTime.StringTime: string;*

Returns a string with the full time and date.

UpdateSystemTime *procedure tTime.UpdateSystemTime;*

Updates the current system time to this time and date.

UpdateSystemTime *procedure tTime.UpdateSystemTime;*

Updates the current system time to this time and date.

tTimeDate EFTIME

EFTIME.tTimeDate = object(tDate)

Fields and Methods:

private
fTime: *pTime*;

Fields

fTime *tTimeDate.fTime: pTime;*

Time in hours, minutes, seconds and milliseconds.

tTimer EFBASIC

EFBASIC.tTimer = object(tObject)

This object controls timing and can be instanciated into independent timer instances that measures time periods with millisecond precision. This object calls the internal timer chip in the computer.

Fields and Methods:

public
constructor *Initialize*;
procedure *Reset*; virtual;
procedure *Delay...*
procedure *Time...*
function *StringTime...*
function *StringMS...*
function *ElapsedSeconds...*
function *ElapsedMS...*
private
fTimer: longint;
fChannel: byte;
function *TimerChipStatus...*

EFTIME.tTimer = object(tObject)

This object controls timing and can be instanciated into independent timer instances that measures time periods with millisecond precision. This object calls the internal timer chip in the computer.

Fields and Methods:

public
constructor *Initialize*;
procedure *Reset*; virtual;
procedure *Delay...*
procedure *Time...*
function *StringTime...* virtual;
function *StringMS...* virtual;
function *ElapsedSeconds...* virtual;
function *ElapsedMS...* virtual;
private
fTimer: longint;
fChannel: byte;
function *TimerChipStatus...*

Fields

fChannel *tTimer.fChannel: byte;*

Current timer channel.

fChannel *tTimer.fChannel: byte;*

Current timer channel.

fTimer *tTimer.fTimer: longint;*

Relative timer status.

fTimer *tTimer.fTimer: longint;*

Relative timer status.

Methods

Delay *procedure tTimer.Delay(MS: longint);*

Wait specified number of milliseconds.

Delay *procedure tTimer.Delay(MS: longint);*

Wait specified number of milliseconds.

ElapsedMS *function tTimer.ElapsedMS: real; virtual;*

Returns the number of milliseconds passed since start.

ElapsedMS *function tTimer.ElapsedMS: real;*

Returns the number of milliseconds passed since start.

ElapsedSeconds *function tTimer.ElapsedSeconds: real;*

Returns the number of seconds passed since start.

ElapsedSeconds *function tTimer.ElapsedSeconds: real; virtual;*

Returns the number of seconds passed since start.

Initialize *constructor tTimer.Initialize;*

Initializes and constructs an instance of this type.

Overrides tObject.Initialize.

Initialize *constructor tTimer.Initialize;*

Initializes and constructs an instance of this type.

Overrides tObject.Initialize.

Reset *procedure tTimer.Reset; virtual;*

Resets the timer (restarts timing).

Reset *procedure tTimer.Reset; virtual;*

Resets the timer (restarts timing).

StringMS *function tTimer.StringMS: string; virtual;*

Returns a string with the milliseconds passed since start.

StringMS *function tTimer.StringMS: string;*

Returns a string with the milliseconds passed since start.

StringTime *function tTimer.StringTime: string; virtual;*

Returns a string with the full timer status.

StringTime *function tTimer.StringTime: string;*

Returns a string with the full timer status.

Time *procedure tTimer.Time(What: pTime);*

Converts timer status into a time object.

Time *procedure tTimer.Time(What: pTime);*

Converts timer status into a time object.

TimerChipStatus *function tTimer.TimerChipStatus: longint;*

Returns the status of the timer chip.

TimerChipStatus *function tTimer.TimerChipStatus: longint;*

Returns the status of the timer chip.

tTiming EFTIME

EFTIME.tTiming = object(tStaticElement)

This is an abstact class that defines the common properties of all time and date classes in *EFTIME*.

Fields and Methods:

public
procedure *Reset*; virtual;
procedure *PassedTime...* virtual;
function *IsEqual...* virtual;

Descendants:

tDate
  tTimeDate
tTime

Methods

IsEqual *function tTiming.IsEqual(What: pObject): boolean; virtual;*

Are objects data equal?

Overrides tElement.IsEqual.

Overrides tObject.IsEqual.

PassedTime *procedure tTiming.PassedTime(var Days, Minutes, Seconds, MS: longint); virtual;*

Returns the passed time in some unit(s).

Reset *procedure tTiming.Reset; virtual;*

Resets the time to 00:00:00.00.

tToken EFSTRING

EFSTRING.tToken = object(tString)

This class extends the string class and enables strings to connect to other strings. You can work with set of strings as if they actually where only one string, and search for several substrings simultaneously. Tokens form a singly linked structure if they connect to other tokens.

Fields and Methods:

public
constructor *Initialize...*
constructor *InitializeEmpty*;
constructor *Duplicate...*
destructor *Intercept*; virtual;
function *Connect...* virtual;
function *Locate...* virtual;
procedure *Separate...* virtual;
procedure *Split...* virtual;
procedure *Expand*; virtual;
procedure *Merge*; virtual;
function *CreateString...* virtual;
function *CreateToken...* virtual;
function *Successor...* virtual;
function *IsToken...* virtual;
private
fSuccessor: *pString*;

Fields

fSuccessor

tToken.fSuccessor: pString;

Successor string in this token chain (any tString).

Methods

Connect

function tToken.Connect(What: pString): boolean; virtual;

Attempts to connect the specified token or string.

CreateString

function tToken.CreateString: pString; virtual;

Creates an instance of the string class (tString).

Overrides tString.CreateString.

CreateToken

function tToken.CreateToken: pToken; virtual;

Creates an instance of the token class (tToken).

Duplicate

constructor tToken.Duplicate(What: pString);

Initializes an duplicated string instance.

Overrides tString.Duplicate.

Overrides tGenericElement.Duplicate.

Expand

procedure tToken.Expand; virtual;

Expands the string using default delimeters.

Initialize

constructor tToken.Initialize(PascalString: string);

Initializes a string with the specified text content.

Overrides tString.Initialize.

Overrides tGenericElement.Initialize.

Overrides tObject.Initialize.

InitializeEmpty

constructor tToken.InitializeEmpty;

Initializes a string without text content.

Overrides tString.InitializeEmpty.

Overrides tGenericElement.InitializeEmpty.

Intercept

destructor tToken.Intercept; virtual;

Intercepts and destructs a string.

Overrides tString.Intercept.

Overrides tElement.Intercept.

Overrides tObject.Intercept.

IsToken *function tToken.IsToken: boolean; virtual;*

Returns TRUE if this string can connect to other strings.

Overrides tString.IsToken.

Locate *function tToken.Locate(What: pString; Start, Stop: word; var Match: word; var Result: pString): boolean; virtual;*

Locates a substring by searching in a direction (1 or -1).

Overrides tString.Locate.

Merge *procedure tToken.Merge; virtual;*

Merges all chained substrings to 64K string(s).

Separate *procedure tToken.Separate(Delimeters: pString); virtual;*

Splits into chained substrings using delimeter(s).

Split *procedure tToken.Split(Delimeters: pString; LengthOfSegment: word); virtual;*

Splits into fixed-length substrings (0 = delimeters).

Successor *function tToken.Successor: pString; virtual;*

Returns the next chained string (optional), or NIL.

Overrides tString.Successor.

tTranslation EFSTRING

EFSTRING.tTranslation = object(tObject)

Translation table class. This class filters characters and strings.

Fields and Methods:

public
constructor *Initialize*;
constructor *Duplicate...*
procedure *SetTranslation...*
procedure *SetUpper*;
procedure *SetLower*;
procedure *Translate...*
private
fTable: array...

Fields

fTable *tTranslation.fTable: array[char] of char;*

Translation table.

Methods

Duplicate *constructor tTranslation.Duplicate(Instance: pTranslation);*

Initializes and duplicates an instance of this class.

Initialize *constructor tTranslation.Initialize;*

Initializes and constructs an instance of this class.

Overrides tObject.Initialize.

SetLower *procedure tTranslation.SetLower;*

Sets standard lower-case translation characters.

SetTranslation *procedure tTranslation.SetTranslation(Data: string; AtOdd: boolean);*

Sets translation according to the string, e.g. "AaBbCc".

SetUpper *procedure tTranslation.SetUpper;*

Sets standard upper-case translation characters.

Translate *procedure tTranslation.Translate(var Data; Size: word);*

Translates the specified variable.

tTree EFTREE

EFTREE.tTree = object(tOrderedADT)

This class defines a generic tree data structure, that is a tree where each node can have arbitrary many children.

Fields and Methods:

public
constructor *Initialize...*
destructor *Intercept*; virtual;
procedure *Clear*; virtual;
procedure *Put...* virtual;
function *Get...* virtual;
procedure *Erase...* virtual;

function *FindNode...* virtual;
procedure *Rebuild*; virtual;
function *CreateIterator...* virtual;
function *CreateNode...* virtual;
constructor *StreamLoad...*
function *Elements...* virtual;
function *Root...* virtual;
private
fRoot: *pTreeLinkage*;

Fields

fRoot *tTree.fRoot: pTreeLinkage;*

The root node for this tree (NIL if empty).

Methods

Clear *procedure tTree.Clear; virtual;*

Clears or removes all elements inside data type.

Overrides tADT.Clear.

CreateIterator *function tTree.CreateIterator: pIterator; virtual;*

Creates an instance of the iterator class.

Overrides tADT.CreateIterator.

CreateNode *function tTree.CreateNode(Element: pElement): pTreeLinkage; virtual;*

Creates an instance of the tree linkage class.

Elements *function tTree.Elements: word; virtual;*

Returns the number of elements that currently are used.

Overrides tADT.Elements.

Erase *procedure tTree.Erase(var Key; KeySize: word); virtual;*

Erases element(s) with the specified search key.

Overrides tOrderedADT.Erase.

FindNode *function tTree.FindNode(var Target: pTreeLinkage; Element: pElement): boolean; virtual;*

Finds the closest node, returns TRUE if match.

Get *function tTree.Get(var Key; KeySize: word): pElement; virtual;*

Gets the specified element instance.

Overrides tOrderedADT.Get.

Initialize *constructor tTree.Initialize(SizeOfElements: word);*

Initializes an empty tree ADT and installs path plug.

Overrides tADT.Initialize.

Overrides tObject.Initialize.

Intercept *destructor tTree.Intercept; virtual;*

Intercepts the tree and removes all nodes.

Overrides tADT.Intercept.

Overrides tObject.Intercept.

Put *procedure tTree.Put(Element: pElement); virtual;*

Stores a new element (tElement instance) to this ADT.

Overrides tADT.Put.

Rebuild *procedure tTree.Rebuild; virtual;*

Rebuilds the tree (optionally makes it balanced).

Root *function tTree.Root: pTreeLinkage; virtual;*

Returns the root node or NIL if tree is empty.

StreamLoad *constructor tTree.StreamLoad(Stream: pStream);*

Loads an ADT from a stream.

Overrides tADT.StreamLoad.

Overrides tObject.StreamLoad.

tTreeIterator EFTREE

EFTREE.tTreeIterator = object(tIterator)

This class defines a generic tree iterator. This iterator does not operate directly on the nodes. Instead, it constructs a data structure with all the elements when the iterator is created. When you are traversing a tree, you must remember not to update the tree while the iterator is being used.

tTreeIterator provides four traversal orders: postorder, preorder, inorder and level-order. Postorder always visits the parent after its children, preorder visits the parent first, and then moves to the children, inorder begins with the left-most children, then visits the parent and then the right-most children. Level-order visits leaves on a certain level, and then moves closer to the tree root.

Fields and Methods:

public
constructor *Initialize...*
destructor *Intercept*; virtual;
procedure *Build...* virtual;
procedure *First*; virtual;
procedure *Last*; virtual;
function *Element...* virtual;
function *Position...* virtual;
procedure *WalkForward*; virtual;
procedure *WalkBackward*; virtual;
function *IsAllocated...* virtual;
function *IsEnd...* virtual;
function *IsValid...* virtual;
function *IsEqual...* virtual;
function *IsCompatible...* virtual;
private
fMirror: *pADT*;
fIterator: *pIterator*;

Fields

fIterator *tTreeIterator.fIterator: pIterator;*

Internal iterator that operates on the mirror structure.

fMirror *tTreeIterator.fMirror: pADT;*

Mirror of the nodes to be traversed.

Methods

Build *procedure tTreeIterator.Build(Order: tTreeOrder); virtual;*

Builds a mirror data structure for traversal (internal).

Element *function tTreeIterator.Element: pElement; virtual;*

Returns a pointer to the current element instance.

Overrides tIterator.Element.

First *procedure tTreeIterator.First; virtual;*

Walks to the first element in this ADT.

Overrides tIterator.First.

Initialize *constructor tTreeIterator.Initialize(ADT: pTree; Order: tTreeOrder);*

Constructs a tree iterator and an auxilary data structure.

Overrides tObject.Initialize.

Intercept *destructor tTreeIterator.Intercept; virtual;*

Destructs the tree iterator after it have been used.

Overrides tObject.Intercept.

IsAllocated *function tTreeIterator.IsAllocated: boolean; virtual;*

Is this iterator assigned to an ADT?

Overrides tIterator.IsAllocated.

IsCompatible *function tTreeIterator.IsCompatible(Instance: pObject): boolean; virtual;*

Are ADT′s compatible, that is has compatible elements.

Overrides tIterator.IsCompatible.

Overrides tObject.IsCompatible.

IsEnd *function tTreeIterator.IsEnd: boolean; virtual;*

Is iterator at boundaries (beginning or end)?

Overrides tIterator.IsEnd.

IsEqual *function tTreeIterator.IsEqual(Instance: pObject): boolean; virtual;*

Are contents of these instances equal (same cursor)?

Overrides tObject.IsEqual.

IsValid *function tTreeIterator.IsValid(ADT: pADT): boolean; virtual;*

Is the specified ADT of a valid class?

Overrides tIterator.IsValid.

Last *procedure tTreeIterator.Last; virtual;*

Walks to the last element in this ADT.

Overrides tIterator.Last.

Position *function tTreeIterator.Position: word; virtual;*

Returns the current position inside the ADT.

Overrides tIterator.Position.

WalkBackward *procedure tTreeIterator.WalkBackward; virtual;*

Walk backwards (to the predecessor element).

Overrides tIterator.WalkBackward.

WalkForward *procedure tTreeIterator.WalkForward; virtual;*

Walk forwards (to the successor element).

Overrides tIterator.WalkForward.

tTreeLinkage EFTREE

EFTREE.tTreeLinkage = object(tLinkage)

This class defines a node for a generic tree with multiple children associated to a single node. All nodes are doubly linked (there are links in both directions).

tTreeLinkage connects to other nodes at the same level using the mechanism inherited from *tLinkage*. tTreeLinkage also connects to arbitrary many children (in the *fChildren* member field) and knows its parent node (*fParent*).

Nodes that does not have a parent node is called root nodes and nodes without children leaves.

Fields and Methods:

public
constructor *Initialize...*
destructor *Intercept*; virtual;
procedure *SetParent...* virtual;
procedure *SetLeft...* virtual;
procedure *SetRight...* virtual;
procedure *SetChildren...* virtual;
procedure *AttachAfter...* virtual;
procedure *AttachBefore...* virtual;
procedure *Detach*; virtual;
procedure *AttachChildren...* virtual;
procedure *DetachChildren*; virtual;
function *Left...* virtual;

function *Right...* virtual;
function *Parent...* virtual;
function *Children...* virtual;
function *LeftMostChildren...* virtual;
function *RightMostChildren...* virtual;
function *Depth...* virtual;
function *Height...* virtual;
function *Balance...* virtual;
function *Childrens...* virtual;
function *TotalChildrens...* virtual;
function *HasParent...* virtual;
function *HasChildren...* virtual;
function *IsAttached...* virtual;
function *IsChildren...* virtual;
function *IsIntact...* virtual;
private
fChildren: *pTreeLinkage*;
fParent: *pTreeLinkage*;

Descendants:

tAVLLinkage

Fields

fChildren *tTreeLinkage.fChildren: pTreeLinkage;*

Chain with children or NIL if no children exists.

fParent *tTreeLinkage.fParent: pTreeLinkage;*

Parent node for this children, or NIL.

Methods

AttachAfter *procedure tTreeLinkage.AttachAfter(Container: pContainer); virtual;*

Attachs container(s) after this container (by insertion).

Overrides tContainer.AttachAfter.

AttachBefore *procedure tTreeLinkage.AttachBefore(Container: pContainer); virtual;*

Attachs container(s) before this container (by insertion).

Overrides tContainer.AttachBefore.

AttachChildren *procedure tTreeLinkage.AttachChildren(ChildrenNode: pTreeLinkage); virtual;*

Attachs a new children to this node.

Balance *function tTreeLinkage.Balance: longint; virtual;*

Difference in height between first and last children.

Children *function tTreeLinkage.Children(Index: word): pTreeLinkage; virtual;*

Returns the n:th children (1..n) to this node, or NIL.

Childrens *function tTreeLinkage.Childrens: word; virtual;*

Returns the number of children connected to this node.

Depth *function tTreeLinkage.Depth: word; virtual;*

Returns the depth of this node (distance to leaves).

Detach *procedure tTreeLinkage.Detach; virtual;*

Detachs this container (preserves any structure).

Overrides tContainer.Detach.

DetachChildren *procedure tTreeLinkage.DetachChildren; virtual;*

Detachs all children nodes without destructing them.

HasChildren *function tTreeLinkage.HasChildren: boolean; virtual;*

Has this node at least one children node?

HasParent *function tTreeLinkage.HasParent: boolean; virtual;*

Has this node at least one parent node?

Height *function tTreeLinkage.Height: word; virtual;*

Returns the height of this node (distance to root).

Initialize *constructor tTreeLinkage.Initialize(HoldElement: pElement; ParentNode: pTreeLinkage);*

Initializes a new node and attaches it to a parent node.

Overrides tLinkage.Initialize.

Overrides tCarrier.Initialize.

Overrides tContainer.Initialize.

Overrides tObject.Initialize.

Intercept *destructor tTreeLinkage.Intercept; virtual;*

Intercepts this node and all its children.

Overrides tCarrier.Intercept.

Overrides tContainer.Intercept.

Overrides tObject.Intercept.

IsAttached *function tTreeLinkage.IsAttached: boolean; virtual;*

Is this node attached, ie. linked to other nodes?

Overrides tContainer.IsAttached.

IsChildren *function tTreeLinkage.IsChildren(Node: pTreeLinkage): boolean; virtual;*

Is the specified node a children to this node?

IsIntact *function tTreeLinkage.IsIntact: boolean; virtual;*

Cerfifies that this container has valid links (or none).

Overrides tContainer.IsIntact.

Left *function tTreeLinkage.Left: pTreeLinkage; virtual;*

Returns the left children, or NIL if there is no children.

LeftMostChildren *function tTreeLinkage.LeftMostChildren: pTreeLinkage; virtual;*

Returns the left-most leave node below this node.

Parent *function tTreeLinkage.Parent: pTreeLinkage; virtual;*

Returns the parent to this node.

Right *function tTreeLinkage.Right: pTreeLinkage; virtual;*

Returns the rigth children, or NIL non-existing.

RightMostChildren *function tTreeLinkage.RightMostChildren: pTreeLinkage; virtual;*

Returns the right-most leave node below this node.

SetChildren *procedure tTreeLinkage.SetChildren(Link: pTreeLinkage); virtual;*

Sets the children chain.

SetLeft *procedure tTreeLinkage.SetLeft(Link: pTreeLinkage); virtual;*

Sets the left children node.

SetParent *procedure tTreeLinkage.SetParent(Link: pTreeLinkage); virtual;*

Sets the parent node.

SetRight *procedure tTreeLinkage.SetRight(Link: pTreeLinkage); virtual;*

Sets the right children node.

TotalChildrens *function tTreeLinkage.TotalChildrens: word; virtual;*

Returns the number of childrens below this node.

tTreeOrder EFDEF

EFDEF.tTreeOrder = (PostOrder, PreOrder, InOrder, LevelOrder);

This type stores the traversal orders for tree data structures in *EFTREE*. See tTreeIterator for details.

tVariable EFMEXP

EFMEXP.tVariable = object(tNamedElement)

This is an element class that associates a name of a variable to some mathematical object (the value). These variables are handled in the tVariables collection and are closely related to expression evaluation (see *tExpression*).

Fields and Methods:

public
private
fValue: *pMathObject*;

Fields

fValue *tVariable.fValue: pMathObject;*

tVector EFMATH

EFMATH.tVector = object(tMathObject)

This class defines a 3-dimensional vector, ie. a quantity that involves both magnitude (size or length) and direction. Such quantities are represented geometrically by arrows (directed line segments).

Fields and Methods:

public
constructor *Initialize...*
constructor *Duplicate...*
procedure *SetValue...*
procedure *Add...* virtual;
procedure *Subtract...* virtual;
procedure *Multiply...* virtual;
procedure *Divide...* virtual;
function *Magnitude...*
function *Projection...*
function *DotProduct...*
function *Angle...*
procedure *Normalize*;
constructor *StreamLoad...*
procedure *StreamStore...* virtual;
procedure *StreamWrite...* virtual;
function *X...*
function *Y...*
function *Z...*
function *IsZero...* virtual;
private
fX,
fY,
fZ: *tBaseReal*;

Fields

fX *tVector.fX,*

fY,

fZ: tBaseReal;

Components for this vector (X, Y, Z).

fY *See fX*

fZ *See fX*

Methods

Add *procedure tVector.Add(What: pMathObject); virtual;*

Adds an object to this object.

Overrides tMathObject.Add.

Angle *function tVector.Angle(What: pVector): tBaseReal;*

Returns the angle between this vector and the specified.

Divide *procedure tVector.Divide(What: pMathObject); virtual;*

Divides this object by another (non-vector) object.

Overrides tMathObject.Divide.

DotProduct *function tVector.DotProduct(What: pVector): tBaseReal;*

Returns the dot product of the two vectors (a real number).

Duplicate *constructor tVector.Duplicate(What: pMathObject);*

Initializes an duplicated type-casted mathematical object. #Z Setup methods

Initialize *constructor tVector.Initialize(nX, nY, nZ: tBaseReal);*

Initializes an instance with the specified content.

Overrides tObject.Initialize.

IsZero *function tVector.IsZero: boolean; virtual;*

Is this a zero vector?

Overrides tMathObject.IsZero.

Magnitude *function tVector.Magnitude: tBaseReal;*

Returns the magnitude of the number, ie. the length.

Multiply *procedure tVector.Multiply(What: pMathObject); virtual;*

Multiplies this object with another object.

Overrides tMathObject.Multiply.

Normalize *procedure tVector.Normalize;*

Normalizes this vector, ie. adjusts X, Y, Z to [-1, 1].

Projection *function tVector.Projection(What: pVector): tBaseReal;*

Returns the scalar projection in the specified direction.

SetValue *procedure tVector.SetValue(nX, nY, nZ: tBaseReal);*

Sets the number.

StreamLoad *constructor tVector.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

Overrides tStaticElement.StreamLoad.

Overrides tElement.StreamLoad.

Overrides tObject.StreamLoad.

StreamStore *procedure tVector.StreamStore(Stream: pStream); virtual;*

Stores an instance to a stream.

Overrides tStaticElement.StreamStore.

Overrides tElement.StreamStore.

Overrides tObject.StreamStore.

StreamWrite *procedure tVector.StreamWrite(Stream: pStream); virtual;*

Writes this math object as formatted text, e.g. "2+2i".

Overrides tElement.StreamWrite.

Overrides tObject.StreamWrite.

Subtract *procedure tVector.Subtract(What: pMathObject); virtual;*

Subtracts an object from this object.

Overrides tMathObject.Subtract.

X *function tVector.X: tBaseReal;*

Returns the X coordinate.

Y *function tVector.Y: tBaseReal;*

Returns the Y coordinate.

Z *function tVector.Z: tBaseReal;*

Returns the Z coordinate.

EFVIEWS.tView = object(tComponent)

This class defines a view. A view is a class that manages a rectangular are of the screen. For example, the menu bar at the top of the screen is a view. Any program action in that area of the screen (for example clicking the mouse on the menu bar) will be dealt with by the view that controls that area.

In general, everything that shows up on the screen in EFLIB must be a view, which means it is a descendant of the tView class. There are three things all views must do:

(1) Manage a rectangular region. See *tField*.

(2) Draw itself on demand - the view must know how to represent itself on the screen (the console). The view must fill its entire rectangle. The drawing is done when the method *Draw* is called, or when the method *Update* is called.

(3) Handle events in its boundaries. See *tComponent*.

Fields and Methods:

public
constructor *Initialize...*
procedure *Draw*; virtual;
procedure *Update*; virtual;
procedure *Show*; virtual;
procedure *Hide*; virtual;
function *IsVisible...* virtual;
function *IsFieldVisible...* virtual;
private
fBoundaries: *pField*;
fMovable: boolean;
fResizable: boolean;
fGrowWithGroup: boolean;
fInsideGroup: boolean;
fFocusSelected: boolean;
fFirstClick: boolean;
fHidden: boolean;
fHelpAnchor: pHyperAnchor;

Descendants:

tGroup

Fields

fBoundaries *tView.fBoundaries: pField;*

Boundaries of the view relative to the console.

fFirstClick *tView.fFirstClick: boolean;*

Is this view selected at the first click?

fFocusSelected *tView.fFocusSelected: boolean;*

Is this view focused whenever it is selected?

fGrowWithGroup *tView.fGrowWithGroup: boolean;*

Must the view grow when its manager does?

fHelpAnchor *tView.fHelpAnchor: pHyperAnchor;*

An optional help anchor for this component.

fHidden *tView.fHidden: boolean;*

Is this view currently not displayed on any console?

fInsideGroup *tView.fInsideGroup: boolean;*

Must this view be inside the boundaries of its group?

fMovable *tView.fMovable: boolean;*

Can this move change its position in the console?

fResizable *tView.fResizable: boolean;*

Can this view change the boundaries, that is resize?

Methods

Draw *procedure tView.Draw; virtual;*

Hide *procedure tView.Hide; virtual;*

Initialize *constructor tView.Initialize(Dimension: pField);*

Constructs a view with the given dimensions.

Overrides tComponent.Initialize.

Overrides tMessage.Initialize.

Overrides tObject.Initialize.

IsFieldVisible *function tView.IsFieldVisible(Field: tField): boolean; virtual;*

Is the specified field inside the view, and visibile?

IsVisible *function tView.IsVisible: boolean; virtual;*

Is this view, or parts of it, visible on the console?

Show *procedure tView.Show; virtual;*

Update *procedure tView.Update; virtual;*

Overrides tComponent.Update.

tVirtualArray EFARRAY

EFARRAY.tVirtualArray = object(tBoundedArray)

This class defines virtual array, that is an array (see *tArray*) with the capability to shrink and grow on demand. The ratio of growth is specified at initialization. When the virtual array is full, it automatically grow the user-defined number of elements.

Virtual arrays are bounded (they are derived from *tBoundedArray*, hence being sequential with lower and higher boundary elements.

Fields and Methods:

constructor *Initialize...*
constructor *Resemble...*
constructor *Duplicate...*
procedure *SetGrowth...* virtual;
procedure *Adjust*; virtual;
constructor *StreamLoad...*
procedure *StreamLoadProperties...* virtual;
procedure *StreamStoreProperties...* virtual;
private
fGrowthSize: word;

Fields

fGrowthSize *tVirtualArray.fGrowthSize: word;*

Growth ratio, ie. the number of elements to grow or shrink.

Methods

Adjust *procedure tVirtualArray.Adjust; virtual;*

Adjust the element capacity (grow or shrink)

Overrides tArray.Adjust.

Duplicate *constructor tVirtualArray.Duplicate(ADT: pLinearADT);*

Initializes a duplicate ADT instance (with equal elements).

Overrides tBoundedArray.Duplicate.

Overrides tArray.Duplicate.

Initialize *constructor tVirtualArray.Initialize(NumberOfElements, SizeOfGrowth, SizeOfElements: word);*

Initializes an instance of this ADT class.

Overrides tBoundedArray.Initialize.

Overrides tArray.Initialize.

Overrides tLinearADT.Initialize.

Overrides tADT.Initialize.

Overrides tObject.Initialize.

Resemble *constructor tVirtualArray.Resemble(ADT: pLinearADT);*

Initializes an ADT that resembles the specified ADT.

Overrides tBoundedArray.Resemble.

Overrides tArray.Resemble.

SetGrowth *procedure tVirtualArray.SetGrowth(SizeOfGrowth: word); virtual;*

Sets growth ratio (number of elements to grow at a time).

StreamLoad *constructor tVirtualArray.StreamLoad(Stream: pStream);*

Loads an instance from a stream.

Overrides tBoundedArray.StreamLoad.

Overrides tArray.StreamLoad.

Overrides tADT.StreamLoad.

Overrides tObject.StreamLoad.

StreamLoadProperties *procedure tVirtualArray.StreamLoadProperties(Stream: pStream); virtual;*

Loads properties of this ADT from a stream (not elements).

Overrides tBoundedArray.StreamLoadProperties.

Overrides tADT.StreamLoadProperties.

StreamStoreProperties *procedure tVirtualArray.StreamStoreProperties(Stream: pStream); virtual;*

Stores properties of this ADT to a stream (not elements).

Overrides tBoundedArray.StreamStoreProperties.

Overrides tADT.StreamStoreProperties.

tVirtualConsole EFCONDRV

EFCONDRV.tVirtualConsole = object(tBoundedConsole)

Virtual console driver. This console driver is abstract, but is the parent to the *tImage* console driver. A virtual console driver keeps track of all its properties itself.

Fields and Methods:

public
constructor *Initialize...*
procedure *GotoXY...* virtual;
constructor *StreamLoad...*
function *X...* virtual;
function *Y...* virtual;
private
fCursor: *pCoordinates*;

Descendants:

tImage
EFSCREEN.tBufferedScreen

Fields

fCursor *tVirtualConsole.fCursor: pCoordinates;*

Current cursor coordinate in this console.

Methods

GotoXY

procedure tVirtualConsole.GotoXY(X1, Y1: word); virtual;

Moves to the specified coordinates.

Overrides tConsole.GotoXY.

Initialize

constructor tVirtualConsole.Initialize(Boundaries: pField);

Initializes an empty image with the specified boundaries.

Overrides tBoundedConsole.Initialize.

Overrides tConsole.Initialize.

Overrides tComponent.Initialize.

Overrides tMessage.Initialize.

Overrides tObject.Initialize.

StreamLoad

constructor tVirtualConsole.StreamLoad(Stream: pStream);

Constructs and loads an instance from a stream.

Overrides tBoundedConsole.StreamLoad.

Overrides tConsole.StreamLoad.

Overrides tComponent.StreamLoad.

Overrides tStaticElement.StreamLoad.

Overrides tElement.StreamLoad.

Overrides tObject.StreamLoad.

X

function tVirtualConsole.X: word; virtual;

Returns the current X coordinate.

Overrides tConsole.X.

Y

function tVirtualConsole.Y: word; virtual;

Returns the current Y coordinate.

Overrides tConsole.Y.

tWildcard EFPATRN

EFPATRN.tWildcard = object(tElement)

This is a pattern class that matches any other element, of any length.

Fields and Methods:

public
constructor *Initialize*;
function *IsEqual...* virtual;

Methods

Initialize *constructor tWildcard.Initialize;*

IsEqual *function tWildcard.IsEqual(Instance: pObject): boolean; virtual;*

Are the contents of these elements equal?

UniqueFilename EFSTREAM

function EFSTREAM.UniqueFilename: string;

Returns a unique filename for this directory, that is a file that does not already exist.

UpperCase EFSTRING

function EFSTRING.UpperCase(Data: string): string;

Convert a string to upper case characters using the *UpperFilter* instance in *EFSTRING.*

UpperFilter EFSTRING

Const EFSTRING.UpperFilter: pTranslation = NIL;

Lower case translation table used for case conversion of strings. See #tTranslation.

White EFVIEWS

Const EFVIEWS.White = 15;

Yellow EFVIEWS

Const EFVIEWS.Yellow = 14;