Run-Time Library (RTL) : Reference guide.

Free Pascal version 3.0.0:
Reference guide for RTL units.
Document version 2.6
February 2016

Michaël Van Canneyt

Contents

About this guide

This document describes all constants, types, variables, functions and procedures as they are declared in the units that come standard with the Free Pascal Run-Time library (RTL).

Throughout this document, we will refer to functions, types and variables with `typewriter` font. Functions and procedures gave their own subsections, and for each function or procedure we have the following topics:

Declaration The exact declaration of the function.

Description What does the procedure exactly do ?

Errors What errors can occur.

See Also Cross references to other related functions/commands.

0.1 Overview

The Run-Time Library is the basis of all Free Pascal programs. It contains the basic units that most programs will use, and are made available on all platforms supported by Free pascal (well, more or less).

There are units for compatibility with the Turbo Pascal Run-Time library, and there are units for compatibility with Delphi.

On top of these two sets, there are also a series of units to handle keyboard/mouse and text screens in a cross-platform way.

Other units include platform specific units that implement the specifics of a platform, these are usually needed to support the Turbo Pascal or Delphi units.

Units that fall outside the above outline do not belong in the RTL, but should be included in the packages, or in the FCL.

Chapter 1

Reference for unit 'BaseUnix'

1.1 Used units

Table 1.1: Used units by unit 'BaseUnix'

Name	Page
System	622
unixtype	747

1.2 Overview

The BaseUnix unit was implemented by Marco Van de Voort. It contains basic unix functionality. It supersedes the Linux unit of version 1.0.X of the compiler, but only implements a cleaned up, portable subset of that unit.

For porting FPC to new unix-like platforms, it should be sufficient to implement the functionality in this unit for the new platform.

Chapter 2

Reference for unit 'character'

2.1 Used units

Table 2.1: Used units by unit 'character'

Name	Page
System	622
unicodedata	702

2.2 Overview

The character unit contains the TCharacter (59) class, which consists mainly of class functions. It should not be constructed, but its class methods can be used. All class methods also exist as regular methods.

Many routines depend on unicode collation data to be present in the binary (or distributed on disc with the application. This data can be loaded using the routines in the unicodedata (702) unit.. The FPC project distributes some unicode collation data in `.bco` files which can be loaded using the LoadCollation (711) routine from that unit.

2.3 Constants, types and variables

2.3.1 Types

```
TCharacterOption = (coIgnoreInvalidSequence)
```

Table 2.2: Enumeration values for type TCharacterOption

Value	Explanation
coIgnoreInvalidSequence	Ignore invalid unicodecode sequences

`TCharacterOption` is used in the toUpper (59) and toLower (59) functions to control the behaviour of the function.

```
TCharacterOptions = Set of TCharacterOption
```

`TCharacterOptions` is the set of `TCharacterOption`, used in toUpper (59) and toLower (59) functions to control the behaviour of the function.

```
TUnicodeCategory = (ucUppercaseLetter,ucLowercaseLetter,
                    ucTitlecaseLetter,ucModifierLetter,ucOtherLetter,
                    ucNonSpacingMark,ucCombiningMark,ucEnclosingMark,
                    ucDecimalNumber,ucLetterNumber,ucOtherNumber,
                    ucConnectPunctuation,ucDashPunctuation,
                    ucOpenPunctuation,ucClosePunctuation,
                    ucInitialPunctuation,ucFinalPunctuation,
                    ucOtherPunctuation,ucMathSymbol,ucCurrencySymbol,
                    ucModifierSymbol,ucOtherSymbol,ucSpaceSeparator,
                    ucLineSeparator,ucParagraphSeparator,ucControl,
                    ucFormat,ucSurrogate,ucPrivateUse,ucUnassigned)
```

Table 2.3: Enumeration values for type TUnicodeCategory

Value	Explanation
ucClosePunctuation	Punctuation, close (Pe)
ucCombiningMark	Mark, spacing combining (Mc)
ucConnectPunctuation	Punctuation, connector (Pc)
ucControl	Other, control (Cc)
ucCurrencySymbol	Symbol, currency (Sc)
ucDashPunctuation	Punctuation, dash (Pd)
ucDecimalNumber	Number, decimal digit (Nd)
ucEnclosingMark	Mark, enclosing (Me)
ucFinalPunctuation	Punctuation, final quote (Pf, may behave like Ps or Pe depending on usage)
ucFormat	Other, format (Cf)
ucInitialPunctuation	Punctuation, initial quote (Pi, may behave like Ps or Pe depending on usage)
ucLetterNumber	Number, letter (Nl)
ucLineSeparator	Separator, line (Zl)
ucLowercaseLetter	Letter, lowercase (Ll)
ucMathSymbol	Symbol, math (Sm)
ucModifierLetter	Letter, modifier (Lm)
ucModifierSymbol	Symbol, modifier (Sk)
ucNonSpacingMark	Mark, nonspacing (Mn)
ucOpenPunctuation	Punctuation, open (Ps)
ucOtherLetter	Letter, other (Lo)
ucOtherNumber	Number, other (No)
ucOtherPunctuation	Punctuation, other (Po)
ucOtherSymbol	Symbol, other (So)
ucParagraphSeparator	Separator, paragraph (Zp)
ucPrivateUse	Other, private use (Co)
ucSpaceSeparator	Separator, space (Zs)
ucSurrogate	Other, surrogate (Cs)
ucTitlecaseLetter	Letter, titlecase (Lt)
ucUnassigned	Other, not assigned (including noncharacters) (Cn)
ucUppercaseLetter	Letter, uppercase (Lu)

This enumeration type contains the characterization of all possible unicode characters. It is used in the GetUnicodeCategory (55) and TCharacter.GetUnicodeCategory (62) functions.

```
TUnicodeCategorySet = Set of TUnicodeCategory
```

`TUnicodeCategorySet` is the set of TUnicodeCategory (53). It is used internally in the TCharacter (59) class.

2.4 Procedures and functions

2.4.1 ConvertFromUtf32

Synopsis: alias for `TCharacter.ConvertFromUtf32`

Declaration:
```
function ConvertFromUtf32(AChar: UCS4Char) : UnicodeString
```

Visibility: default

Description: `ConvertFromUtf32` is a shortcut for TCharacter.ConvertFromUtf32 (60).

See also: TCharacter.ConvertFromUtf32 (60)

2.4.2 ConvertToUtf32

Synopsis: alias for `TCharacter.ConvertToUtf32`

Declaration:
```
function ConvertToUtf32(const AString: UnicodeString;AIndex: Integer)
                         : UCS4Char;  Overload
function ConvertToUtf32(const AString: UnicodeString;AIndex: Integer;
                        out ACharLength: Integer) : UCS4Char;  Overload
function ConvertToUtf32(const AHighSurrogate: UnicodeChar;
                        const ALowSurrogate: UnicodeChar) : UCS4Char
                        ;  Overload
```

Visibility: default

Description: `ConvertToUtf32` is a shortcut for TCharacter.ConvertToUtf32 (61).

See also: TCharacter.ConvertToUtf32 (61)

2.4.3 GetNumericValue

Synopsis: Alias for `TCharacter.GetNumericValue`

Declaration:
```
function GetNumericValue(AChar: UnicodeChar) : Double;  Overload
function GetNumericValue(const AString: UnicodeString;AIndex: Integer)
                         : Double;  Overload
```

Visibility: default

Description: `GetNumericValue` is a shortcut for TCharacter.GetNumericValue (61).

See also: TCharacter.GetNumericValue (61)

2.4.4 GetUnicodeCategory

Synopsis: Alias for `TCharacter.GetUnicodeCategory`

Declaration:
```
function GetUnicodeCategory(AChar: UnicodeChar) : TUnicodeCategory
                            ;  Overload
function GetUnicodeCategory(const AString: UnicodeString;
                            AIndex: Integer) : TUnicodeCategory
                            ;  Overload
```

Visibility: default

Description: `GetUnicodeCategory` is a shortcut for TCharacter.GetUnicodeCategory (62).

See also: TCharacter.GetUnicodeCategory (62)

2.4.5 IsControl

Synopsis: Alias for `TCharacter.IsControl`

Declaration:
```
function IsControl(AChar: UnicodeChar) : Boolean;  Overload
function IsControl(const AString: UnicodeString;AIndex: Integer)
                   : Boolean;  Overload
```

Visibility: default

Description: `IsControl` is a shortcut for TCharacter.IsControl (62).

See also: TCharacter.IsControl (62)

2.4.6 IsDigit

Synopsis: Alias for `TCharacter.IsDigit`

Declaration:
```
function IsDigit(AChar: UnicodeChar) : Boolean;  Overload
function IsDigit(const AString: UnicodeString;AIndex: Integer) : Boolean
                 ;  Overload
```

Visibility: default

Description: `IsDigit` is a shortcut for TCharacter.IsDigit (62).

See also: TCharacter.IsDigit (62)

2.4.7 IsHighSurrogate

Synopsis: Alias for `TCharacter.IsHighSurrogate`

Declaration:
```
function IsHighSurrogate(AChar: UnicodeChar) : Boolean;  Overload
function IsHighSurrogate(const AString: UnicodeString;AIndex: Integer)
                         : Boolean;  Overload
```

Visibility: default

Description: `IsHighSurrogate` is a shortcut for TCharacter.IsHighSurrogate (63)

See also: TCharacter.IsHighSurrogate (63)

2.4.8 IsLetter

Synopsis: Alias for `TCharacter.IsLetter`

Declaration:
```
function IsLetter(AChar: UnicodeChar) : Boolean;  Overload
function IsLetter(const AString: UnicodeString;AIndex: Integer)
                  : Boolean;  Overload
```

Visibility: default

Description: `IsLetter` is a shortcut for TCharacter.IsLetter (64)

See also: TCharacter.IsLetter (64)

2.4.9 IsLetterOrDigit

Synopsis: Alias for `TCharacter.IsLetterOrDigit`

Declaration:
```
function IsLetterOrDigit(AChar: UnicodeChar) : Boolean;  Overload
function IsLetterOrDigit(const AString: UnicodeString;AIndex: Integer)
                         : Boolean;  Overload
```

Visibility: default

Description: `IsLetterOrDigit` is a shortcut for TCharacter.IsLetterOrDigit (65).

See also: TCharacter.IsLetterOrDigit (65)

2.4.10 IsLower

Synopsis: Alias for `TCharacter.IsLower`

Declaration:
```
function IsLower(AChar: UnicodeChar) : Boolean;  Overload
function IsLower(const AString: UnicodeString;AIndex: Integer) : Boolean
                 ;  Overload
```

Visibility: default

Description: `IsLower` is a shortcut for TCharacter.IsLower (65)

See also: TCharacter.IsLower (65)

2.4.11 IsLowSurrogate

Synopsis: Alias for `TCharacter.IsLowSurrogate`

Declaration:
```
function IsLowSurrogate(AChar: UnicodeChar) : Boolean;  Overload
function IsLowSurrogate(const AString: UnicodeString;AIndex: Integer)
                        : Boolean;  Overload
```

Visibility: default

Description: `IsLowSurrogate` is a shortcut for TCharacter.IsLowSurrogate (64)

See also: TCharacter.IsLowSurrogate (64)

2.4.12 IsNumber

Synopsis: Alias for `TCharacter.IsNumber`

Declaration:
```
function IsNumber(AChar: UnicodeChar) : Boolean;  Overload
function IsNumber(const AString: UnicodeString;AIndex: Integer)
                  : Boolean;  Overload
```

Visibility: default

Description: `IsNumber` is a shortcut for TCharacter.IsNumber (66)

See also: TCharacter.IsNumber (66)

2.4.13 IsPunctuation

Synopsis: Alias for `TCharacter.IsPunctuation`

Declaration:
```
function IsPunctuation(AChar: UnicodeChar) : Boolean;  Overload
function IsPunctuation(const AString: UnicodeString;AIndex: Integer)
                       : Boolean;  Overload
```

Visibility: default

Description: `IsPunctuation` is a shortcut for TCharacter.IsPunctuation (66)

See also: TCharacter.IsPunctuation (66)

2.4.14 IsSeparator

Synopsis: Alias for `TCharacter.IsSeparator`

Declaration:
```
function IsSeparator(AChar: UnicodeChar) : Boolean;  Overload
function IsSeparator(const AString: UnicodeString;AIndex: Integer)
                     : Boolean;  Overload
```

Visibility: default

Description: `IsSeparator` is a shortcut for TCharacter.IsSeparator (66)

See also: TCharacter.IsSeparator (66)

2.4.15 IsSurrogate

Synopsis: Alias for `TCharacter.IsSurrogate`

Declaration:
```
function IsSurrogate(AChar: UnicodeChar) : Boolean;  Overload
function IsSurrogate(const AString: UnicodeString;AIndex: Integer)
                     : Boolean;  Overload
```

Visibility: default

Description: `IsSurrogate` is a shortcut for TCharacter.IsSurrogate (63).

See also: TCharacter.IsSurrogate (63)

2.4.16 IsSurrogatePair

Synopsis: Alias for `TCharacter.IsSurrogatePair`

Declaration:
```
function IsSurrogatePair(const AHighSurrogate: UnicodeChar;
                         const ALowSurrogate: UnicodeChar) : Boolean
                         ;  Overload
function IsSurrogatePair(const AString: UnicodeString;AIndex: Integer)
                          : Boolean;  Overload
```

Visibility: default

Description: `IsSurrogatePair` is a shortcut for TCharacter.IsSurrogatePair (64)

See also: TCharacter.IsSurrogatePair (64)

2.4.17 IsSymbol

Synopsis: Alias for `TCharacter.IsSymbol`

Declaration:
```
function IsSymbol(AChar: UnicodeChar) : Boolean;  Overload
function IsSymbol(const AString: UnicodeString;AIndex: Integer)
                  : Boolean;  Overload
```

Visibility: default

Description: `IsSymbol` is a shortcut for TCharacter.IsSymbol (67)

See also: TCharacter.IsSymbol (67)

2.4.18 IsUpper

Synopsis: Alias for `TCharacter.IsUpper`

Declaration:
```
function IsUpper(AChar: UnicodeChar) : Boolean;  Overload
function IsUpper(const AString: UnicodeString;AIndex: Integer) : Boolean
                 ;  Overload
```

Visibility: default

Description: `IsUpper` is a shortcut for TCharacter.IsUpper (67)

See also: TCharacter.IsUpper (67)

2.4.19 IsWhiteSpace

Synopsis: Alias for `TCharacter.IsWhiteSpace`

Declaration:
```
function IsWhiteSpace(AChar: UnicodeChar) : Boolean;  Overload
function IsWhiteSpace(const AString: UnicodeString;AIndex: Integer)
                      : Boolean;  Overload
```

Visibility: default

Description: `IsWhiteSpace` is a shortcut for TCharacter.IsWhiteSpace (68)

See also: TCharacter.IsWhiteSpace (68)

2.4.20 ToLower

Synopsis: Alias for `TCharacter.ToLower`

Declaration:
```
function ToLower(AChar: UnicodeChar) : UnicodeChar;  Overload
function ToLower(const AString: UnicodeString) : UnicodeString
                ;  Overload
```

Visibility: default

Description: `ToLower` is a shortcut for TCharacter.ToLower (68)

See also: TCharacter.ToLower (68)

2.4.21 ToUpper

Synopsis: Alias for `TCharacter.ToUpper`

Declaration:
```
function ToUpper(AChar: UnicodeChar) : UnicodeChar;  Overload
function ToUpper(const AString: UnicodeString) : UnicodeString
                ;  Overload
```

Visibility: default

Description: `ToUpper` is a shortcut for TCharacter.ToUpper (68)

See also: TCharacter.ToUpper (68)

2.5 TCharacter

2.5.1 Description

`TCharacter` is provided for Delphi compatibility. All it's class functions and methods are also available as regular functions.

2.5.2 Method overview

Page	Method	Description
60	ConvertFromUtf32	Convert a UTF32 character to UnicodeString
61	ConvertToUtf32	Convert a UTF16 character to a UTF32 character
60	Create	Constructor (do not call)
61	GetNumericValue	Get the numeric value of the character
62	GetUnicodeCategory	Get the unicode category of a character
62	IsControl	Check whether a unicode character is a unicode control character
62	IsDigit	Check whether a unicode character is a digit
63	IsHighSurrogate	Check whether a unicode character is a surrogate in the high range
64	IsLetter	Check if a unicode character is a letter.
65	IsLetterOrDigit	Check if a unicode character is a letter or digit
65	IsLower	Check if a unicode character is a lowercase letter
64	IsLowSurrogate	Check whether a unicode character is a surrogate in the low range
66	IsNumber	Check if a unicode character is a number
66	IsPunctuation	Check if a unicode character is a punctuation character
66	IsSeparator	Check if a unicode character is a separator character
63	IsSurrogate	Check whether a unicode character is a surrogate
64	IsSurrogatePair	Check if a pair of characters is a set of high/low surrorgate characters
67	IsSymbol	Check if a unicode character is a symbol character
67	IsUpper	Check whether a unicode character is an uppercase letter
68	IsWhiteSpace	Check whether a unicode character is a whitespace character
68	ToLower	Convert a character or string to lowercase
68	ToUpper	Convert a character or string to uppercase

2.5.3 TCharacter.Create

Synopsis: Constructor (do not call)

Declaration: `constructor Create`

Visibility: public

Description: `Create` is provided for completeness and Delphi compatibility, but should not be called in FPC code, it will raise an exception.

Errors: Any attempt to call `Create` will result in an exception being raised.

2.5.4 TCharacter.ConvertFromUtf32

Synopsis: Convert a UTF32 character to UnicodeString

Declaration: `class function ConvertFromUtf32(AChar: UCS4Char);  Static`

Visibility: public

Description: `TCharacter.ConvertFromUtf32` converts a single UTF32 character `AChar` to a UTF16 string. This is the opposite of TCharacter.ConvertToUtf32 (61).

The result is a string, since multiple UTF16 characters can be needed to encode a single UTF32 character.

Errors: If `AChar` is not in the valid range of UTF32 characters, an EArgumentOutOfRangeException (52) exception is raised.

See also: EArgumentOutOfRangeException (52), TCharacter.ConvertToUtf32 (61)

2.5.5 TCharacter.ConvertToUtf32

Synopsis: Convert a UTF16 character to a UTF32 character

Declaration:
```
class function ConvertToUtf32(const AString: UnicodeString;
                              AIndex: Integer);  Overload;  Static
class function ConvertToUtf32(const AString: UnicodeString;
                              AIndex: Integer;out ACharLength: Integer)
                              ;  Overload;  Static
class function ConvertToUtf32(const AHighSurrogate: UnicodeChar;
                              const ALowSurrogate: UnicodeChar)
                              ;  Overload;  Static
```

Visibility: public

Description: `TCharacter.ConvertToUtf32` converts a UTF16-encoded unicode character to a Unicode32 character. This is the opposite of TCharacter.ConvertFromUtf32 (60). The function exists in several overloaded versions, to be able to present the unicode character in one of 2 ways:

1. As a position `AIndex` (in unicodechar units) in a string `AString` to a Unicode32 character. The source is a string, since multiple UTF16 characters can be needed to encode a single UTF32 character. In this form, Optionally, the character length (1 or 2) can be returned in `ACharLength`.
2. As 2 UTF16 unicode characters, representing the high and low surrogate pairs: `AHighSurrogate` and `ALowSurrogate`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an EArgumentOutOfRangeException (52) exception is raised. If the character at that position is not complete, an EArgumentException (52) exception is raised.

See also: TCharacter.ConvertFromUtf32 (60), EArgumentOutOfRangeException (52), EArgumentException (52)

2.5.6 TCharacter.GetNumericValue

Synopsis: Get the numeric value of the character

Declaration:
```
class function GetNumericValue(AChar: UnicodeChar);  Overload;  Static
class function GetNumericValue(const AString: UnicodeString;
                               AIndex: Integer);  Overload;  Static
```

Visibility: public

Description: `TCharacter.GetNumericValue` returns the numerical value (ID) of the unicode character. The character can be presented in 2 ways: `AChar`, a UTF16 unicode character, or a surrogate pair in a unicode string `AString` starting at position `AIndex`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an EArgumentOutOfRangeException (52) exception is raised. If the character at that position is not complete, an EArgumentException (52) exception is raised.

See also: TCharacter.GetUnicodeCategory (62)

2.5.7 TCharacter.GetUnicodeCategory

Synopsis: Get the unicode category of a character

Declaration:
```
class function GetUnicodeCategory(AChar: UnicodeChar);  Overload
                                  ;  Static
class function GetUnicodeCategory(const AString: UnicodeString;
                                  AIndex: Integer);  Overload;  Static
```

Visibility: public

Description: `TCharacter.GetUnicodeCategory` returns the unicode category of a character. The character can be presented in 2 ways: `AChar`, a UTF16 unicode character, or a surrogate pair in a unicode string `AString` starting at position `AIndex`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an EArgumentOutOfRangeException (52) exception is raised. If the character at that position is not complete, an EArgumentException (52) exception is raised.

See also: TUnicodeCategory (53)

2.5.8 TCharacter.IsControl

Synopsis: Check whether a unicode character is a unicode control character

Declaration:
```
class function IsControl(AChar: UnicodeChar);  Overload;  Static
class function IsControl(const AString: UnicodeString;AIndex: Integer)
                         ;  Overload;  Static
```

Visibility: public

Description: `IsControl` returns `True` if a unicode character has category `ucControl`. The character can be specified as a UTF16 character `AChar` or a UTF16 encoded character starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an EArgumentOutOfRangeException (52) exception is raised. If the character at that position is not complete, an EArgumentException (52) exception is raised.

See also: GetUnicodeCategory (62), IsDigit (62), IsSurrogate (63), IsHighSurrogate (63), IsLowSurrogate (64), IsSurrogatePair (64), IsLetter (64), IsLetterOrDigit (65), IsLower (65), IsNumber (66), IsPunctuation (66), IsSeparator (66), IsSymbol (67), IsUpper (67), IsWhiteSpace (68)

2.5.9 TCharacter.IsDigit

Synopsis: Check whether a unicode character is a digit

Declaration:
```
class function IsDigit(AChar: UnicodeChar);  Overload;  Static
class function IsDigit(const AString: UnicodeString;AIndex: Integer)
                       ;  Overload;  Static
```

Visibility: public

Description: `IsDigit` returns `True` if a unicode character has category `ucDecimalNumber`. The character can be specified as a UTF16 character `AChar` or a UTF16 encoded character starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an EArgumentOutOfRangeException (52) exception is raised. If the character at that position is not complete, an EArgumentException (52) exception is raised.

See also: IsControl (62), IsDigit (62), IsSurrogate (63), IsHighSurrogate (63), IsLowSurrogate (64), IsSurrogatePair (64), IsLetter (64), IsLetterOrDigit (65), IsLower (65), IsNumber (66), IsPunctuation (66), IsSeparator (66), IsSymbol (67), IsUpper (67), IsWhiteSpace (68)

2.5.10 TCharacter.IsSurrogate

Synopsis: Check whether a unicode character is a surrogate

Declaration:
```
class function IsSurrogate(AChar: UnicodeChar); Overload; Static
class function IsSurrogate(const AString: UnicodeString;AIndex: Integer)
                           ; Overload; Static
```

Visibility: public

Description: `IsSurrogate` returns `True` if a unicode character has category `ucSurrogate`. The character can be specified as a UTF16 character `AChar` or a UTF16 encoded character starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an EArgumentOutOfRangeException (52) exception is raised. If the character at that position is not complete, an EArgumentException (52) exception is raised.

See also: EArgumentException (52), IsControl (62), IsDigit (62), IsHighSurrogate (63), IsLowSurrogate (64), IsLetter (64), IsLetterOrDigit (65), IsLower (65), IsNumber (66), IsPunctuation (66), IsSymbol (67), IsUpper (67), IsWhiteSpace (68)

2.5.11 TCharacter.IsHighSurrogate

Synopsis: Check whether a unicode character is a surrogate in the high range

Declaration:
```
class function IsHighSurrogate(AChar: UnicodeChar); Overload; Static
class function IsHighSurrogate(const AString: UnicodeString;
                               AIndex: Integer); Overload; Static
```

Visibility: public

Description: `IsHighSurrogate` returns `True` if a unicode character has category `ucSurrogate` and is in the high range of the surrogate characters (between HIGH_SURROGATE_BEGIN and HIGH_SURROGATE_END). The character can be specified as a UTF16 character `AChar` or a UTF16 encoded character starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an EArgumentOutOfRangeException (52) exception is raised. If the character at that position is not complete, an EArgumentException (52) exception is raised.

See also: EArgumentException (52), IsControl (62), IsDigit (62), IsSurrogate (63), IsLowSurrogate (64), IsLetter (64), IsLetterOrDigit (65), IsLower (65), IsNumber (66), IsPunctuation (66), IsSymbol (67), IsUpper (67), IsWhiteSpace (68)

2.5.12 TCharacter.IsLowSurrogate

Synopsis: Check whether a unicode character is a surrogate in the low range

Declaration:
```
class function IsLowSurrogate(AChar: UnicodeChar);  Overload;  Static
class function IsLowSurrogate(const AString: UnicodeString;
                              AIndex: Integer);  Overload;  Static
```

Visibility: public

Description: `IsLowSurrogate` returns `True` if a unicode character has category `ucSurrogate` and is in the low range of the surrogate characters (between LOW_SURROGATE_BEGIN and LOW_SURROGATE_END). The character can be specified as a UTF16 character `AChar` or a UTF16 encoded character starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an EArgumentOutOfRangeException (52) exception is raised. If the character at that position is not complete, an EArgumentException (52) exception is raised.

See also: EArgumentException (52), IsControl (62), IsDigit (62), IsSurrogate (63), IsHighSurrogate (63), IsLetter (64), IsLetterOrDigit (65), IsLower (65), IsNumber (66), IsPunctuation (66), IsSymbol (67), IsUpper (67), IsWhiteSpace (68)

2.5.13 TCharacter.IsSurrogatePair

Synopsis: Check if a pair of characters is a set of high/low surrorgate characters

Declaration:
```
class function IsSurrogatePair(const AHighSurrogate: UnicodeChar;
                               const ALowSurrogate: UnicodeChar)
                               ;  Overload;  Static
class function IsSurrogatePair(const AString: UnicodeString;
                               AIndex: Integer);  Overload;  Static
```

Visibility: public

Description: `IsSurrogatePair` returns `True` if `AHighSurrogate` and `ALowSurrogate` form a valid unicode surrogate pair. (`AHighSurrogate` is a high surrogate and `ALowSurrogate` a matching low surrogate) The character can be specified as a UTF16 character `AChar` or a pair of UTF16 encoded characters starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an EArgumentOutOfRangeException (52) exception is raised. If the character at that position is not complete, an EArgumentException (52) exception is raised.

See also: EArgumentException (52), IsControl (62), IsDigit (62), IsSurrogate (63), IsHighSurrogate (63), IsLowSurrogate (64), IsLetter (64), IsLetterOrDigit (65), IsLower (65), IsNumber (66), IsPunctuation (66), IsSymbol (67), IsUpper (67), IsWhiteSpace (68)

2.5.14 TCharacter.IsLetter

Synopsis: Check if a unicode character is a letter.

Declaration:
```
class function IsLetter(AChar: UnicodeChar);  Overload;  Static
class function IsLetter(const AString: UnicodeString;AIndex: Integer)
                        ;  Overload;  Static
```

Visibility: public

Description: `IsLetter` returns `True` if a unicode character has category that is one of the letter categories (`ucUppercaseLetter`, `ucLowercaseLetter`, `ucTitlecaseLetter`, `ucModifierLetter`, `ucOtherLetter`). The character can be specified as a UTF16 character `AChar` or a UTF16 encoded character starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an EArgumentOutOfRangeException (52) exception is raised. If the character at that position is not complete, an

See also: EArgumentException (52), IsControl (62), IsDigit (62), IsSurrogate (63), IsHighSurrogate (63), IsLowSurrogate (64), IsSurrogatePair (64), IsLetter (64), IsLetterOrDigit (65), IsLower (65), IsNumber (66), IsPunctuation (66), IsSymbol (67), IsUpper (67), IsWhiteSpace (68)

2.5.15 TCharacter.IsLetterOrDigit

Synopsis: Check if a unicode character is a letter or digit

Declaration:
```
class function IsLetterOrDigit(AChar: UnicodeChar); Overload; Static
class function IsLetterOrDigit(const AString: UnicodeString;
                               AIndex: Integer); Overload; Static
```

Visibility: public

Description: `IsLetterOrDigit` returns `True` if a unicode character has category that is one of the letter categories (`ucUppercaseLetter`, `ucLowercaseLetter`, `ucTitlecaseLetter`, `ucModifierLetter`, `ucOtherLetter`,`ucDecimalNumber`,`ucLetterNumber`). The character can be specified as a UTF16 character `AChar` or a UTF16 encoded character starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an EArgumentOutOfRangeException (52) exception is raised. If the character at that position is not complete, an EArgumentException (52) exception is raised.

See also: IsControl (62), IsDigit (62), IsSurrogate (63), IsHighSurrogate (63), IsLowSurrogate (64), IsSurrogatePair (64), IsLetter (64), IsLower (65), IsNumber (66), IsPunctuation (66), IsSymbol (67), IsUpper (67), IsWhiteSpace (68)

2.5.16 TCharacter.IsLower

Synopsis: Check if a unicode character is a lowercase letter

Declaration:
```
class function IsLower(AChar: UnicodeChar); Overload; Static
class function IsLower(const AString: UnicodeString;AIndex: Integer)
                       ; Overload; Static
```

Visibility: public

Description: `IsLower` returns `True` if a unicode character has category `ucLowercaseLetter`. The character can be specified as a UTF16 character `AChar` or a UTF16 encoded character starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an EArgumentOutOfRangeException (52) exception is raised. If the character at that position is not complete, an EArgumentException (52) exception is raised.

See also: IsControl (62), IsDigit (62), IsSurrogate (63), IsHighSurrogate (63), IsLowSurrogate (64), IsSurrogatePair (64), IsLetter (64), IsLetterOrDigit (65), IsNumber (66), IsPunctuation (66), IsSymbol (67), IsUpper (67), IsWhiteSpace (68)

2.5.17 TCharacter.IsNumber

Synopsis: Check if a unicode character is a number

Declaration:
```
class function IsNumber(AChar: UnicodeChar); Overload; Static
class function IsNumber(const AString: UnicodeString;AIndex: Integer)
                        ; Overload; Static
```

Visibility: public

Description: `IsNumber` returns `True` if a unicode character has category that is one of the number categories (`ucDecimalNumber`, `ucLetterNumber`, `ucOtherNumber`). The character can be specified as a UTF16 character `AChar` or a UTF16 encoded character starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an EArgumentOutOfRangeException (52) exception is raised. If the character at that position is not complete, an EArgumentException (52) exception is raised.

See also: IsControl (62), IsDigit (62), IsSurrogate (63), IsHighSurrogate (63), IsLowSurrogate (64), IsSurrogatePair (64), IsLetter (64), IsLetterOrDigit (65), IsLower (65), IsNumber (66), IsPunctuation (66), IsSymbol (67), IsUpper (67), IsWhiteSpace (68)

2.5.18 TCharacter.IsPunctuation

Synopsis: Check if a unicode character is a punctuation character

Declaration:
```
class function IsPunctuation(AChar: UnicodeChar); Overload; Static
class function IsPunctuation(const AString: UnicodeString;
                             AIndex: Integer); Overload; Static
```

Visibility: public

Description: `IsPunctuation` returns `True` if a unicode character has category that is one of the punctuation categories (`ucConnectPunctuation`, `ucDashPunctuation`, `ucOpenPunctuation`, `ucClosePunctuation`, `ucInitialPunctuation`, `ucFinalPunctuation`, `ucOtherPunctuation`). The character can be specified as a UTF16 character `AChar` or a UTF16 encoded character starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an EArgumentOutOfRangeException (52) exception is raised. If the character at that position is not complete, an EArgumentException (52) exception is raised.

See also: IsControl (62), IsDigit (62), IsSurrogate (63), IsHighSurrogate (63), IsLowSurrogate (64), IsSurrogatePair (64), IsLetter (64), IsLetterOrDigit (65), IsLower (65), IsNumber (66), IsSymbol (67), IsUpper (67), IsWhiteSpace (68)

2.5.19 TCharacter.IsSeparator

Synopsis: Check if a unicode character is a separator character

Declaration:
```
class function IsSeparator(AChar: UnicodeChar); Overload; Static
class function IsSeparator(const AString: UnicodeString;AIndex: Integer)
                           ; Overload; Static
```

Visibility: public

Description: `IsSeparator` returns `True` if a unicode character has category that is one of the separator categories (`ucSpaceSeparator`, `ucLineSeparator`, `ucParagraphSeparator`). The character can be specified as a UTF16 character `AChar` or a UTF16 encoded character starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an EArgumentOutOfRangeException (52) exception is raised. If the character at that position is not complete, an EArgumentException (52) exception is raised.

See also: IsControl (62), IsDigit (62), IsSurrogate (63), IsHighSurrogate (63), IsLowSurrogate (64), IsSurrogatePair (64), IsLetter (64), IsLetterOrDigit (65), IsLower (65), IsNumber (66), IsPunctuation (66), IsSymbol (67), IsUpper (67), IsWhiteSpace (68)

2.5.20 TCharacter.IsSymbol

Synopsis: Check if a unicode character is a symbol character

Declaration:
```
class function IsSymbol(AChar: UnicodeChar);  Overload;  Static
class function IsSymbol(const AString: UnicodeString;AIndex: Integer)
                        ;  Overload;  Static
```

Visibility: public

Description: `IsSymbol` returns `True` if a unicode character has category that is one of the symbol categories (`ucMathSymbol`, `ucCurrencySymbol`, `ucModifierSymbol`, `ucOtherSymbol`). The character can be specified as a UTF16 character `AChar` or a UTF16 encoded character starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an EArgumentOutOfRangeException (52) exception is raised. If the character at that position is not complete, an EArgumentException (52) exception is raised.

See also: IsControl (62), IsDigit (62), IsSurrogate (63), IsHighSurrogate (63), IsLowSurrogate (64), IsSurrogatePair (64), IsLetter (64), IsLetterOrDigit (65), IsLower (65), IsNumber (66), IsPunctuation (66), IsSeparator (66), IsUpper (67), IsWhiteSpace (68)

2.5.21 TCharacter.IsUpper

Synopsis: Check whether a unicode character is an uppercase letter

Declaration:
```
class function IsUpper(AChar: UnicodeChar);  Overload;  Static
class function IsUpper(const AString: UnicodeString;AIndex: Integer)
                       ;  Overload;  Static
```

Visibility: public

Description: `IsUpper` returns `True` if a unicode character has category `ucUppercaseLetter`. The character can be specified as a UTF16 character `AChar` or a UTF16 encoded character starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an EArgumentOutOfRangeException (52) exception is raised. If the character at that position is not complete, an EArgumentException (52) exception is raised.

See also: IsControl (62), IsDigit (62), IsSurrogate (63), IsHighSurrogate (63), IsLowSurrogate (64), IsSurrogatePair (64), IsLetter (64), IsLetterOrDigit (65), IsLower (65), IsNumber (66), IsPunctuation (66), IsSeparator (66), IsSymbol (67), IsWhiteSpace (68)

2.5.22 TCharacter.IsWhiteSpace

Synopsis: Check whether a unicode character is a whitespace character

Declaration:
```
class function IsWhiteSpace(AChar: UnicodeChar); Overload; Static
class function IsWhiteSpace(const AString: UnicodeString;
                            AIndex: Integer); Overload; Static
```

Visibility: public

Description: `IsUpper` returns `True` if a unicode character has is a whitespace character. It checks the character properties. The character can be specified as a UTF16 character `AChar` or a UTF16 encoded character starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an EArgumentOutOfRangeException (52) exception is raised. If the character at that position is not complete, an EArgumentException (52) exception is raised.

See also: IsControl (62), IsDigit (62), IsSurrogate (63), IsHighSurrogate (63), IsLowSurrogate (64), IsSurrogatePair (64), IsLetter (64), IsLetterOrDigit (65), IsLower (65), IsNumber (66), IsPunctuation (66), IsSeparator (66), IsSymbol (67), IsUpper (67)

2.5.23 TCharacter.ToLower

Synopsis: Convert a character or string to lowercase

Declaration:
```
class function ToLower(AChar: UnicodeChar); Overload; Static
class function ToLower(const AString: UnicodeString); Overload; Static
class function ToLower(const AString: UnicodeString;
                       const AOptions: TCharacterOptions); Overload
                       ; Static
```

Visibility: public

Description: `ToLower` converts the unicode character `AChar` or string `AString` to lowercase. `Options` determines the behaviour of the conversion: if `AOptions` contains `coIgnoreInvalidSequence` then no exception will be raised when the string or character contains an invalid unicode sequence. The default behaviour is to raise an EArgumentException (52) exception when this happens.

Errors: If an invalid character is encountered, an EArgumentException (52) exception is raised, unless `coIgnoreInvalidSequence` is specified in the options.

See also: TCharacter.ToUpper (68), TCharacter.IsLower (65), TCharacter.IsUpper (67)

2.5.24 TCharacter.ToUpper

Synopsis: Convert a character or string to uppercase

Declaration:
```
class function ToUpper(AChar: UnicodeChar); Overload; Static
class function ToUpper(const AString: UnicodeString); Overload; Static
class function ToUpper(const AString: UnicodeString;
                       const AOptions: TCharacterOptions); Overload
                       ; Static
```

Visibility: public

Description: `ToUpper` converts the unicode character `AChar` or string `AString` to uppercase. `Options` determines the behaviour of the conversion: if `AOptions` contains `coIgnoreInvalidSequence` then no exception will be raised when the string or character contains an invalid unicode sequence. The default behaviour is to raise an EArgumentException (52) exception when this happens.

Errors: If an invalid character is encountered, an EArgumentException (52) exception is raised, unless `coIgnoreInvalidSequence` is specified in the options.

See also: TCharacter.ToUpper (68), TCharacter.IsLower (65), TCharacter.IsUpper (67)

Chapter 3

Reference for unit 'charset'

3.1 Overview

The charset unit can be used to load single-byte character set (code page) descriptions. It is used in the fpwidestring (70) unit to add support for converting single-byte codepage strings to unicode strings (and vice versa).

Data of a code page may be included using one of the ready-made units, or can be loaded (in a binary form) at runtime with the loadbinaryunicodemapping (73) function. The binary files have the .bcm extension and are produced by the creumap utility distributed with Free Pascal.

Pre-made units are available for the following codepages: cp895 (70), cp932 (70), cp936 (70), cp949 (70) and cp950 (70).

3.2 Constants, types and variables

3.2.1 Constants

`BINARY_MAPPING_FILE_EXT = '.bcm'`

`BINARY_MAPPING_FILE_EXT` contains the default extension of a file containing a binary-coded map.

3.2.2 Types

`preversecharmapping = ^treversecharmapping`

Pointer to `treversecharmapping`

`punicodecharmapping = ^tunicodecharmapping`

Pointer to `tunicodecharmapping`

`punicodemap = ^tunicodemap`

Pointer to `tunicodemap`

```
treversecharmapping = packed record
  unicode : tunicodechar;
  char1 : Byte;
  char2 : Byte;
end
```

treversecharmapping describes how a unicode character can be created in terms of single-byte characters.

```
TSerializedMapHeader = packed record
  cpName : string;
  cp : UInt16;
  mapLength : UInt32;
  lastChar : Int32;
  reverseMapLength : UInt32;
end
```

TSerializedMapHeader is a record describing the binary map data file. The contents of this record can be found at offset zero of a (.bcm) file containing a single-byte unicode map.

```
tunicodechar = Word
```

tunicodechar is a type used to represent unicode characters in this file, it should not be used for other unicode routines.

```
tunicodecharmapping = packed record
  unicode : tunicodechar;
  flag : tunicodecharmappingflag;
  reserved : Byte;
end
```

tunicodecharmapping describes a unicode character. An array of these mappings is built for each character in the single-byte character set,

```
tunicodecharmappingflag = (umf_noinfo,umf_leadbyte,umf_undefined,
                           umf_unused)
```

Table 3.1: Enumeration values for type tunicodecharmappingflag

Value	Explanation
umf_leadbyte	Unicode character uses leading byte
umf_noinfo	No extra information about unicode character
umf_undefined	Currently unused
umf_unused	Unused position in code page

tunicodecharmappingflag contains various Flags describing information about a unicode character.

```
tunicodemap = record
  cpname : string;
  cp : Word;
  map : punicodecharmapping;
  lastchar : LongInt;
  reversemap : preversecharmapping;
  reversemaplength : LongInt;
  next : punicodemap;
  internalmap : Boolean;
end
```

`tunicodemap` describes a complete mapping between a single-byte code page and a unicode character set. It contains both a forward and backward mapping.

```
tunicodestring = ^tunicodechar
```

`tunicodestring` is a type used to represent unicode strings in this file, it should not be used for other unicode routines.

3.3 Procedures and functions

3.3.1 getascii

Synopsis: Convert unicode character or string to single-byte character or string.

Declaration:
```
function getascii(c: tunicodechar;p: punicodemap) : string
function getascii(c: tunicodechar;p: punicodemap;ABuffer: PAnsiChar;
                  ABufferLen: LongInt) : LongInt
```

Visibility: default

Description: `getascii` converts a unicode character `c` to one or more single-byte characters according to the map in `p`. The result can be a string containing up to 2 characters, or the number of characters copied to the buffer `ABuffer` with length `ABufferLen`.

Errors: If the character cannot be translated, ASCII character 63 is returned (or copied to the buffer). In the case of the buffer variant of the function, -1 is then returned. If the buffer is not large enough, -1 is returned.

See also: getunicode (73)

3.3.2 getmap

Synopsis: Find a codepage map

Declaration:
```
function getmap(const s: string) : punicodemap
function getmap(cp: Word) : punicodemap
```

Visibility: default

Description: `getmap` looks in the registered codepage mappings and returns the mapping for the requested codepage. The codepage can be specified using a name `s` or a numerical identifier `cp`. The search is case sensitive.

Errors: if the requested map is not found, `Nil` is returned.

See also: registermapping (74), registerbinarymapping (74), mappingavailable (74)

3.3.3 getunicode

Synopsis: Map single-byte character to unicode character.

Declaration:
```
function getunicode(c: Char;p: punicodemap) : tunicodechar
function getunicode(AAnsiStr: pansichar;AAnsiLen: LongInt;
                    AMap: punicodemap;ADest: tunicodestring) : LongInt
```

Visibility: default

Description: The first form of `getunicode` will map a single character `c` to its unicode equivalent for mapping `p`. If no equivalent can be found, `0` is returned.

The second form of `getunicode` will transform a string (specified using a pointer `AAnsiStr` to a buffer with length `AAnsiLen`) to a unicode string using single byte codepage map `AMap`. It returns the number of unicode characters. If `ADest` is `Nil` then just the number of characters is returned. If `ADest` is not `nil`, it must point to a buffer large enough to contain the unicode string, and the converted string will be copied to it.

Errors: No checking on the validity of the buffers is done.

See also: getascii (72)

3.3.4 loadbinaryunicodemapping

Synopsis: Load binary single-byte codepage to unicode map from file or memory

Declaration:
```
function loadbinaryunicodemapping(const directory: string;
                                  const cpname: string) : punicodemap
                                  ;  Overload
function loadbinaryunicodemapping(const filename: string) : punicodemap
                                  ;  Overload
function loadbinaryunicodemapping(const AData: Pointer;
                                  const ADataLength: Integer)
                                   : punicodemap;  Overload
```

Visibility: default

Description: `loadbinaryunicodemapping` loads a binary description of a single-byte unicode mapping. The mapping can reside in a file, in which case the file to load can be specified using a filename `filename` or using a directory `directory` and codepage name `cpname`. In the latter case, a suffix consisting of _le or _be depending on the endianness of the current platform will be appended, and the filename extension is .bcm. Note that the file names may be case sensitive.

The data can also be loaded from a memory block `AData` with size `ADataLength`.

It will produce an in-memory map of the file. It returns a pointer to the map, or `Nil` if something went wrong. The resulting mapping can be registered using registermapping (74).

Errors: On error, `Nil` is returned.

See also: loadunicodemapping (73), registermapping (74)

3.3.5 loadunicodemapping

Synopsis: Load textual single-byte codepage to unicode map from file

Declaration:
```
function loadunicodemapping(const cpname: string;const f: string;
                            cp: Word) : punicodemap
```

Visibility: default

Description: `loadunicodemapping` loads a text description of a single-byte unicode mapping. It will analyse the textual description in file `f`, and produce an in-memory map of the file. It returns a pointer to the map, or `Nil` if something went wrong. The unicode map name must be specified in `cpName`, and the numerical identifier in `cp`

The resulting mapping can be registered using registermapping (74).

Errors: On error, `Nil` is returned.

See also: loadbinaryunicodemapping (73), registermapping (74)

3.3.6 mappingavailable

Synopsis: Check if a mapping is available for a specified code page.

Declaration:
```
function mappingavailable(const s: string) : Boolean
function mappingavailable(cp: Word) : Boolean
```

Visibility: default

Description: `mappingavailable` returns `True` if a mapping for a specified code page (using name `s` or numerical identifier `cp`) is available, or `False` if it is not.

See also: registermapping (74), registerbinarymapping (74), getmap (72)

3.3.7 registerbinarymapping

Synopsis: Load and register binary single-byte codepage to unicode map from file

Declaration:
```
function registerbinarymapping(const directory: string;
                               const cpname: string) : Boolean
```

Visibility: default

Description: `registerbinarymapping` calls loadbinaryunicodemapping (73) using `directory` and `cpname` and registers the resulting mapping, if any was succesfully loaded, using registermapping (74). It returns `True` if the operation was succesful.

Errors: On error, `False` is returned.

See also: loadbinaryunicodemapping (73), registermapping (74)

3.3.8 registermapping

Synopsis: Register mapping

Declaration:
```
procedure registermapping(p: punicodemap)
```

Visibility: default

Description: `RegisterMapping` registers mapping `p` in the registry of single-byte codepages. No attempt is made to avoid double registrations. In case of doubles, the last registered mapping will be used first.

See also: loadunicodemapping (73), loadbinaryunicodemapping (73), registerbinarymapping (74)

Chapter 4

Reference for unit 'Classes'

4.1 Used units

Table 4.1: Used units by unit 'Classes'

Name	Page
rtlconsts	??
System	622
sysutils	628
Types	637
typinfo	656

4.2 Overview

This documentation describes the FPC classes unit. The Classes unit contains basic classes for the Free Component Library (FCL):

- a TList (75) class for maintaining lists of pointers,
- TStringList (75) for lists of strings,
- TCollection (75) to manage collections of objects
- TStream (75) classes to support streaming.

Furthermore it introduces methods for object persistence, and classes that understand an owner-owned relationship, with automatic memory management.

Chapter 5

Reference for unit 'clocale'

5.1 Overview

The `clocale` offers no API by itself: it just initializes the internationalization settings of the sysutils (628) unit with the values provided by the C library found on most Unix or Linux systems that are POSIX compliant.

The clocale should simply be included in the uses clause of the program, preferably as one of the first units, and the initialization section of the unit will do all the work.

Note that including this unit, links your program to the C library of the system.

It makes no sense to use this unit on a non-posix system: Windows, OS/2 or DOS - therefore it should always be between an ifdef statement:

```
program myprogram;

uses
  {$ifdef unix}clocale{$endif},
   classes, sysutils;
```

Chapter 6

Reference for unit 'cmem'

6.1 Overview

The cmem memory manager sets the system units memory manager to a C-based memory manager: all memory management calls are shunted through to the C memory manager, using Malloc (78), Free (77) and ReAlloc (78). For this reason, the cmem unit should be the first unit of the uses clause of the program.

The unit also offers the C memory calls directly as external declarations from the C library, but it is recommended to use the normal FPC routines for this.

Obviously, including this unit links your program to the C library.

6.2 Constants, types and variables

6.2.1 Constants

```
LibName = 'c'
```

`LibName` is the name of the library that is actually used. On most systems, this is simply "libc.so".

6.3 Procedures and functions

6.3.1 CAlloc

Synopsis: Allocate memory based on item size and count

Declaration: `function CAlloc(unitSize: ptruint;UnitCount: ptruint) : pointer`

Visibility: default

Description: `Calloc` allocates memory to hold `UnitCount` units of size `UnitSize` each. The memory is one block of memory. It returns a pointer to the newly allocated memory block.

See also: Malloc (78), Free (77), Realloc (78)

6.3.2 Free

Synopsis: Free a previously allocated block

Declaration: `procedure Free(P: pointer)`

Visibility: default

Description: `Free` returns the memory block pointed to by `P` to the system. After `Free` was called, the pointer `P` is no longer valid.

See also: Malloc (78), ReAlloc (78)

6.3.3 Malloc

Synopsis: Malloc external declaration.

Declaration: `function Malloc(Size: ptruint) : Pointer`

Visibility: default

Description: `Malloc` is the external declaration of the C librarys `malloc` call. It accepts a size parameter, and returns a pointer to a memory block of the requested size or `Nil` if no more memory could be allocated.

See also: Free (77), ReAlloc (78)

6.3.4 ReAlloc

Synopsis: Reallocates a memory block

Declaration: `function ReAlloc(P: Pointer;Size: ptruint) : pointer`

Visibility: default

Description: `ReAlloc` re-allocates a block of memory pointed to by `p`. The new block will have size `Size`, and as much data as was available or as much data as fits is copied from the old to the new location.

See also: Malloc (78), Free (77)

Chapter 7

Reference for unit 'collation_de'

7.1 Overview

The collation_de unit registers the German Unicode collation (de). This collation bases itself on the DUCET collation, so that collation will be included as well.

This unit does not contain any routines. It simply registers the collation in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 8

Reference for unit 'collation_es'

8.1 Overview

The collation_es unit registers the Spanish Unicode collation (de). This collation bases itself on the DUCET collation, so that collation will be included as well.

This unit does not contain any routines. It simply registers the collation in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 9

Reference for unit 'collation_fr_ca'

9.1 Overview

The collation_fr_ca unit registers the French Unicode collation (fr). This collation bases itself on the DUCET collation, so that collation will be included as well.

This unit does not contain any routines. It simply registers the collation in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 10

Reference for unit 'collation_ja'

10.1 Overview

The collation_ja unit registers the Japanese Unicode collation (ja). This collation bases itself on the DUCET collation, so that collation will be included as well.

This unit does not contain any routines. It simply registers the collation in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 11

Reference for unit 'collation_ko'

11.1 Overview

The collation_ko unit registers the Korean Unicode collation (ko). This collation bases itself on the DUCET collation, so that collation will be included as well.

This unit does not contain any routines. It simply registers the collation in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 12

Reference for unit 'collation_ru'

12.1 Overview

The collation_ru unit registers the Russian Unicode collation (ru). This collation bases itself on the DUCET collation, so that collation will be included as well.

This unit does not contain any routines. It simply registers the collation in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 13

Reference for unit 'collation_sv'

13.1 Overview

The collation_sv unit registers the Swedish Unicode collation (sv). This collation bases itself on the DUCET collation, so that collation will be included as well.

This unit does not contain any routines. It simply registers the collation in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 14

Reference for unit 'collation_zh'

14.1 Overview

The collation_zh unit registers the Chinese Unicode collation (zh). This collation bases itself on the DUCET collation, so that collation will be included as well.

This unit does not contain any routines. It simply registers the collation in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 15

Reference for unit 'cp1250'

15.1 Overview

The `cp1250` unit registers single-byte codepage 1250. This is necessary to convert single-byte strings using codepage 1250 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 16

Reference for unit 'cp1251'

16.1 Overview

The `cp1251` unit registers single-byte codepage 1251. This is necessary to convert single-byte strings using codepage 1251 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 17

Reference for unit 'cp1252'

17.1 Overview

The `cp1252` unit registers single-byte codepage 1252. This is necessary to convert single-byte strings using codepage 1252 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 18

Reference for unit 'cp1253'

18.1 Overview

The `cp1253` unit registers single-byte codepage 1253. This is necessary to convert single-byte strings using codepage 1253 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 19

Reference for unit 'cp1254'

19.1 Overview

The `cp1254` unit registers single-byte codepage 1254. This is necessary to convert single-byte strings using codepage 1254 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 20

Reference for unit 'cp1255'

20.1 Overview

The `cp1255` unit registers single-byte codepage 1255. This is necessary to convert single-byte strings using codepage 1255 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 21

Reference for unit 'cp1256'

21.1 Overview

The `cp1256` unit registers single-byte codepage 1256. This is necessary to convert single-byte strings using codepage 1256 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 22

Reference for unit 'cp1257'

22.1 Overview

The `cp1257` unit registers single-byte codepage 1257. This is necessary to convert single-byte strings using codepage 1257 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 23

Reference for unit 'cp1258'

23.1 Overview

The `cp1258` unit registers single-byte codepage 1258. This is necessary to convert single-byte strings using codepage 1258 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 24

Reference for unit 'cp437'

24.1 Overview

The `cp437` unit registers single-byte codepage 437. This is necessary to convert single-byte strings using codepage 437 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 25

Reference for unit 'cp646'

25.1 Overview

The `cp646` unit registers single-byte codepage 646. This is necessary to convert single-byte strings using codepage 646 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 26

Reference for unit 'cp850'

26.1 Overview

The `cp850` unit registers single-byte codepage 850. This is necessary to convert single-byte strings using codepage 850 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 27

Reference for unit 'cp852'

27.1 Overview

The `cp852` unit registers single-byte codepage 852. This is necessary to convert single-byte strings using codepage 852 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 28

Reference for unit 'cp856'

28.1 Overview

The `cp856` unit registers single-byte codepage 856. This is necessary to convert single-byte strings using codepage 856 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 29

Reference for unit 'cp866'

29.1 Overview

The `cp866` unit registers single-byte codepage 866. This is necessary to convert single-byte strings using codepage 866 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 30

Reference for unit 'cp874'

30.1 Overview

The `cp874` unit registers single-byte codepage 874. This is necessary to convert single-byte strings using codepage 874 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 31

Reference for unit 'cp8859_1'

31.1 Overview

The `cp8859_1` unit registers single-byte codepage 8859-1. This is necessary to convert single-byte strings using codepage 8859-1 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 32

Reference for unit 'cp8859_2'

32.1 Overview

The `cp8859_2` unit registers single-byte codepage 8859-2. This is necessary to convert single-byte strings using codepage 8859-2 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 33

Reference for unit 'cp8859_5'

33.1 Overview

The `cp8859_5` unit registers single-byte codepage 8859-5. This is necessary to convert single-byte strings using codepage 8859-5 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 34

Reference for unit 'cp895'

34.1 Overview

The `cp895` unit registers single-byte codepage 895. This is necessary to convert single-byte strings using codepage 895 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 35

Reference for unit 'cp932'

35.1 Overview

The `cp932` unit registers single-byte codepage 932. This is necessary to convert single-byte strings using codepage 932 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 36

Reference for unit 'cp936'

36.1 Overview

The `cp936` unit registers single-byte codepage 936. This is necessary to convert single-byte strings using codepage 936 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 37

Reference for unit 'cp949'

37.1 Overview

The `cp949` unit registers single-byte codepage 949. This is necessary to convert single-byte strings using codepage 949 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 38

Reference for unit 'cp950'

38.1 Overview

The `cp950` unit registers single-byte codepage 950. This is necessary to convert single-byte strings using codepage 950 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 39

Reference for unit 'cpall'

39.1 Used units

Table 39.1: Used units by unit 'cpall'

Name	Page
cp1250	87
cp1251	88
cp1252	89
cp1253	90
cp1254	91
cp1255	92
cp1256	93
cp1257	94
cp1258	95
cp437	96
cp646	97
cp850	98
cp852	99
cp856	100
cp866	101
cp874	102
cp8859_1	103
cp8859_2	104
cp8859_5	105
System	622

39.2 Overview

The `cpall` unit registers all known single-byte codepages: 1251 (111), 866 (111), ISO 8856-5 (111) (cyrillic), 8859-1 (111), 8859-2 (111), 1253 (111) (greek), 850 (111), 437 (111), 1252 (111), 646 (111), 874 (111), 856 (111), 1250 (111), 1254 (111), 1255 (111), 1256 (111), 1257 (111), 1258 (111), 852 (111).

This unit does not contain any routines. It simply uses the other units so all corresponding code pages are registered.

Chapter 40

Reference for unit 'Crt'

40.1 Overview

This chapter describes the CRT unit for Free Pascal, both under dos linux and Windows. The unit was first written for dos by Florian klaempfl. The unit was ported to linux by Mark May and enhanced by Michael Van Canneyt and Peter Vreman. It works on the linux console, and in xterm and rxvt windows under X-Windows. The functionality for both is the same, except that under linux the use of an early implementation (versions 0.9.1 and earlier of the compiler) the crt unit automatically cleared the screen at program startup.

There are some caveats when using the CRT unit:

- Programs using the CRT unit will *not* be usable when input/output is being redirected on the command-line.
- For similar reasons they are not usable as CGI-scripts for use with a webserver.
- The use of the CRT unit and the graph unit may not always be supported.
- The CRT unit is not thread safe.
- On linux or other unix OSes , executing other programs that expect special terminal behaviour (using one of the special functions in the linux unit) will not work. The terminal is set in RAW mode, which will destroy most terminal emulation settings.
- The CRT unit stems from the TP/Dos area. It is designed to work with single-byte character sets, where 1 char = 1 byte. That means that widestrings or UTF-8 encoded (ansi)strings will not correctly work.

Chapter 41

Reference for unit 'cthreads'

41.1 Overview

The CThreads unit initializes the system unit's thread management routines with an implementation based on the POSIX thread managing routines in the C library. This assures that C libraries that are thread-aware still work if they are linked to by a FPC program.

It doesn't offer any API by itself: the initialization section of the unit just initializes the `ThreadManager` record in the System (622) unit. This is done using the SetCThreadManager (113) call

The cthreads unit simply needs to be included in the uses clause of the program, preferably the very first unit, and the initialization section of the unit will do all the work.

Note that including this unit links your program to the C library of the system.

It makes no sense to use this unit on a non-posix system: Windows, OS/2 or DOS, therefor it should always between an ifdef statement:

```
program myprogram;

uses
  {$ifdef unix}cthreads{$endif},
   classes, sysutils;
```

The Lazarus IDE inserts this conditional automatically for each new started program.

41.2 Procedures and functions

41.2.1 SetCThreadManager

Synopsis: Sets the thread manager to the C thread manager

Declaration: `procedure SetCThreadManager`

Visibility: default

Description: `SetCThreadManager` actually sets the thread manager to the C thread manager. It can be called to re-set the thread manager if the thread manager was set to some other thread manager during the life-time of the program.

Chapter 42

Reference for unit 'ctypes'

42.1 Used units

Table 42.1: Used units by unit 'ctypes'

Name	Page
System	622
unixtype	747

42.2 Overview

The ctypes unit contains the definitions of commonly found C types. It can be used when interfaces to C libraries need to be defined. The types here are correct on all platforms, 32 or 64 bit.

The main advantage of using this file is to make sure that all C header import units use the same definitions for basic C types.

The h2pas program can include the ctypes unit automatically in the units it generates. The `-C` command-line switch can be used for this.

Chapter 43

Reference for unit 'cwstring'

43.1 Overview

The `cwstring` unit offers no API by itself: it just initializes the widestring manager record of the system (622) unit with an implementation that uses collation and conversion routines which are provided by the C library found on most Unix or Linux systems that are POSIX compliant.

The cwstring should simply be included in the uses clause of the program, preferably as one of the first units, and the initialization section of the unit will do all the work.

Note that including this unit links your program to the C library of the system.

It makes no sense to use this unit on a non-POSIX system like Windows, OS/2 or DOS. Therefor it should always be enclosed with an ifdef statement:

```
program myprogram;

uses
  {$ifdef unix}cwstring,{$endif}
   classes, sysutils;
```

43.2 Procedures and functions

43.2.1 SetCWidestringManager

Synopsis: Set the Widestring manager of the system unit to the C version

Declaration: `procedure SetCWidestringManager`

Visibility: default

Description: `SetCWidestringManager` actually sets the widestring manager record of the system unit. It is called automatically by the initialization section of the unit.

Chapter 44

Reference for unit 'dateutils'

44.1 Used units

Table 44.1: Used units by unit 'dateutils'

Name	Page
math	371
System	622
sysutils	628

44.2 Overview

DateUtils contains a large number of date/time manipulation routines, all based on the `TDateTime` type. There are routines for date/time math, for comparing dates and times, for composing dates and decomposing dates in their constituent parts.

44.3 Constants, types and variables

44.3.1 Constants

```
ApproxDaysPerMonth : Double = 30.4375
```

Average number of days in a month, measured over a year. Used in MonthsBetween (163).

```
ApproxDaysPerYear : Double = 365.25
```

Average number of days in a year, measured over 4 years. Used in YearsBetween (204).

```
DayFriday = 5
```

ISO day number for Friday

```
DayMonday = 1
```

ISO day number for Monday

```
DaySaturday = 6
```

ISO day number for Saturday

```
DaysPerWeek = 7
```

Number of days in a week.

```
DaysPerYear : Array[Boolean] of Word = (365, 366)
```

Array with number of days in a year. The boolean index indicates whether it is a leap year or not.

```
DaySunday = 7
```

ISO day number for Sunday

```
DayThursday = 4
```

ISO day number for Thursday

```
DayTuesday = 2
```

ISO day number for Tuesday

```
DayWednesday = 3
```

ISO day number for Wednesday

```
MonthsPerYear = 12
```

Number of months in a year

```
OneHour = (1) / HoursPerDay
```

One hour as a fraction of a day (suitable for TDateTime)

```
OneMillisecond = (1) / MSecsPerDay
```

One millisecond as a fraction of a day (suitable for TDateTime)

```
OneMinute = (1) / MinsPerDay
```

One minute as a fraction of a day (suitable for TDateTime)

```
OneSecond = (1) / SecsPerDay
```

One second as a fraction of a day (suitable for TDateTime)

```
RecodeLeaveFieldAsIs = (Word)
```

Bitmask deciding what to do with each TDateTime field in recode routines

`WeeksPerFortnight = 2`

Number of weeks in fortnight

`YearsPerCentury = 100`

Number of years in a century

`YearsPerDecade = 10`

Number of years in a decade

`YearsPerMillennium = 1000`

Number of years in a millenium

44.4 Procedures and functions

44.4.1 CompareDate

Synopsis: Compare 2 dates, disregarding the time of day

Declaration:
```
function CompareDate(const A: TDateTime;const B: TDateTime)
                          : TValueRelationship
```

Visibility: default

Description: `CompareDate` compares the date parts of two timestamps `A` and `B` and returns the following results:

< 0 if the day part of `A` is earlier than the day part of `B`.

0 if `A` and `B` are the on same day (times may differ) .

> 0 if the day part of `A` is later than the day part of `B`.

See also: CompareTime (120), CompareDateTime (119), SameDate (173), SameTime (174), SameDateTime (173)

Listing: ./datutex/ex99.pp

```
Program Example99;

{ This program demonstrates the CompareDate function }

Uses SysUtils, DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy';

Procedure Test(D1,D2 : TDateTime);

Var
  Cmp : Integer;
```

```
begin
  Write(FormatDateTime(Fmt,D1),' is ');
  Cmp:=CompareDate(D1,D2);
  If Cmp<0 then
    write('earlier than ')
  else if Cmp>0 then
    Write('later than ')
  else
    Write('equal to ');
  WriteIn(FormatDateTime(Fmt,D2));
end;

Var
  D,N : TDateTime;

Begin
  D:=Today;
  N:=Now;
  Test(D,D);
  Test(N,N);
  Test(D+1,D);
  Test(D-1,D);
  Test(D+OneSecond,D);
  Test(D-OneSecond,D);
  Test(N+OneSecond,N);
  Test(N-OneSecond,N);
End.
```

44.4.2 CompareDateTime

Synopsis: Compare 2 dates, taking into account the time of day

Declaration:
```
function CompareDateTime(const A: TDateTime;const B: TDateTime)
                                : TValueRelationship
```

Visibility: default

Description: `CompareDateTime` compares two timestamps `A` and `B` and returns the following results:

< 0 if `A` is earlier in date/time than `B`.

0 if `A` and `B` are the same date/time .

> 0 if `A` is later in date/time than `B`.

See also: CompareTime (120), CompareDate (118), SameDate (173), SameTime (174), SameDateTime (173)

Listing: ./datutex/ex98.pp

```
Program Example98;

{ This program demonstrates the CompareDateTime function }

Uses SysUtils, DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss.zzz';
```

```
Procedure Test(D1,D2 : TDateTime);

Var
  Cmp : Integer;

begin
  Write(FormatDateTime(Fmt,D1),' is ');
  Cmp:=CompareDateTime(D1,D2);
  If Cmp<0 then
    write('earlier than ')
  else if Cmp>0 then
    Write('later than ')
  else
    Write('equal to ');
  WriteIn(FormatDateTime(Fmt,D2));
end;

Var
  D,N : TDateTime;

Begin
  D:=Today;
  N:=Now;
  Test(D,D);
  Test(N,N);
  Test(D+1,D);
  Test(D-1,D);
  Test(D+OneSecond,D);
  Test(D-OneSecond,D);
  Test(N+OneSecond,N);
  Test(N-OneSecond,N);
End.
```

44.4.3 CompareTime

Synopsis: Compares two times of the day, disregarding the date part.

Declaration:
```
function CompareTime(const A: TDateTime;const B: TDateTime)
                          : TValueRelationship
```

Visibility: default

Description: `CompareTime` compares the time parts of two timestamps `A` and `B` and returns the following results:

< 0 if the time part of `A` is earlier than the time part of `B`.

0 if `A` and `B` have the same time part (dates may differ) .

> 0 if the time part of `A` is later than the time part of `B`.

See also: CompareDateTime (119), CompareDate (118), SameDate (173), SameTime (174), SameDateTime (173)

Listing: ./datutex/ex100.pp

```
Program Example100;

{ This program demonstrates the CompareTime function }

Uses SysUtils,DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss.zzz';

Procedure Test(D1,D2 : TDateTime);

Var
  Cmp : Integer;

begin
  Write(FormatDateTime(Fmt,D1),' has ');
  Cmp:=CompareDateTime(D1,D2);
  If Cmp<0 then
    write('earlier time than ')
  else if Cmp>0 then
    Write('later time than ')
  else
    Write('equal time with ');
  WriteIn(FormatDateTime(Fmt,D2));
end;

Var
  D,N : TDateTime;

Begin
  D:=Today;
  N:=Now;
  Test(D,D);
  Test(N,N);
  Test(N+1,N);
  Test(N-1,N);
  Test(N+OneSecond,N);
  Test(N-OneSecond,N);
End.
```

44.4.4 DateOf

Synopsis: Extract the date part from a DateTime indication.

Declaration: `function DateOf(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `DateOf` extracts the date part from `AValue` and returns the result.

Since the `TDateTime` is actually a double with the date part encoded in the integer part, this operation corresponds to a call to `Trunc`.

See also: TimeOf (184), YearOf (203), MonthOf (162), DayOf (123), HourOf (138), MinuteOf (158), SecondOf (175), MilliSecondOf (154)

Listing: ./datutex/ex1.pp

```
Program Example1;

{ This program demonstrates the DateOf function }

Uses SysUtils, DateUtils;

Begin
  WriteIn('Date is: ',DateTimeToStr(DateOf(Now)));
End.
```

44.4.5 DateTimeToDosDateTime

Synopsis: Convert `TDateTime` format to DOS date/time format

Declaration: `function DateTimeToDosDateTime(const AValue: TDateTime) : LongInt`

Visibility: default

Description: `DateTimeToDosDatetime` takes `Value`, a `TDateTime` formatted timestamp, and recodes it to a MS-DOS encoded date/time value. This is a longint with the date/time encoded in the bits as:

0-4 Seconds divided by 2

5-10 Minutes

11-15 Hours

16-20 Day

21-24 Month

25-31 Years since 1980

See also: DosDateTimeToDateTime (131)

44.4.6 DateTimeToJulianDate

Synopsis: Converts a TDateTime value to a Julian date representation

Declaration: `function DateTimeToJulianDate(const AValue: TDateTime) : Double`

Visibility: default

Description: `DateTimeToJulianDate` converts the `AValue` date/time indication to a julian (as opposed to Gregorian) date.

See also: JulianDateToDateTime (153), TryJulianDateToDateTime (189), DateTimeToModifiedJulianDate (123), TryModifiedJulianDateToDateTime (189)

44.4.7 DateTimeToMac

Synopsis: Convert a `TDateTime` timestamp to a Mac timestamp

Declaration: `function DateTimeToMac(const AValue: TDateTime) : Int64`

Visibility: default

Description: `DateTimeToMac` converts the `TDateTime` value `AValue` to a valid Mac timestamp indication and returns the result.

Errors: None.

See also: UnixTimeStampToMac (191), MacToDateTime (154), MacTimeStampToUnix (154)

44.4.8 DateTimeToModifiedJulianDate

Synopsis: Convert a `TDateTime` value to a modified Julian date representation

Declaration: `function DateTimeToModifiedJulianDate(const AValue: TDateTime) : Double`

Visibility: default

Description: Not yet implemented.

Errors: Currently, trying to use this function will raise an exception.

See also: DateTimeToJulianDate (122), JulianDateToDateTime (153), TryJulianDateToDateTime (189), TryModifiedJulianDateToDateTime (189)

44.4.9 DateTimeToUnix

Synopsis: Convert a `TDateTime` value to Unix epoch time

Declaration: `function DateTimeToUnix(const AValue: TDateTime) : Int64`

Visibility: default

Description: `DateTimeToUnix` converts a `TDateTime` value to a epoch time (i.e. the number of seconds elapsed since 1/1/1970).

See also: UnixToDateTime (191)

44.4.10 DayOf

Synopsis: Extract the day (of month) part from a DateTime value

Declaration: `function DayOf(const AValue: TDateTime) : Word`

Visibility: default

Description: `DayOf` returns the day of the month part of the `AValue` date/time indication. It is a number between 1 and 31.

For an example, see YearOf (203)

See also: YearOf (203), WeekOf (191), MonthOf (162), HourOf (138), MinuteOf (158), SecondOf (175), MilliSecondOf (154)

44.4.11 DayOfTheMonth

Synopsis: Extract the day (of month) part of a DateTime value

Declaration: `function DayOfTheMonth(const AValue: TDateTime) : Word`

Visibility: default

Description: `DayOfTheMonth` returns the number of days that have passed since the start of the month till the moment indicated by `AValue`. This is a one-based number, i.e. the first day of the month will return 1.

For an example, see the WeekOfTheMonth (192) function.

See also: DayOfTheYear (124), WeekOfTheMonth (192), HourOfTheMonth (139), MinuteOfTheMonth (159), SecondOfTheMonth (177), MilliSecondOfTheMonth (155)

44.4.12 DayOfTheWeek

Synopsis: Extracts the day of the week from a DateTime value

Declaration: `function DayOfTheWeek(const AValue: TDateTime) : Word`

Visibility: default

Description: `DayOfTheWeek` returns the number of days that have passed since the start of the week till the moment indicated by `AValue`. This is a one-based number, i.e. the first day of the week will return 1.

See also: DayOfTheYear (124), DayOfTheMonth (123), HourOfTheWeek (139), MinuteOfTheWeek (160), SecondOfTheWeek (177), MilliSecondOfTheWeek (156)

Listing: ./datutex/ex42.pp

```
Program Example42;

{ This program demonstrates the WeekOfTheMonth function }

Uses SysUtils, DateUtils;

Var
  N : TDateTime;

Begin
  N:=Now;
  Writeln('Day of the Week          : ',DayOfTheWeek(N));
  Writeln('Hour of the Week         : ',HourOfTheWeek(N));
  Writeln('Minute of the Week       : ',MinuteOfTheWeek(N));
  Writeln('Second of the Week       : ',SecondOfTheWeek(N));
  Writeln('MilliSecond of the Week : ',
          MilliSecondOfTheWeek(N));
End.
```

44.4.13 DayOfTheYear

Synopsis: Extracts the day of the year from a `TDateTime` value

Declaration: `function DayOfTheYear(const AValue: TDateTime) : Word`

Visibility: default

Description: `DayOfTheYear` returns the number of days that have passed since the start of the year till the moment indicated by `AValue`. This is a one-based number, i.e. January 1 will return 1.

For an example, see the WeekOfTheYear (192) function.

See also: WeekOfTheYear (192), HourOfTheYear (140), MinuteOfTheYear (160), SecondOfTheYear (177), MilliSecondOfTheYear (156)

44.4.14 DaysBetween

Synopsis: Number of whole days between two DateTime values.

Declaration: `function DaysBetween(const ANow: TDateTime;const AThen: TDateTime) : Integer`

Visibility: default

Description: `DaysBetween` returns the number of whole days between `ANow` and `AThen`. This means the fractional part of a day (hours, minutes, etc.) is dropped.

See also: YearsBetween (204), MonthsBetween (163), WeeksBetween (193), HoursBetween (140), MinutesBetween (160), SecondsBetween (178), MilliSecondsBetween (157)

Listing: ./datutex/ex58.pp

```
Program Example58;

{ This program demonstrates the DaysBetween function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
  Write('Number of days between ');
  Write(DateTimeToStr(AThen),' and ',DateTimeToStr(ANow));
  Writeln(' : ',DaysBetween(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=Today-23/24;
  Test(D1,D2);
  D2:=Today-1;
  Test(D1,D2);
  D2:=Today-25/24;
  Test(D1,D2);
  D2:=Today-26/24;
  Test(D1,D2);
  D2:=Today-5.4;
  Test(D1,D2);
  D2:=Today-2.5;
  Test(D1,D2);
End.
```

44.4.15 DaysInAMonth

Synopsis: Number of days in a month of a certain year.

Declaration: `function DaysInAMonth(const AYear: Word;const AMonth: Word) : Word`

Visibility: default

Description: `DaysInYMonth` returns the number of days in the month `AMonth` in the year`AYear`. The return value takes leap years into account.

See also: WeeksInAYear (194), WeeksInYear (194), DaysInYear (127), DaysInAYear (126), DaysInMonth (126)

Listing: ./datutex/ex17.pp

```
Program Example17;

{ This program demonstrates the DaysInAMonth function }

Uses SysUtils,DateUtils;

Var
  Y,M : Word;

Begin
  For Y:=1992 to 2010 do
    For M:=1 to 12 do
      WriteIn(LongMonthNames[m],' ',Y,' has ',DaysInAMonth(Y,M),' days.');
End.
```

44.4.16 DaysInAYear

Synopsis: Number of days in a particular year.

Declaration: `function DaysInAYear(const AYear: Word) : Word`

Visibility: default

Description: `DaysInAYear` returns the number of weeks in the year `AYear`. The return value is either 365 or 366.

See also: WeeksInAYear (194), WeeksInYear (194), DaysInYear (127), DaysInMonth (126), DaysInAMonth (125)

Listing: ./datutex/ex15.pp

```
Program Example15;

{ This program demonstrates the DaysInAYear function }

Uses SysUtils,DateUtils;

Var
  Y : Word;

Begin
  For Y:=1992 to 2010 do
    WriteIn(Y,' has ',DaysInAYear(Y),' days.');
End.
```

44.4.17 DaysInMonth

Synopsis: Return the number of days in the month in which a date occurs.

Declaration: `function DaysInMonth(const AValue: TDateTime) : Word`

Visibility: default

Description: `DaysInMonth` returns the number of days in the month in which `AValue` falls. The return value takes leap years into account.

See also: WeeksInAYear (194), WeeksInYear (194), DaysInYear (127), DaysInAYear (126), DaysInAMonth (125)

Listing: ./datutex/ex16.pp

```
Program Example16;

{ This program demonstrates the DaysInMonth function }

Uses SysUtils,DateUtils;

Var
  Y,M : Word;

Begin
  For Y:=1992 to 2010 do
    For M:=1 to 12 do
      WriteIn(LongMonthNames[m],' ',Y,' has ',DaysInMonth(EncodeDate(Y,M,1)),' days.');
End.
```

44.4.18 DaysInYear

Synopsis: Return the number of days in the year in which a date occurs.

Declaration: `function DaysInYear(const AValue: TDateTime) : Word`

Visibility: default

Description: `daysInYear` returns the number of days in the year part of `AValue`. The return value is either 365 or 366.

See also: WeeksInAYear (194), WeeksInYear (194), DaysInAYear (126), DaysInMonth (126), DaysInAMonth (125)

Listing: ./datutex/ex14.pp

```
Program Example14;

{ This program demonstrates the DaysInYear function }

Uses SysUtils,DateUtils;

Var
  Y : Word;

Begin
  For Y:=1992 to 2010 do
    WriteIn(Y,' has ',DaysInYear(EncodeDate(Y,1,1)),' days.');
End.
```

44.4.19 DaySpan

Synopsis: Calculate the approximate number of days between two DateTime values.

Declaration: `function DaySpan(const ANow: TDateTime;const AThen: TDateTime) : Double`

Visibility: default

Description: `DaySpan` returns the number of Days between `ANow` and `AThen`, including any fractional parts of a Day.

See also: YearSpan (205), MonthSpan (164), WeekSpan (195), HourSpan (141), MinuteSpan (161), SecondSpan (179), MilliSecondSpan (157), DaysBetween (124)

Listing: ./datutex/ex66.pp

```
Program Example66;

{ This program demonstrates the DaySpan function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
  Write('Number of days between ');
  Write(DateTimeToStr(AThen),' and ',DateTimeToStr(ANow));
  Writeln(' : ',DaySpan(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=Today-23/24;
  Test(D1,D2);
  D2:=Today-1;
  Test(D1,D2);
  D2:=Today-25/24;
  Test(D1,D2);
  D2:=Today-26/24;
  Test(D1,D2);
  D2:=Today-5.4;
  Test(D1,D2);
  D2:=Today-2.5;
  Test(D1,D2);
End.
```

44.4.20 DecodeDateDay

Synopsis: Decode a DateTime value in year and year of day.

Declaration:
```
procedure DecodeDateDay(const AValue: TDateTime;out AYear: Word;
                        out ADayOfYear: Word)
```

Visibility: default

Description: `DecodeDateDay` decomposes the date indication in `AValue` and returns the various components in `AYear`, `ADayOfYear`.

See also: EncodeDateTime (132), EncodeDateMonthWeek (132), EncodeDateWeek (133), EncodeDateDay (132), DecodeDateTime (129), DecodeDateWeek (130), DecodeDateMonthWeek (129)

Listing: ./datutex/ex83.pp

```
Program Example83;

{ This program demonstrates the DecodeDateDay function }

Uses SysUtils, DateUtils;

Var
  Y,DoY : Word;
  TS : TDateTime;

Begin
  DecodeDateDay(Now,Y,DoY);
  TS:=EncodeDateDay(Y,DoY);
  Writeln('Today is : ',DateToStr(TS));
End.
```

44.4.21 DecodeDateMonthWeek

Synopsis: Decode a DateTime value in a month, week of month and day of week

Declaration:
```
procedure DecodeDateMonthWeek(const AValue: TDateTime;out AYear: Word;
                              out AMonth: Word;out AWeekOfMonth: Word;
                              out ADayOfWeek: Word)
```

Visibility: default

Description: `DecodeDateMonthWeek` decomposes the date indication in `AValue` and returns the various components in `AYear`, `AMonthAWeekOfMonth` and `ADayOfWeek`.

See also: EncodeDateTime (132), EncodeDateMonthWeek (132), EncodeDateWeek (133), EncodeDateDay (132), DecodeDateTime (129), DecodeDateWeek (130), DecodeDateDay (128)

Listing: ./datutex/ex85.pp

```
Program Example85;

{ This program demonstrates the DecodeDateMonthWeek function }

Uses SysUtils, DateUtils;

Var
  Y,M,Wom,Dow : Word;
  TS : TDateTime;

Begin
  DecodeDateMonthWeek(Now,Y,M,WoM,DoW);
  TS:=EncodeDateMonthWeek(Y,M,WoM,Dow);
  Writeln('Today is : ',DateToStr(TS));
End.
```

44.4.22 DecodeDateTime

Synopsis: Decode a datetime value in a date and time value

Declaration: `procedure DecodeDateTime(const AValue: TDateTime;out AYear: Word;
out AMonth: Word;out ADay: Word;out AHour: Word;
out AMinute: Word;out ASecond: Word;
out AMilliSecond: Word)`

Visibility: default

Description: `DecodeDateTime` decomposes the date/time indication in `AValue` and returns the various components in `AYear`, `AMonth`, `ADay`, `AHour`, `AMinute`, `ASecond`, `AMilliSecond`

See also: EncodeDateTime (132), EncodeDateMonthWeek (132), EncodeDateWeek (133), EncodeDateDay (132), DecodeDateWeek (130), DecodeDateDay (128), DecodeDateMonthWeek (129)

Listing: ./datutex/ex79.pp

```
Program Example79;

{ This program demonstrates the DecodeDateTime function }

Uses SysUtils, DateUtils;

Var
  Y,Mo,D,H,Mi,S,MS : Word;
  TS : TDateTime;

Begin
  DecodeDateTime(Now,Y,Mo,D,H,Mi,S,MS);
  TS:=EncodeDateTime(Y,Mo,D,H,Mi,S,MS);
  Writeln('Now is : ',DateTimeToStr(TS));
End.
```

44.4.23 DecodeDateWeek

Synopsis: Decode a DateTime value in a week of year and day of week.

Declaration: `procedure DecodeDateWeek(const AValue: TDateTime;out AYear: Word;
out AWeekOfYear: Word;out ADayOfWeek: Word)`

Visibility: default

Description: `DecodeDateWeek` decomposes the date indication in `AValue` and returns the various components in `AYear`, `AWeekOfYear`, `ADayOfWeek`.

See also: EncodeDateTime (132), EncodeDateMonthWeek (132), EncodeDateWeek (133), EncodeDateDay (132), DecodeDateTime (129), DecodeDateDay (128), DecodeDateMonthWeek (129)

Listing: ./datutex/ex81.pp

```
Program Example81;

{ This program demonstrates the DecodeDateWeek function }

Uses SysUtils, DateUtils;

Var
  Y,W,Dow : Word;
  TS : TDateTime;
```

```
Begin
  DecodeDateWeek(Now,Y,W,Dow);
  TS:=EncodeDateWeek(Y,W,Dow);
  Writeln('Today is : ',DateToStr(TS));
End.
```

44.4.24 DecodeDayOfWeekInMonth

Synopsis: Decode a DateTime value in year, month, day of week parts

Declaration:
```
procedure DecodeDayOfWeekInMonth(const AValue: TDateTime;
                                 out AYear: Word;out AMonth: Word;
                                 out ANthDayOfWeek: Word;
                                 out ADayOfWeek: Word)
```

Visibility: default

Description: `DecodeDayOfWeekInMonth` decodes the date `AValue` in a `AYear`, `AMonth`, `ADayOfweek` and `ANthDayOfweek`. (This is the N-th time that this weekday occurs in the month, e.g. the third saturday of the month.)

See also: NthDayOfWeek (164), EncodeDateMonthWeek (132), #rtl.sysutils.DayOfWeek (628), EncodeDayOfWeekInMonth (133), TryEncodeDayOfWeekInMonth (188)

Listing: ./datutex/ex105.pp

```
Program Example105;

{ This program demonstrates the DecodeDayOfWeekInMonth function }

Uses SysUtils,DateUtils;

Var
  Y,M,NDoW,DoW : Word;
  D : TDateTime;
Begin
  DecodeDayOfWeekInMonth(Date,Y,M,NDoW,DoW);
  D:=EncodeDayOfWeekInMonth(Y,M,NDoW,DoW);
  Write(DateToStr(D),' is the ',NDow,'-th ');
  Writeln(formatdateTime('dddd',D),' of the month.');
End.
```

44.4.25 DosDateTimeToDateTime

Synopsis: Convert DOS date/time format to `TDateTime` format

Declaration:
```
function DosDateTimeToDateTime(AValue: LongInt) : TDateTime
```

Visibility: default

Description: `DosDateTimeToDateTime` takes a DOS encoded date/time `AValue` and recodes it as a `TDateTime` value.

The bit encoding of the DOS date/time is explained in the DateTimeToDosDateTime (122) function.

See also: DateTimeToDosDateTime (122)

44.4.26 EncodeDateDay

Synopsis: Encodes a year and day of year to a DateTime value

Declaration:
```
function EncodeDateDay(const AYear: Word;const ADayOfYear: Word)
                       : TDateTime
```

Visibility: default

Description: `EncodeDateDay` encodes the values `AYear` and `ADayOfYear` to a date value and returns this value.

For an example, see DecodeDateDay (128).

Errors: If any of the arguments is not valid, then an `EConvertError` exception is raised.

See also: EncodeDateMonthWeek (132), DecodeDateDay (128), EncodeDateTime (132), EncodeDateWeek (133), TryEncodeDateTime (187), TryEncodeDateMonthWeek (186), TryEncodeDateWeek (187)

44.4.27 EncodeDateMonthWeek

Synopsis: Encodes a year, month, week of month and day of week to a DateTime value

Declaration:
```
function EncodeDateMonthWeek(const AYear: Word;const AMonth: Word;
                             const AWeekOfMonth: Word;
                             const ADayOfWeek: Word) : TDateTime
```

Visibility: default

Description: `EncodeDateTime` encodes the values `AYearAMonth`, `WeekOfMonth`,`ADayOfWeek`, to a date value and returns this value.

For an example, see DecodeDateMonthWeek (129).

Errors: If any of the arguments is not valid, then an `EConvertError` exception is raised.

See also: DecodeDateMonthWeek (129), EncodeDateTime (132), EncodeDateWeek (133), EncodeDateDay (132), TryEncodeDateTime (187), TryEncodeDateWeek (187), TryEncodeDateMonthWeek (186), TryEncodeDateDay (185), NthDayOfWeek (164)

44.4.28 EncodeDateTime

Synopsis: Encodes a DateTime value from all its parts

Declaration:
```
function EncodeDateTime(const AYear: Word;const AMonth: Word;
                        const ADay: Word;const AHour: Word;
                        const AMinute: Word;const ASecond: Word;
                        const AMilliSecond: Word) : TDateTime
```

Visibility: default

Description: `EncodeDateTime` encodes the values `AYearAMonth`, `ADay`,`AHour`, `AMinute`,`ASecond` and `AMilliSecond` to a date/time valueand returns this value.

For an example, see DecodeDateTime (129).

Errors: If any of the arguments is not valid, then an `EConvertError` exception is raised.

See also: DecodeDateTime (129), EncodeDateMonthWeek (132), EncodeDateWeek (133), EncodeDateDay (132), TryEncodeDateTime (187), TryEncodeDateWeek (187), TryEncodeDateDay (185), TryEncodeDateMonthWeek (186)

44.4.29 EncodeDateWeek

Synopsis: Encode a `TDateTime` value from a year, week and day of week triplet

Declaration:
```
function EncodeDateWeek(const AYear: Word;const AWeekOfYear: Word;
                        const ADayOfWeek: Word) : TDateTime
function EncodeDateWeek(const AYear: Word;const AWeekOfYear: Word)
                        : TDateTime
```

Visibility: default

Description: `EncodeDateWeek` encodes the values `AYear`, `AWeekOfYear` and `ADayOfWeek` to a date value and returns this value.

For an example, see DecodeDateWeek (130).

Errors: If any of the arguments is not valid, then an `EConvertError` exception is raised.

See also: EncodeDateMonthWeek (132), DecodeDateWeek (130), EncodeDateTime (132), EncodeDateDay (132), TryEncodeDateTime (187), TryEncodeDateWeek (187), TryEncodeDateMonthWeek (186)

44.4.30 EncodeDayOfWeekInMonth

Synopsis: Encodes a year, month, week, day of week specification to a `TDateTime` value

Declaration:
```
function EncodeDayOfWeekInMonth(const AYear: Word;const AMonth: Word;
                                const ANthDayOfWeek: Word;
                                const ADayOfWeek: Word) : TDateTime
```

Visibility: default

Description: `EncodeDayOfWeekInMonth` encodes `AYear`, `AMonth`, `ADayOfweek` and `ANthDayOfweek` to a valid date stamp and returns the result.

`ANthDayOfweek`is the N-th time that this weekday occurs in the month, e.g. the third saturday of the month.

For an example, see DecodeDayOfWeekInMonth (131).

Errors: If any of the values is not in range, then an `EConvertError` exception will be raised.

See also: NthDayOfWeek (164), EncodeDateMonthWeek (132), #rtl.sysutils.DayOfWeek (628), DecodeDayOfWeekInMonth (131), TryEncodeDayOfWeekInMonth (188)

44.4.31 EncodeTimeInterval

Synopsis: Encode an interval as a `TDateTime` value.

Declaration:
```
function EncodeTimeInterval(Hour: Word;Minute: Word;Second: Word;
                            MilliSecond: Word) : TDateTime
```

Visibility: default

Description: `EncodeTimeInterval` encodes a time interval expressed in `Hour`, `Min`, `Sec`, `MSec` as a `TDateTime` value and returns the value in `Time`.

Errors: If `Min`, `Sec`, `MSec` do not contain a valid time indication, then an `EConvertError` exception is raised.

See also: TryEncodeTimeInterval (189)

44.4.32 EndOfADay

Synopsis: Calculates a DateTime value representing the end of a specified day

Declaration:
```
function EndOfADay(const AYear: Word;const AMonth: Word;
                   const ADay: Word) : TDateTime;  Overload
function EndOfADay(const AYear: Word;const ADayOfYear: Word) : TDateTime
                   ;  Overload
```

Visibility: default

Description: `EndOfADay` returns a `TDateTime` value with the date/time indication of the last moment (23:59:59.999) of the day given by `AYear`, `AMonth`, `ADay`.

The day may also be indicated with a `AYear`, `ADayOfYear` pair.

See also: StartOfTheDay (182), StartOfADay (179), StartOfTheWeek (183), StartOfAWeek (181), StartOfAMonth (180), StartOfTheMonth (182), EndOfTheWeek (137), EndOfAWeek (135), EndOfTheYear (138), EndOfAYear (136), EndOfTheMonth (137), EndOfAMonth (134), EndOfTheDay (136)

Listing: ./datutex/ex39.pp

```
Program Example39;

{ This program demonstrates the EndOfADay function }

Uses SysUtils, DateUtils;

Const
  Fmt = '"End of the day : "dd mmmm yyyy hh:nn:ss';

Var
  Y,M,D : Word;

Begin
  Y:=YearOf(Today);
  M:=MonthOf(Today);
  D:=DayOf(Today);
  WriteLn(FormatDateTime(Fmt,EndOfADay(Y,M,D)));
  DecodeDateDay(Today,Y,D);
  WriteLn(FormatDateTime(Fmt,EndOfADay(Y,D)));
End.
```

44.4.33 EndOfAMonth

Synopsis: Calculate a datetime value representing the last day of the indicated month

Declaration:
```
function EndOfAMonth(const AYear: Word;const AMonth: Word) : TDateTime
```

Visibility: default

Description: `EndOfAMonth` e returns a `TDateTime` value with the date of the last day of the month indicated by the `AYear`, `AMonth` pair.

See also: StartOfTheMonth (182), StartOfAMonth (180), EndOfTheMonth (137), EndOfTheYear (138), EndOfAYear (136), StartOfAWeek (181), StartOfTheWeek (183)

Listing: ./datutex/ex31.pp

```
Program Example31;

{ This program demonstrates the EndOfAMonth function }

Uses SysUtils,DateUtils;

Const
  Fmt = '"Last day of this month : "dd mmmm yyyy';
Var
  Y,M : Word;

Begin
  Y:=YearOf(Today);
  M:=MonthOf(Today);
  WriteLn(FormatDateTime(Fmt,EndOfAMonth(Y,M)));
End.
```

44.4.34 EndOfAWeek

Synopsis: Return the last moment of day of the week, given a year and a week in the year.

Declaration:
```
function EndOfAWeek(const AYear: Word;const AWeekOfYear: Word;
                    const ADayOfWeek: Word) : TDateTime
function EndOfAWeek(const AYear: Word;const AWeekOfYear: Word)
                     : TDateTime
```

Visibility: default

Description: `EndOfAWeek` returns a `TDateTime` value with the date of the last moment (23:59:59:999) on the indicated day of the week indicated by the `AYear`, `AWeek`, `ADayOfWeek` values.

The default value for `ADayOfWeek` is 7.

See also: StartOfTheWeek (183), EndOfTheWeek (137), EndOfAWeek (135), StartOfAMonth (180), EndOfTheYear (138), EndOfAYear (136), EndOfTheMonth (137), EndOfAMonth (134)

Listing: ./datutex/ex35.pp

```
Program Example35;

{ This program demonstrates the EndOfAWeek function }

Uses SysUtils,DateUtils;

Const
  Fmt = '"Last day of this week : "dd mmmm yyyy hh:nn:ss';
  Fmt2 = '"Last-1 day of this week : "dd mmmm yyyy hh:nn:ss';

Var
  Y,W : Word;

Begin
  Y:=YearOf(Today);
  W:=WeekOf(Today);
  WriteLn(FormatDateTime(Fmt,EndOfAWeek(Y,W)));
  WriteLn(FormatDateTime(Fmt2,EndOfAWeek(Y,W,6)));
End.
```

44.4.35 EndOfAYear

Synopsis: Calculate a DateTime value representing the last day of a year

Declaration: `function EndOfAYear(const AYear: Word) : TDateTime`

Visibility: default

Description: `StartOfAYear` returns a `TDateTime` value with the date of the last day of the year `AYear` (December 31).

See also: StartOfTheYear (184), EndOfTheYear (138), EndOfAYear (136), EndOfTheMonth (137), EndOfAMonth (134), StartOfAWeek (181), StartOfTheWeek (183)

Listing: ./datutex/ex27.pp

```
Program Example27;

{ This program demonstrates the EndOfAYear function }

Uses SysUtils, DateUtils;

Const
  Fmt = '"Last day of this year : "dd mmmm yyyy';

Begin
  WriteLn(FormatDateTime(Fmt, EndOfAYear(YearOf(Today))));
End.
```

44.4.36 EndOfTheDay

Synopsis: Calculate a datetime value that represents the end of a given day.

Declaration: `function EndOfTheDay(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `EndOfTheDay` extracts the date part of `AValue` and returns a `TDateTime` value with the date/-time indication of the last moment (23:59:59.999) of this day.

See also: StartOftheDay (182), StartOfADay (179), StartOfTheWeek (183), StartOfAWeek (181), StartOfAMonth (180), StartOfTheMonth (182), EndOfTheWeek (137), EndOfAWeek (135), EndOfTheYear (138), EndOfAYear (136), EndOfTheMonth (137), EndOfAMonth (134), EndOfADay (134)

Listing: ./datutex/ex37.pp

```
Program Example37;

{ This program demonstrates the EndOfTheDay function }

Uses SysUtils, DateUtils;

Const
  Fmt = '"End of the day : "dd mmmm yyyy hh:nn:ss';


Begin
  WriteLn(FormatDateTime(Fmt, EndOfTheDay(Today)));
End.
```

44.4.37 EndOfTheMonth

Synopsis: Calculate a DateTime value representing the last day of the month, given a day in that month.

Declaration: `function EndOfTheMonth(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `EndOfTheMonth` extracts the year and month parts of `AValue` and returns a `TDateTime` value with the date of the first day of that year and month as the EndOfAMonth (134) function.

See also: StartOfAMonth (180), StartOfTheMonth (182), EndOfAMonth (134), EndOfTheYear (138), EndOfAYear (136), StartOfAWeek (181), StartOfTheWeek (183)

Listing: ./datutex/ex29.pp

```
Program Example29;

{ This program demonstrates the EndOfTheMonth function }

Uses SysUtils, DateUtils;

Const
  Fmt = '"last day of this month : "dd mmmm yyyy';

Begin
  WriteLn(FormatDateTime(Fmt, EndOfTheMonth(Today)));
End.
```

44.4.38 EndOfTheWeek

Synopsis: Calculate a DateTime value which represents the end of a week, given a date in that week.

Declaration: `function EndOfTheWeek(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `EndOfTheWeek` extracts the year and week parts of `AValue` and returns a `TDateTime` value with the date of the last day of that week as the EndOfAWeek (135) function.

See also: StartOfAWeek (181), StartOfTheWeek (183), EndOfAWeek (135), StartOfAMonth (180), EndOfTheYear (138), EndOfAYear (136), EndOfTheMonth (137), EndOfAMonth (134)

Listing: ./datutex/ex33.pp

```
Program Example33;

{ This program demonstrates the EndOfTheWeek function }

Uses SysUtils, DateUtils;

Const
  Fmt = '"last day of this week : "dd mmmm yyyy';

Begin
  WriteLn(FormatDateTime(Fmt, EndOfTheWeek(Today)));
End.
```

44.4.39 EndOfTheYear

Synopsis: Calculate a DateTime value representing the last day of a year, given a date in that year.

Declaration: `function EndOfTheYear(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `EndOfTheYear` extracts the year part of `AValue` and returns a `TDateTime` value with the date of the last day of that year (December 31), as the EndOfAYear (136) function.

See also: StartOfAYear (181), StartOfTheYear (184), EndOfTheMonth (137), EndOfAMonth (134), StartOfAWeek (181), StartOfTheWeek (183), EndOfAYear (136)

Listing: ./datutex/ex25.pp

```
Program Example25;

{ This program demonstrates the EndOfTheYear function }

Uses SysUtils, DateUtils;

Const
  Fmt = '"Last day of this year : "dd mmmm yyyy';

Begin
  WriteLn(FormatDateTime(Fmt, EndOfTheYear(Today)));
End.
```

44.4.40 HourOf

Synopsis: Extract the hour part from a DateTime value.

Declaration: `function HourOf(const AValue: TDateTime) : Word`

Visibility: default

Description: `HourOf` returns the hour of the day part of the `AValue` date/time indication. It is a number between 0 and 23.

For an example, see YearOf (203)

See also: YearOf (203), WeekOf (191), MonthOf (162), DayOf (123), MinuteOf (158), SecondOf (175), MilliSecondOf (154)

44.4.41 HourOfTheDay

Synopsis: Calculate the hour of a given DateTime value

Declaration: `function HourOfTheDay(const AValue: TDateTime) : Word`

Visibility: default

Description: `HourOfTheDay` returns the number of hours that have passed since the start of the day till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:59:59 will return 0.

See also: HourOfTheYear (140), HourOfTheMonth (139), HourOfTheWeek (139), MinuteOfTheDay (158), SecondOfTheDay (176), MilliSecondOfTheDay (154)

Listing: ./datutex/ex43.pp

```
Program Example43;

{ This program demonstrates the HourOfTheDay function }

Uses SysUtils, DateUtils;

Var
  N : TDateTime;

Begin
  N:=Now;
  Writeln('Hour of the Day        : ',HourOfTheDay(N));
  Writeln('Minute of the Day      : ',MinuteOfTheDay(N));
  Writeln('Second of the Day      : ',SecondOfTheDay(N));
  Writeln('MilliSecond of the Day : ',
          MilliSecondOfTheDay(N));
End.
```

44.4.42 HourOfTheMonth

Synopsis: Calculate the number of hours passed since the start of the month.

Declaration: `function HourOfTheMonth(const AValue: TDateTime) : Word`

Visibility: default

Description: `HourOfTheMonth` returns the number of hours that have passed since the start of the month till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:59:59 on the first day of the month will return 0.

For an example, see the WeekOfTheMonth (192) function.

See also: WeekOfTheMonth (192), DayOfTheMonth (123), MinuteOfTheMonth (159), SecondOfTheMonth (177), MilliSecondOfTheMonth (155)

44.4.43 HourOfTheWeek

Synopsis: Calculate the number of hours elapsed since the start of the week.

Declaration: `function HourOfTheWeek(const AValue: TDateTime) : Word`

Visibility: default

Description: `HourOfTheWeek` returns the number of hours that have passed since the start of the Week till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:59:59 on the first day of the week will return 0.

For an example, see the DayOfTheWeek (124) function.

See also: HourOfTheYear (140), HourOfTheMonth (139), HourOfTheDay (138), DayOfTheWeek (124), MinuteOfTheWeek (160), SecondOfTheWeek (177), MilliSecondOfTheWeek (156)

44.4.44 HourOfTheYear

Synopsis: Calculate the number of hours passed since the start of the year.

Declaration: `function HourOfTheYear(const AValue: TDateTime) : Word`

Visibility: default

Description: `HourOfTheYear` returns the number of hours that have passed since the start of the year (January 1, 00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. January 1 00:59:59 will return 0.

For an example, see the WeekOfTheYear (192) function.

See also: WeekOfTheYear (192), DayOfTheYear (124), MinuteOfTheYear (160), SecondOfTheYear (177), MilliSecondOfTheYear (156)

44.4.45 HoursBetween

Synopsis: Calculate the number of whole hours between two DateTime values.

Declaration: `function HoursBetween(const ANow: TDateTime;const AThen: TDateTime) : Int64`

Visibility: default

Description: `HoursBetween` returns the number of whole hours between `ANow` and `AThen`. This means the fractional part of an hour (minutes,seconds etc.) is dropped.

See also: YearsBetween (204), MonthsBetween (163), WeeksBetween (193), DaysBetween (124), MinutesBetween (160), SecondsBetween (178), MilliSecondsBetween (157)

Listing: ./datutex/ex59.pp

```
Program Example59;

{ This program demonstrates the HoursBetween function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
  Write('Number of hours between ');
  Write(DateTimeToStr(AThen),' and ',DateTimeToStr(ANow));
  Writeln(' : ',HoursBetween(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=D1-(59*OneMinute);
  Test(D1,D2);
  D2:=D1-(61*OneMinute);
  Test(D1,D2);
  D2:=D1-(122*OneMinute);
  Test(D1,D2);
  D2:=D1-(306*OneMinute);
```

```
  Test(D1,D2);
  D2:=D1-(5.4*OneHour);
  Test(D1,D2);
  D2:=D1-(2.5*OneHour);
  Test(D1,D2);
End.
```

44.4.46 HourSpan

Synopsis: Calculate the approximate number of hours between two DateTime values.

Declaration: `function HourSpan(const ANow: TDateTime;const AThen: TDateTime) : Double`

Visibility: default

Description: `HourSpan` returns the number of Hours between `ANow` and `AThen`, including any fractional parts of a Hour.

See also: YearSpan (205), MonthSpan (164), WeekSpan (195), DaySpan (127), MinuteSpan (161), SecondSpan (179), MilliSecondSpan (157), HoursBetween (140)

Listing: ./datutex/ex67.pp

```
Program Example67;

{ This program demonstrates the HourSpan function }

Uses SysUtils,DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
 Write('Number of hours between ');
 Write(DateTimeToStr(AThen),' and ',DateTimeToStr(ANow));
 Writeln(' : ',HourSpan(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=D1-(59*OneMinute);
  Test(D1,D2);
  D2:=D1-(61*OneMinute);
  Test(D1,D2);
  D2:=D1-(122*OneMinute);
  Test(D1,D2);
  D2:=D1-(306*OneMinute);
  Test(D1,D2);
  D2:=D1-(5.4*OneHour);
  Test(D1,D2);
  D2:=D1-(2.5*OneHour);
  Test(D1,D2);
End.
```

44.4.47 IncDay

Synopsis: Increase a DateTime value with a number of days.

Declaration:
```
function IncDay(const AValue: TDateTime;const ANumberOfDays: Integer)
                  : TDateTime
function IncDay(const AValue: TDateTime) : TDateTime
```

Visibility: default

Description: `IncDay` adds `ANumberOfDays` days to `AValue` and returns the resulting date/time.`ANumberOfDays` can be positive or negative.

See also: IncYear (145), #rtl.sysutils.IncMonth (628), IncWeek (144), IncHour (142), IncMinute (143), IncSecond (144), IncMilliSecond (143)

Listing: ./datutex/ex74.pp

```
Program Example74;

{ This program demonstrates the IncDay function }

Uses SysUtils, DateUtils;

Begin
  WriteLn('One Day from today is ',DateToStr(IncDay(Today,1)));
  WriteLn('One Day ago from today is ',DateToStr(IncDay(Today,-1)));
End.
```

44.4.48 IncHour

Synopsis: Increase a DateTime value with a number of hours.

Declaration:
```
function IncHour(const AValue: TDateTime;const ANumberOfHours: Int64)
                  : TDateTime
function IncHour(const AValue: TDateTime) : TDateTime
```

Visibility: default

Description: `IncHour` adds `ANumberOfHours` hours to `AValue` and returns the resulting date/time.`ANumberOfHours` can be positive or negative.

See also: IncYear (145), #rtl.sysutils.IncMonth (628), IncWeek (144), IncDay (142), IncMinute (143), IncSecond (144), IncMilliSecond (143)

Listing: ./datutex/ex75.pp

```
Program Example75
;

{ This program demonstrates the IncHour function }

Uses SysUtils, DateUtils;

Begin
  WriteLn('One Hour from now is ',DateTimeToStr(IncHour(Now,1)));
  WriteLn('One Hour ago from now is ',DateTimeToStr(IncHour(Now,-1)));
End.
```

44.4.49 IncMilliSecond

Synopsis: Increase a DateTime value with a number of milliseconds.

Declaration:
```
function IncMilliSecond(const AValue: TDateTime;
                        const ANumberOfMilliSeconds: Int64) : TDateTime
function IncMilliSecond(const AValue: TDateTime) : TDateTime
```

Visibility: default

Description: `IncMilliSecond` adds `ANumberOfMilliSeconds` milliseconds to `AValue` and returns the resulting date/time.`ANumberOfMilliSeconds` can be positive or negative.

See also: IncYear (145), #rtl.sysutils.IncMonth (628), IncWeek (144), IncDay (142), IncHour (142), IncSecond (144), IncMilliSecond (143)

Listing: ./datutex/ex78.pp

```
Program Example78;

{ This program demonstrates the IncMilliSecond function }

Uses SysUtils, DateUtils;

Begin
  WriteLn('One MilliSecond from now is ',TimeToStr(IncMilliSecond(Now,1)));
  WriteLn('One MilliSecond ago from now is ',TimeToStr(IncMilliSecond(Now,-1)));
End.
```

44.4.50 IncMinute

Synopsis: Increase a DateTime value with a number of minutes.

Declaration:
```
function IncMinute(const AValue: TDateTime;
                   const ANumberOfMinutes: Int64) : TDateTime
function IncMinute(const AValue: TDateTime) : TDateTime
```

Visibility: default

Description: `IncMinute` adds `ANumberOfMinutes` minutes to `AValue` and returns the resulting date/time.`ANumberOfMinutes` can be positive or negative.

See also: IncYear (145), #rtl.sysutils.IncMonth (628), IncWeek (144), IncDay (142), IncHour (142), IncSecond (144), IncMilliSecond (143)

Listing: ./datutex/ex76.pp

```
Program Example76;

{ This program demonstrates the IncMinute function }

Uses SysUtils, DateUtils;

Begin
  WriteLn('One Minute from now is ',TimeToStr(IncMinute(Time,1)));
  WriteLn('One Minute ago from now is ',TimeToStr(IncMinute(Time,-1)));
End.
```

44.4.51 IncSecond

Synopsis: Increase a DateTime value with a number of seconds.

Declaration:
```
function IncSecond(const AValue: TDateTime;
                   const ANumberOfSeconds: Int64) : TDateTime
function IncSecond(const AValue: TDateTime) : TDateTime
```

Visibility: default

Description: `IncSecond` adds `ANumberOfSeconds` seconds to `AValue` and returns the resulting date/-time.`ANumberOfSeconds` can be positive or negative.

See also: IncYear (145), #rtl.sysutils.IncMonth (628), IncWeek (144), IncDay (142), IncHour (142), IncSecond (144), IncMilliSecond (143)

Listing: ./datutex/ex77.pp

```
Program Example77;

{ This program demonstrates the IncSecond function }

Uses SysUtils, DateUtils;

Begin
  WriteLn('One Second from now is ',TimeToStr(IncSecond(Time,1)));
  WriteLn('One Second ago from now is ',TimeToStr(IncSecond(Time,-1)));
End.
```

44.4.52 IncWeek

Synopsis: Increase a DateTime value with a number of weeks.

Declaration:
```
function IncWeek(const AValue: TDateTime;const ANumberOfWeeks: Integer)
                 : TDateTime
function IncWeek(const AValue: TDateTime) : TDateTime
```

Visibility: default

Description: `IncWeek` adds `ANumberOfWeeks` weeks to `AValue` and returns the resulting date/time.`ANumberOfWeeks` can be positive or negative.

See also: IncYear (145), #rtl.sysutils.IncMonth (628), IncDay (142), IncHour (142), IncMinute (143), IncSecond (144), IncMilliSecond (143)

Listing: ./datutex/ex73.pp

```
Program Example73;

{ This program demonstrates the IncWeek function }

Uses SysUtils, DateUtils;

Begin
  WriteLn('One Week from today is ',DateToStr(IncWeek(Today,1)));
  WriteLn('One Week ago from today is ',DateToStr(IncWeek(Today,-1)));
End.
```

44.4.53 IncYear

Synopsis: Increase a DateTime value with a number of years.

Declaration:
```
function IncYear(const AValue: TDateTime;const ANumberOfYears: Integer)
                 : TDateTime
function IncYear(const AValue: TDateTime) : TDateTime
```

Visibility: default

Description: `IncYear` adds `ANumberOfYears` years to `AValue` and returns the resulting date/time. `ANumberOfYears` can be positive or negative.

See also: #rtl.sysutils.IncMonth (628), IncWeek (144), IncDay (142), IncHour (142), IncMinute (143), IncSecond (144), IncMilliSecond (143)

Listing: ./datutex/ex71.pp

```
Program Example71;

{ This program demonstrates the IncYear function }

Uses SysUtils, DateUtils;

Begin
  WriteLn('One year from today is ',DateToStr(IncYear(Today,1)));
  WriteLn('One year ago from today is ',DateToStr(IncYear(Today,-1)));
End.
```

44.4.54 InvalidDateDayError

Synopsis: Raise an `EConvertError` exception when a day is not a valid day of a year.

Declaration:
```
procedure InvalidDateDayError(const AYear: Word;const ADayOfYear: Word)
```

Visibility: default

Description: `InvalidDateDayError` raises an EConvertError (628) exception and formats the error message with an appropriate description made up from the parts `AYear` and `ADayOfYear`.

Normally this function should not be needed, the conversion routines call it when they have received invalid arguments.

See also: InvalidDateWeekError (146), InvalidDateTimeError (146), InvalidDateMonthWeekError (145), InvalidDayOfWeekInMonthError (147)

44.4.55 InvalidDateMonthWeekError

Synopsis: Raise an `EConvertError` exception when a Year,Month,WeekOfMonth,DayofWeek is invalid.

Declaration:
```
procedure InvalidDateMonthWeekError(const AYear: Word;
                                    const AMonth: Word;
                                    const AWeekOfMonth: Word;
                                    const ADayOfWeek: Word)
```

Visibility: default

Description: `InvalidDateMonthWeekError` raises an EConvertError (628) exception and formats the error message with an appropriate description made up from the parts `AYear`, `Amonth`, `AWeekOfMonth` and `ADayOfWeek`.

Normally this function should not be needed, the conversion routines call it when they have received invalid arguments.

See also: InvalidDateWeekError (146), InvalidDateTimeError (146), InvalidDateDayError (145), InvalidDayOfWeekInMonthError (147)

44.4.56 InvalidDateTimeError

Synopsis: Raise an `EConvertError` about an invalid date-time specification.

Declaration:
```
procedure InvalidDateTimeError(const AYear: Word;const AMonth: Word;
                               const ADay: Word;const AHour: Word;
                               const AMinute: Word;const ASecond: Word;
                               const AMilliSecond: Word;
                               const ABaseDate: TDateTime)
procedure InvalidDateTimeError(const AYear: Word;const AMonth: Word;
                               const ADay: Word;const AHour: Word;
                               const AMinute: Word;const ASecond: Word;
                               const AMilliSecond: Word)
```

Visibility: default

Description: `InvalidDateTimeError` raises an EConvertError (628) exception and formats the error message with an appropriate description made up from the parts `AYear`, `AMonth`, `ADay`,`AHour`, `AMinute`, `ASecond` and `AMilliSecond`.

Normally this function should not be needed, the conversion routines call it when they have received invalid arguments.

See also: InvalidDateWeekError (146), InvalidDateDayError (145), InvalidDateMonthWeekError (145), InvalidDayOfWeekInMonthError (147)

44.4.57 InvalidDateWeekError

Synopsis: Raise an EConvertError with an invalid Year, WeekOfyear and DayOfWeek specification

Declaration:
```
procedure InvalidDateWeekError(const AYear: Word;
                               const AWeekOfYear: Word;
                               const ADayOfWeek: Word)
```

Visibility: default

Description: `InvalidDateWeekError` raises an EConvertError (628) exception and formats the error message with an appropriate description made up from the parts `AYear`, `AWeek`, `ADayOfWeek`

Normally this function should not be needed, the conversion routines call it when they have received invalid arguments.

See also: InvalidDateTimeError (146), InvalidDateDayError (145), InvalidDateMonthWeekError (145), InvalidDayOfWeekInMonthError (147)

44.4.58 InvalidDayOfWeekInMonthError

Synopsis: Raise an `EConvertError` exception when a Year,Month,NthDayofWeek,DayofWeek is invalid.

Declaration:
```
procedure InvalidDayOfWeekInMonthError(const AYear: Word;
                                       const AMonth: Word;
                                       const ANthDayOfWeek: Word;
                                       const ADayOfWeek: Word)
```

Visibility: default

Description: `InvalidDayOfWeekInMonthError` raises an EConvertError (628) exception and formats the error message with an appropriate description made up from the parts `AYear`, `Amonth`, `ANthDayOfWeek` and `ADayOfWeek`.

Normally this function should not be needed, the conversion routines call it when they have received invalid arguments.

See also: InvalidDateWeekError (146), InvalidDateTimeError (146), InvalidDateDayError (145), InvalidDateMonthWeekError (145)

44.4.59 IsInLeapYear

Synopsis: Determine whether a date is in a leap year.

Declaration:
```
function IsInLeapYear(const AValue: TDateTime) : Boolean
```

Visibility: default

Description: `IsInLeapYear` returns `True` if the year part of `AValue` is leap year, or `False` if not.

See also: YearOf (203), IsPM (147), IsToday (149), IsSameDay (148)

Listing: ./datutex/ex3.pp

```
Program Example3;

{ This program demonstrates the IsInLeapYear function }

Uses SysUtils, DateUtils;

Begin
  WriteIn('Current year is leap year: ',IsInLeapYear(Date));
End.
```

44.4.60 IsPM

Synopsis: Determine whether a time is PM or AM.

Declaration:
```
function IsPM(const AValue: TDateTime) : Boolean
```

Visibility: default

Description: `IsPM` returns `True` if the time part of `AValue` is later then 12:00 (PM, or afternoon).

See also: YearOf (203), IsInLeapYear (147), IsToday (149), IsSameDay (148)

Listing: ./datutex/ex4.pp

```
Program Example4;

{ This program demonstrates the IsPM function }

Uses SysUtils,DateUtils;

Begin
  WriteLn('Current time is PM : ',IsPM(Now));
End.
```

44.4.61 IsSameDay

Synopsis: Check if two date/time indications are the same day.

Declaration:
```
function IsSameDay(const AValue: TDateTime;const ABasis: TDateTime)
                        : Boolean
```

Visibility: default

Description: `IsSameDay` checks whether `AValue` and `ABasis` have the same date part, and returns `True` if they do, `False` if not.

See also: Today (185), Yesterday (206), Tomorrow (185), IsToday (149)

Listing: ./datutex/ex21.pp

```
Program Example21;

{ This program demonstrates the IsSameDay function }

Uses SysUtils,DateUtils;

Var
  I : Integer;
  D : TDateTime;

Begin
  For I:=1 to 3 do
    begin
    D:=Today+Random(3)-1;
    Write(FormatDateTime('dd mmmm yyyy "is today : "',D));
    WriteLn(IsSameDay(D,Today));
    end;
End.
```

44.4.62 IsSameMonth

Synopsis: Check if 2 dates are in the same month.

Declaration:
```
function IsSameMonth(const Avalue: TDateTime;const ABasis: TDateTime)
                        : Boolean
```

Visibility: default

Description: `IsSameMonth` will return `True` if the two dates `Avalue` and `ABasis` occur in the same year and month. (i.e. if their month and year parts match). Otherwise, `False` is returned.

See also: IsSameDay (148), IsToday (149), SameDate (173)

44.4.63 IsToday

Synopsis: Check whether a given date is today.

Declaration: `function IsToday(const AValue: TDateTime) : Boolean`

Visibility: default

Description: `IsToday` returns `True` if `AValue` is today's date, and `False` otherwise.

See also: Today (185), Yesterday (206), Tomorrow (185), IsSameDay (148)

Listing: ./datutex/ex20.pp

```
Program Example20;

{ This program demonstrates the IsToday function }

Uses SysUtils, DateUtils;

Begin
  Writeln('Today     : ',IsToday(Today));
  Writeln('Tomorrow  : ',IsToday(Tomorrow));
  Writeln('Yesterday : ',IsToday(Yesterday));
End.
```

44.4.64 IsValidDate

Synopsis: Check whether a set of values is a valid date indication.

Declaration: `function IsValidDate(const AYear: Word;const AMonth: Word;`
`const ADay: Word) : Boolean`

Visibility: default

Description: `IsValidDate` returns `True` when the values `AYear`, `AMonth`, `ADay` form a valid date indication. If one of the values is not valid (e.g. the day is invalid or does not exist in that particular month), `False` is returned.

`AYear` must be in the range 1..9999 to be valid.

See also: IsValidTime (153), IsValidDateTime (151), IsValidDateDay (150), IsValidDateWeek (152), IsValidDateMonthWeek (150)

Listing: ./datutex/ex5.pp

```
Program Example5;

{ This program demonstrates the IsValidDate function }

Uses SysUtils, DateUtils;

Var
  Y,M,D : Word;

Begin
  For Y:=2000 to 2004 do
   For M:=1 to 12 do
     For D:=1 to 31 do
```

```
    If Not IsValidDate(Y,M,D) then
      Writeln(D,' is not a valid day in ',Y,'/',M);
End.
```

44.4.65 IsValidDateDay

Synopsis: Check whether a given year/day of year combination is a valid date.

Declaration:
```
function IsValidDateDay(const AYear: Word;const ADayOfYear: Word)
                        : Boolean
```

Visibility: default

Description: `IsValidDateDay` returns `True` if `AYear` and `ADayOfYear` form a valid date indication, or `False` otherwise.

`AYear` must be in the range 1..9999 to be valid.

The `ADayOfYear` value is checked to see whether it falls within the valid range of dates for `AYear`.

See also: IsValidDate (149), IsValidTime (153), IsValidDateTime (151), IsValidDateWeek (152), IsValidDateMonthWeek (150)

Listing: ./datutex/ex9.pp

```
Program Example9;

{ This program demonstrates the IsValidDateDay function }

Uses SysUtils, DateUtils;

Var
  Y : Word;

Begin
  For Y:=1996 to 2004 do
    if  IsValidDateDay(Y,366) then
      Writeln(Y,' is a leap year');
End.
```

44.4.66 IsValidDateMonthWeek

Synopsis: Check whether a given year/month/week/day of the week combination is a valid day

Declaration:
```
function IsValidDateMonthWeek(const AYear: Word;const AMonth: Word;
                              const AWeekOfMonth: Word;
                              const ADayOfWeek: Word) : Boolean
```

Visibility: default

Description: `IsValidDateMonthWeek` returns `True` if `AYear`, `AMonthAWeekOfMonth` and `ADayOfWeek` form a valid date indication, or `False` otherwise.

`AYear` must be in the range 1..9999 to be valid.

The `AWeekOfMonth`,`ADayOfWeek` values are checked to see whether the combination falls within the valid range of weeks for the `AYear`,`AMonth` combination.

See also: IsValidDate (149), IsValidTime (153), IsValidDateTime (151), IsValidDateDay (150), IsValidDateWeek (152)

Listing: ./datutex/ex11.pp

```
Program Example11;

{ This program demonstrates the IsValidDateMonthWeek function }

Uses SysUtils, DateUtils;

Var
  Y,W,D : Word;
  B : Boolean;

Begin
  For Y:=2000 to 2004 do
    begin
    B:=True;
    For W:=4 to 6 do
      For D:=1 to 7 do
        If B then
          begin
          B:=IsValidDateMonthWeek(Y,12,W,D);
          If Not B then
            if (D=1) then
              Writeln('December ',Y,' has exactly ',W,' weeks.')
            else
              Writeln('December ',Y,' has ',W,' weeks and ',D-1,' days.');
          end;
    end;
End.
```

44.4.67 IsValidDateTime

Synopsis: Check whether a set of values is a valid date and time indication.

Declaration:
```
function IsValidDateTime(const AYear: Word;const AMonth: Word;
                         const ADay: Word;const AHour: Word;
                         const AMinute: Word;const ASecond: Word;
                         const AMilliSecond: Word) : Boolean
```

Visibility: default

Description: `IsValidTime` returns `True` when the values `AYear`, `AMonth`, `ADay`, `AHour`, `AMinute`, `ASecond` and `AMilliSecond` form a valid date and time indication. If one of the values is not valid (e.g. the seconds are larger than 60), `False` is returned.

`AYear` must be in the range 1..9999 to be valid.

See also: IsValidDate (149), IsValidTime (153), IsValidDateDay (150), IsValidDateWeek (152), IsValidDateMonthWeek (150)

Listing: ./datutex/ex7.pp

```
Program Example7;

{ This program demonstrates the IsValidDateTime function }
```

```
Uses SysUtils,DateUtils;

Var
  Y,Mo,D : Word;
  H,M,S,MS : Word;
  I : Integer;

Begin
  For I:=1 to 10 do
    begin
    Y:=2000+Random(5);
    Mo:=Random(15);
    D:=Random(40);
    H:=Random(32);
    M:=Random(90);
    S:=Random(90);
    MS:=Random(1500);
    If Not IsValidDateTime(Y,Mo,D,H,M,S,MS) then
      WriteIn(Y,'-',Mo,'-',D,' ',H,':',M,':',S,'.',MS,' is not a valid date/time.');
    end;
End.
```

44.4.68 IsValidDateWeek

Synopsis: Check whether a given year/week/day of the week combination is a valid day.

Declaration:
```
function IsValidDateWeek(const AYear: Word;const AWeekOfYear: Word;
                         const ADayOfWeek: Word) : Boolean
```

Visibility: default

Description: `IsValidDateWeek` returns `True` if `AYear`, `AWeekOfYear` and `ADayOfWeek` form a valid date indication, or `False` otherwise.

`AYear` must be in the range 1..9999 to be valid.

The `ADayOfWeek`,`ADayOfWeek` values are checked to see whether the combination falls within the valid range of weeks for `AYear`.

See also: IsValidDate (149), IsValidTime (153), IsValidDateTime (151), IsValidDateDay (150), IsValidDateMonthWeek (150)

Listing: ./datutex/ex10.pp

```
Program Example10;

{ This program demonstrates the IsValidDateWeek function }

Uses SysUtils,DateUtils;

Var
  Y,W,D : Word;
  B : Boolean;

Begin
  For Y:=2000 to 2004 do
    begin
```

```
    B:=True;
    For W:=51 to 54 do
      For D:=1 to 7 do
        If B then
          begin
          B:=IsValidDateWeek(Y,W,D);
          If Not B then
            if (D=1) then
              Writeln(Y,' has exactly ',W,' weeks.')
            else
              Writeln(Y,' has ',W,' weeks and ',D-1,' days.');
          end;
    end;
End.
```

44.4.69 IsValidTime

Synopsis: Check whether a set of values is a valid time indication.

Declaration:
```
function IsValidTime(const AHour: Word;const AMinute: Word;
                     const ASecond: Word;const AMilliSecond: Word)
                      : Boolean
```

Visibility: default

Description: Check whether a set of values is a valid time indication.

44.4.70 JulianDateToDateTime

Synopsis: Convert a Julian date representation to a `TDateTime` value.

Declaration:
```
function JulianDateToDateTime(const AValue: Double) : TDateTime
```

Visibility: default

Description: `JulianDateToDateTime` converts the Julian `AValue` date/time indication to a regular `TDateTime` date/time indication.

See also: DateTimeToJulianDate (122), TryJulianDateToDateTime (189), DateTimeToModifiedJulianDate (123), TryModifiedJulianDateToDateTime (189)

44.4.71 LocalTimeToUniversal

Synopsis: Convert local time to UTC time

Declaration:
```
function LocalTimeToUniversal(LT: TDateTime) : TDateTime
function LocalTimeToUniversal(LT: TDateTime;TZOffset: Integer)
                              : TDateTime
```

Visibility: default

Description: `UniversalTimeToLocal` converts a local time indication to a universal time indication: it undoes the `TZOffset` timezone offset from the `UT` Universal time (UTC). If no `TZOffset` is specified, the local time offset as returned by GetLocalTimeOffset (116) is used.

See also: GetLocalTimeOffset (116), UniversalTimeToLocal (190)

44.4.72 MacTimeStampToUnix

Synopsis: Convert a Mac timestamp to a Unix timestamp

Declaration: `function MacTimeStampToUnix(const AValue: Int64) : Int64`

Visibility: default

Description: `MacTimeStampToUnix` converts the Mac timestamp indication in `AValue` to a unix timestamp indication (epoch time)

Errors: None.

See also: UnixTimeStampToMac (191), DateTimeToMac (122), MacToDateTime (154)

44.4.73 MacToDateTime

Synopsis: Convert a Mac timestamp to a `TDateTime` timestamp

Declaration: `function MacToDateTime(const AValue: Int64) : TDateTime`

Visibility: default

Description: `MacToDateTime` converts the Mac timestamp indication in `AValue` to a valid `TDateTime` indication.

Errors: None.

See also: UnixTimeStampToMac (191), DateTimeToMac (122), MacTimeStampToUnix (154)

44.4.74 MilliSecondOf

Synopsis: Extract the millisecond part from a DateTime value.

Declaration: `function MilliSecondOf(const AValue: TDateTime) : Word`

Visibility: default

Description: `MillisecondOf` returns the second of the minute part of the `AValue` date/time indication. It is a number between 0 and 999.

For an example, see YearOf (203)

See also: YearOf (203), WeekOf (191), MonthOf (162), DayOf (123), HourOf (138), MinuteOf (158), MilliSecondOf (154)

44.4.75 MilliSecondOfTheDay

Synopsis: Calculate the number of milliseconds elapsed since the start of the day

Declaration: `function MilliSecondOfTheDay(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `MilliSecondOfTheDay` returns the number of milliseconds that have passed since the start of the Day (00:00:00.000) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:00.000 will return 0.

For an example, see the HourOfTheDay (138) function.

See also: MilliSecondOfTheYear (156), MilliSecondOfTheMonth (155), MilliSecondOfTheWeek (156), MilliSecondOfTheHour (155), MilliSecondOfTheMinute (155), MilliSecondOfTheSecond (155), HourOfTheDay (138), MinuteOfTheDay (158), SecondOfTheDay (176)

44.4.76 MilliSecondOfTheHour

Synopsis: Calculate the number of milliseconds elapsed since the start of the hour

Declaration: `function MilliSecondOfTheHour(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `MilliSecondOfTheHour` returns the number of milliseconds that have passed since the start of the Hour (HH:00:00.000) till the moment indicated by `AValue`. This is a zero-based number, i.e. HH:00:00.000 will return 0.

For an example, see the MinuteOfTheHour (159) function.

See also: MilliSecondOfTheYear (156), MilliSecondOfTheMonth (155), MilliSecondOfTheWeek (156), MilliSecondOfTheDay (154), MilliSecondOfTheMinute (155), MilliSecondOfTheSecond (155), MinuteOfTheHour (159), SecondOfTheHour (176)

44.4.77 MilliSecondOfTheMinute

Synopsis: Calculate the number of milliseconds elapsed since the start of the minute

Declaration: `function MilliSecondOfTheMinute(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `MilliSecondOfTheMinute` returns the number of milliseconds that have passed since the start of the Minute (HH:MM:00.000) till the moment indicated by `AValue`. This is a zero-based number, i.e. HH:MM:00.000 will return 0.

For an example, see the SecondOfTheMinute (176) function.

See also: MilliSecondOfTheYear (156), MilliSecondOfTheMonth (155), MilliSecondOfTheWeek (156), MilliSecondOfTheDay (154), MilliSecondOfTheHour (155), MilliSecondOfTheMinute (155), MilliSecondOfTheSecond (155), SecondOfTheMinute (176)

44.4.78 MilliSecondOfTheMonth

Synopsis: Calculate number of milliseconds elapsed since the start of the month.

Declaration: `function MilliSecondOfTheMonth(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `MilliSecondOfTheMonth` returns the number of milliseconds that have passed since the start of the month (00:00:00.000) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:00.000 on the first of the month will return 0.

For an example, see the WeekOfTheMonth (192) function.

See also: WeekOfTheMonth (192), DayOfTheMonth (123), HourOfTheMonth (139), MinuteOfTheMonth (159), SecondOfTheMonth (177), MilliSecondOfTheMonth (155)

44.4.79 MilliSecondOfTheSecond

Synopsis: Calculate the number of milliseconds elapsed since the start of the second

Declaration: `function MilliSecondOfTheSecond(const AValue: TDateTime) : Word`

Visibility: default

Description: `MilliSecondOfTheSecond` returns the number of milliseconds that have passed since the start of the second (HH:MM:SS.000) till the moment indicated by `AValue`. This is a zero-based number, i.e. HH:MM:SS.000 will return 0.

See also: MilliSecondOfTheYear (156), MilliSecondOfTheMonth (155), MilliSecondOfTheWeek (156), MilliSecondOfTheDay (154), MilliSecondOfTheHour (155), MilliSecondOfTheMinute (155), SecondOfTheMinute (176)

Listing: ./datutex/ex46.pp

```
Program Example46;

{ This program demonstrates the MilliSecondOfTheSecond function }

Uses SysUtils, DateUtils;

Var
  N : TDateTime;

Begin
  N:=Now;
  Writeln('MilliSecond of the Second : ',
          MilliSecondOfTheSecond(N));
End.
```

44.4.80 MilliSecondOfTheWeek

Synopsis: Calculate the number of milliseconds elapsed since the start of the week

Declaration: `function MilliSecondOfTheWeek(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `MilliSecondOfTheWeek` returns the number of milliseconds that have passed since the start of the Week (00:00:00.000) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:00.000 on the first of the Week will return 0.

For an example, see the DayOfTheWeek (124) function.

See also: MilliSecondOfTheYear (156), MilliSecondOfTheMonth (155), MilliSecondOfTheDay (154), MilliSecondOfTheHour (155), MilliSecondOfTheMinute (155), MilliSecondOfTheSecond (155), DayOfTheWeek (124), HourOfTheWeek (139), MinuteOfTheWeek (160), SecondOfTheWeek (177)

44.4.81 MilliSecondOfTheYear

Synopsis: Calculate the number of milliseconds elapsed since the start of the year.

Declaration: `function MilliSecondOfTheYear(const AValue: TDateTime) : Int64`

Visibility: default

Description: `MilliSecondOfTheYear` returns the number of milliseconds that have passed since the start of the year (January 1, 00:00:00.000) till the moment indicated by `AValue`. This is a zero-based number, i.e. January 1 00:00:00.000 will return 0.

For an example, see the WeekOfTheYear (192) function.

See also: WeekOfTheYear (192), DayOfTheYear (124), HourOfTheYear (140), MinuteOfTheYear (160), SecondOfTheYear (177), MilliSecondOfTheYear (156)

44.4.82 MilliSecondsBetween

Synopsis: Calculate the number of whole milliseconds between two DateTime values.

Declaration:
```
function MilliSecondsBetween(const ANow: TDateTime;
                             const AThen: TDateTime) : Int64
```

Visibility: default

Description: `MillisSecondsBetween` returns the number of whole milliseconds between `ANow` and `AThen`. This means a fractional part of a millisecond is dropped.

See also: YearsBetween (204), MonthsBetween (163), WeeksBetween (193), DaysBetween (124), HoursBetween (140), MinutesBetween (160), SecondsBetween (178)

Listing: ./datutex/ex62.pp

```
Program Example62;

{ This program demonstrates the MilliSecondsBetween function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
  Write('Number of milliseconds between ');
  Write(TimeToStr(AThen),' and ',TimeToStr(ANow));
  WriteIn(' : ',MilliSecondsBetween(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=D1-(0.9*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(1.0*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(1.1*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(2.5*OneMilliSecond);
  Test(D1,D2);
End.
```

44.4.83 MilliSecondSpan

Synopsis: Calculate the approximate number of milliseconds between two DateTime values.

Declaration:
```
function MilliSecondSpan(const ANow: TDateTime;const AThen: TDateTime)
                         : Double
```

Visibility: default

Description: `MilliSecondSpan` returns the number of milliseconds between `ANow` and `AThen`. Since millisecond is the smallest fraction of a `TDateTime` indication, the returned number will always be an integer value.

See also: YearSpan (205), MonthSpan (164), WeekSpan (195), DaySpan (127), HourSpan (141), MinuteSpan (161), SecondSpan (179), MilliSecondsBetween (157)

Listing: ./datutex/ex70.pp

```
Program Example70;

{ This program demonstrates the MilliSecondSpan function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
  Write('Number of milliseconds between ');
  Write(TimeToStr(AThen),' and ',TimeToStr(ANow));
  WriteIn(' : ',MilliSecondSpan(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=D1-(0.9*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(1.0*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(1.1*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(2.5*OneMilliSecond);
  Test(D1,D2);
End.
```

44.4.84 MinuteOf

Synopsis: Extract the minute part from a DateTime value.

Declaration: `function MinuteOf(const AValue: TDateTime) : Word`

Visibility: default

Description: `MinuteOf` returns the minute of the hour part of the `AValue` date/time indication. It is a number between 0 and 59.

For an example, see YearOf (203)

See also: YearOf (203), WeekOf (191), MonthOf (162), DayOf (123), HourOf (138), SecondOf (175), MilliSecondOf (154)

44.4.85 MinuteOfTheDay

Synopsis: Calculate the number of minutes elapsed since the start of the day

Declaration: `function MinuteOfTheDay(const AValue: TDateTime) : Word`

Visibility: default

Description: `MinuteOfTheDay` returns the number of minutes that have passed since the start of the Day (00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:59 will return 0.

For an example, see the HourOfTheDay (138) function.

See also: MinuteOfTheYear (160), MinuteOfTheMonth (159), MinuteOfTheWeek (160), MinuteOfTheHour (159), HourOfTheDay (138), SecondOfTheDay (176), MilliSecondOfTheDay (154)

44.4.86 MinuteOfTheHour

Synopsis: Calculate the number of minutes elapsed since the start of the hour

Declaration: `function MinuteOfTheHour(const AValue: TDateTime) : Word`

Visibility: default

Description: `MinuteOfTheHour` returns the number of minutes that have passed since the start of the Hour (HH:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. HH:00:59 will return 0.

See also: MinuteOfTheYear (160), MinuteOfTheMonth (159), MinuteOfTheWeek (160), MinuteOfTheDay (158), SecondOfTheHour (176), MilliSecondOfTheHour (155)

Listing: ./datutex/ex44.pp

```
Program Example44;

{ This program demonstrates the MinuteOfTheHour function }

Uses SysUtils, DateUtils;

Var
  N : TDateTime;

Begin
  N:=Now;
  Writeln('Minute of the Hour       : ',MinuteOfTheHour(N));
  Writeln('Second of the Hour       : ',SecondOfTheHour(N));
  Writeln('MilliSecond of the Hour : ',
          MilliSecondOfTheHour(N));
End.
```

44.4.87 MinuteOfTheMonth

Synopsis: Calculate number of minutes elapsed since the start of the month.

Declaration: `function MinuteOfTheMonth(const AValue: TDateTime) : Word`

Visibility: default

Description: `MinuteOfTheMonth` returns the number of minutes that have passed since the start of the Month (00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:59 on the first day of the month will return 0.

For an example, see the WeekOfTheMonth (192) function.

See also: WeekOfTheMonth (192), DayOfTheMonth (123), HourOfTheMonth (139), MinuteOfTheMonth (159), SecondOfTheMonth (177), MilliSecondOfTheMonth (155)

44.4.88 MinuteOfTheWeek

Synopsis: Calculate the number of minutes elapsed since the start of the week

Declaration: `function MinuteOfTheWeek(const AValue: TDateTime) : Word`

Visibility: default

Description: `MinuteOfTheWeek` returns the number of minutes that have passed since the start of the week (00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:59 on the first day of the week will return 0.

For an example, see the DayOfTheWeek (124) function.

See also: MinuteOfTheYear (160), MinuteOfTheMonth (159), MinuteOfTheDay (158), MinuteOfTheHour (159), DayOfTheWeek (124), HourOfTheWeek (139), SecondOfTheWeek (177), MilliSecondOfTheWeek (156)

44.4.89 MinuteOfTheYear

Synopsis: Calculate the number of minutes elapsed since the start of the year

Declaration: `function MinuteOfTheYear(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `MinuteOfTheYear` returns the number of minutes that have passed since the start of the year (January 1, 00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. January 1 00:00:59 will return 0.

For an example, see the WeekOfTheYear (192) function.

See also: WeekOfTheYear (192), DayOfTheYear (124), HourOfTheYear (140), MinuteOfTheYear (160), SecondOfTheYear (177), MilliSecondOfTheYear (156)

44.4.90 MinutesBetween

Synopsis: Calculate the number of whole minutes between two DateTime values.

Declaration:
```
function MinutesBetween(const ANow: TDateTime;const AThen: TDateTime)
                        : Int64
```

Visibility: default

Description: `MinutesBetween` returns the number of whole minutes between `ANow` and `AThen`. This means the fractional part of a minute (seconds, milliseconds etc.) is dropped.

See also: YearsBetween (204), MonthsBetween (163), WeeksBetween (193), DaysBetween (124), HoursBetween (140), SecondsBetween (178), MilliSecondsBetween (157)

Listing: ./datutex/ex60.pp

```
Program Example60;

{ This program demonstrates the MinutesBetween function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime);
```

```
begin
  Write('Number of minutes between ');
  Write(TimeToStr(AThen),' and ',TimeToStr(ANow));
  Writeln(' : ',MinutesBetween(ANow,AThen));
end;

Var
   D1,D2 : TDateTime;

Begin
   D1:=Now;
   D2:=D1-(59*OneSecond);
   Test(D1,D2);
   D2:=D1-(61*OneSecond);
   Test(D1,D2);
   D2:=D1-(122*OneSecond);
   Test(D1,D2);
   D2:=D1-(306*OneSecond);
   Test(D1,D2);
   D2:=D1-(5.4*OneMinute);
   Test(D1,D2);
   D2:=D1-(2.5*OneMinute);
   Test(D1,D2);
End.
```

44.4.91 MinuteSpan

Synopsis: Calculate the approximate number of minutes between two DateTime values.

Declaration:
```
function MinuteSpan(const ANow: TDateTime;const AThen: TDateTime)
                          : Double
```

Visibility: default

Description: `MinuteSpan` returns the number of minutes between `ANow` and `AThen`, including any fractional parts of a minute.

See also: YearSpan (205), MonthSpan (164), WeekSpan (195), DaySpan (127), HourSpan (141), SecondSpan (179), MilliSecondSpan (157), MinutesBetween (160)

Listing: ./datutex/ex68.pp

```
Program Example68;

{ This program demonstrates the MinuteSpan function }

Uses SysUtils,DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
  Write('Number of minutes between ');
  Write(TimeToStr(AThen),' and ',TimeToStr(ANow));
  Writeln(' : ',MinuteSpan(ANow,AThen));
end;

Var
```

```
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=D1-(59*OneSecond);
  Test(D1,D2);
  D2:=D1-(61*OneSecond);
  Test(D1,D2);
  D2:=D1-(122*OneSecond);
  Test(D1,D2);
  D2:=D1-(306*OneSecond);
  Test(D1,D2);
  D2:=D1-(5.4*OneMinute);
  Test(D1,D2);
  D2:=D1-(2.5*OneMinute);
  Test(D1,D2);
End.
```

44.4.92 ModifiedJulianDateToDateTime

Synopsis: Convert a modified Julian date representation to a `TDateTime` value.

Declaration: `function ModifiedJulianDateToDateTime(const AValue: Double) : TDateTime`

Visibility: default

Description: Not yet implemented.

Errors: Currently, trying to use this function will raise an exception.

See also: DateTimeToJulianDate (122), JulianDateToDateTime (153), TryJulianDateToDateTime (189), DateTimeToModifiedJulianDate (123), TryModifiedJulianDateToDateTime (189)

44.4.93 MonthOf

Synopsis: Extract the month from a given date.

Declaration: `function MonthOf(const AValue: TDateTime) : Word`

Visibility: default

Description: `MonthOf` returns the month part of the `AValue` date/time indication. It is a number between 1 and 12.

For an example, see YearOf (203)

See also: YearOf (203), DayOf (123), WeekOf (191), HourOf (138), MinuteOf (158), SecondOf (175), MilliSecondOf (154)

44.4.94 MonthOfTheYear

Synopsis: Extract the month of a DateTime indication.

Declaration: `function MonthOfTheYear(const AValue: TDateTime) : Word`

Visibility: default

Description: `MonthOfTheYear` extracts the month part of `Avalue` and returns it. It is an alias for MonthOf (162), and is provided for completeness only, corresponding to the other `PartOfTheYear` functions.

For an example, see the WeekOfTheYear (192) function.

See also: MonthOf (162), WeekOfTheYear (192), DayOfTheYear (124), HourOfTheYear (140), MinuteOfTheYear (160), SecondOfTheYear (177), MilliSecondOfTheYear (156)

44.4.95 MonthsBetween

Synopsis: Calculate the number of whole months between two DateTime values

Declaration:
```
function MonthsBetween(const ANow: TDateTime;const AThen: TDateTime)
                       : Integer
```

Visibility: default

Description: `MonthsBetween` returns the number of whole months between `ANow` and `AThen`. This number is an approximation, based on an average number of days of 30.4375 per month (average over 4 years). This means the fractional part of a month is dropped.

See also: YearsBetween (204), WeeksBetween (193), DaysBetween (124), HoursBetween (140), MinutesBetween (160), SecondsBetween (178), MilliSecondsBetween (157)

Listing: ./datutex/ex56.pp

```
Program Example56;

{ This program demonstrates the MonthsBetween function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
  Write('Number of months between ');
  Write(DateToStr(AThen),' and ',DateToStr(ANow));
  Writeln(' : ',MonthsBetween(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Today;
  D2:=Today-364;
  Test(D1,D2);
  D2:=Today-365;
  Test(D1,D2);
  D2:=Today-366;
  Test(D1,D2);
  D2:=Today-390;
  Test(D1,D2);
  D2:=Today-368;
  Test(D1,D2);
  D2:=Today-1000;
  Test(D1,D2);
End.
```

44.4.96 MonthSpan

Synopsis: Calculate the approximate number of months between two DateTime values.

Declaration:
```
function MonthSpan(const ANow: TDateTime;const AThen: TDateTime)
                         : Double
```

Visibility: default

Description: `MonthSpan` returns the number of month between `ANow` and `AThen`, including any fractional parts of a month. This number is an approximation, based on an average number of days of 30.4375 per month (average over 4 years).

See also: YearSpan (205), WeekSpan (195), DaySpan (127), HourSpan (141), MinuteSpan (161), SecondSpan (179), MilliSecondSpan (157), MonthsBetween (163)

Listing: ./datutex/ex64.pp

```
Program Example64;

{ This program demonstrates the MonthSpan function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
  Write('Number of months between ');
  Write(DateToStr(AThen),' and ',DateToStr(ANow));
  Writeln(' : ',MonthSpan(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Today;
  D2:=Today-364;
  Test(D1,D2);
  D2:=Today-365;
  Test(D1,D2);
  D2:=Today-366;
  Test(D1,D2);
  D2:=Today-390;
  Test(D1,D2);
  D2:=Today-368;
  Test(D1,D2);
  D2:=Today-1000;
  Test(D1,D2);
End.
```

44.4.97 NthDayOfWeek

Synopsis: Calculate which occurrence of weekday in the month a given day represents

Declaration:
```
function NthDayOfWeek(const AValue: TDateTime) : Word
```

Visibility: default

Description: `NthDayOfWeek` returns the occurrence of the weekday of `AValue` in the month. This is the N-th time that this weekday occurs in the month (e.g. the third saturday of the month).

See also: EncodeDateMonthWeek (132), #rtl.sysutils.DayOfWeek (628), DecodeDayOfWeekInMonth (131), EncodeDayOfWeekInMonth (133), TryEncodeDayOfWeekInMonth (188)

Listing: ./datutex/ex104.pp

```
Program Example104;

{ This program demonstrates the NthDayOfWeek function }

Uses SysUtils, DateUtils;

Begin
  Write('Today is the ',NthDayOfWeek(Today),'-th ');
  WriteLn(formatdateTime('dddd',Today),' of the month.');
End.
```

44.4.98 PeriodBetween

Synopsis: Return the period (in years, months, days) between two dates

Declaration:
```
procedure PeriodBetween(const ANow: TDateTime;const AThen: TDateTime;
                        out Years: Word;out months: Word;out days: Word)
```

Visibility: default

Description: `PeriodBetween` returns the timespan between 2 dates (`ANow` and `AThen`), expressed as a number of years, months and days in the parameters `Years`, `months` and `days`. Only complete years, months and days are reported.

If `ANow` is before `AThen`, their values are reversed so the result is always positive.

See also: YearsBetween (204), MonthsBetween (163), WeeksBetween (193), DaysBetween (124)

44.4.99 PreviousDayOfWeek

Synopsis: Given a day of the week, return the previous day of the week.

Declaration:
```
function PreviousDayOfWeek(DayOfWeek: Word) : Word
```

Visibility: default

Description: `PreviousDayOfWeek` returns the previous day of the week. If the current day is the first day of the week (1) then the last day will be returned (7).

Remark: Note that the days of the week are in ISO notation, i.e. 1-based.

See also: Yesterday (206)

Listing: ./datutex/ex22.pp

```
Program Example22;

{ This program demonstrates the PreviousDayOfWeek function }

Uses SysUtils, DateUtils;
```

```
Var
  D : Word;

Begin
  For D:=1 to 7 do
    WriteIn('Previous day of ',D,' is : ',PreviousDayOfWeek(D));
End.
```

44.4.100 RecodeDate

Synopsis: Replace date part of a `TDateTime` value with another date.

Declaration:
```
function RecodeDate(const AValue: TDateTime;const AYear: Word;
                    const AMonth: Word;const ADay: Word) : TDateTime
```

Visibility: default

Description: `RecodeDate` replaces the date part of the timestamp `AValue` with the date specified in `AYear`, `AMonth`, `ADay`. All other parts (the time part) of the date/time stamp are left untouched.

Errors: If one of the `AYear`, `AMonth`, `ADay` values is not within a valid range then an `EConvertError` exception is raised.

See also: RecodeYear (172), RecodeMonth (170), RecodeDay (167), RecodeHour (168), RecodeMinute (169), RecodeSecond (171), RecodeDate (166), RecodeTime (171), RecodeDateTime (166)

Listing: ./datutex/ex94.pp

```
Program Example94;

{ This program demonstrates the RecodeDate function }

Uses SysUtils, DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var
  S : AnsiString;

Begin
  S:=FormatDateTime(Fmt,RecodeDate(Now,2001,1,1));
  WriteIn('This moment on the first of the millenium : ',S);
End.
```

44.4.101 RecodeDateTime

Synopsis: Replace selected parts of a `TDateTime` value with other values

Declaration:
```
function RecodeDateTime(const AValue: TDateTime;const AYear: Word;
                        const AMonth: Word;const ADay: Word;
                        const AHour: Word;const AMinute: Word;
                        const ASecond: Word;const AMilliSecond: Word)
                         : TDateTime
```

Visibility: default

Description: `RecodeDateTime` replaces selected parts of the timestamp `AValue` with the date/time values specified in `AYear`, `AMonth`, `ADay`, `AHour`, `AMinute`, `ASecond` and `AMilliSecond`. If any of these values equals the pre-defined constant RecodeLeaveFieldAsIs (118), then the corresponding part of the date/time stamp is left untouched.

Errors: If one of the values `AYear`, `AMonth`, `ADay`, `AHour`, `AMinute`, `ASecondAMilliSecond` is not within a valid range (`RecodeLeaveFieldAsIs` excepted) then an `EConvertError` exception is raised.

See also: RecodeYear (172), RecodeMonth (170), RecodeDay (167), RecodeHour (168), RecodeMinute (169), RecodeSecond (171), RecodeMilliSecond (169), RecodeDate (166), RecodeTime (171), TryRecodeDateTime (190)

Listing: ./datutex/ex96.pp

```
Program Example96;

{ This program demonstrates the RecodeDateTime function }

Uses SysUtils, DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var
  S : AnsiString;
  D : TDateTime ;

Begin
  D:=Now;
  D:=RecodeDateTime(D,2000,2,RecodeLeaveFieldAsIs,0,0,0,0);
  S:=FormatDateTime(Fmt,D);
  Writeln('This moment in februari 2000 : ',S);
End.
```

44.4.102 RecodeDay

Synopsis: Replace day part of a `TDateTime` value with another day.

Declaration: `function RecodeDay(const AValue: TDateTime;const ADay: Word) : TDateTime`

Visibility: default

Description: `RecodeDay` replaces the Day part of the timestamp `AValue` with `ADay`. All other parts of the date/time stamp are left untouched.

Errors: If the `ADay` value is not within a valid range (1 till the number of days in the month) then an `EConvertError` exception is raised.

See also: RecodeYear (172), RecodeMonth (170), RecodeHour (168), RecodeMinute (169), RecodeSecond (171), RecodeMilliSecond (169), RecodeDate (166), RecodeTime (171), RecodeDateTime (166)

Listing: ./datutex/ex89.pp

```
Program Example89;

{ This program demonstrates the RecodeDay function }

Uses SysUtils,DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var
  S : AnsiString;

Begin
  S:=FormatDateTime(Fmt,RecodeDay(Now,1));
  Writeln('This moment on the first of the month : ',S);
End.
```

44.4.103 RecodeHour

Synopsis: Replace hours part of a `TDateTime` value with another hour.

Declaration:
```
function RecodeHour(const AValue: TDateTime;const AHour: Word)
                          : TDateTime
```

Visibility: default

Description: `RecodeHour` replaces the Hour part of the timestamp `AValue` with `AHour`. All other parts of the date/time stamp are left untouched.

Errors: If the `AHour` value is not within a valid range (0..23) then an `EConvertError` exception is raised.

See also: RecodeYear (172), RecodeMonth (170), RecodeDay (167), RecodeMinute (169), RecodeSecond (171), RecodeMilliSecond (169), RecodeDate (166), RecodeTime (171), RecodeDateTime (166)

Listing: ./datutex/ex90.pp

```
Program Example90;

{ This program demonstrates the RecodeHour function }

Uses SysUtils,DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var
  S : AnsiString;

Begin
  S:=FormatDateTime(Fmt,RecodeHour(Now,0));
  Writeln('Today, in the first hour : ',S);
End.
```

44.4.104 RecodeMilliSecond

Synopsis: Replace milliseconds part of a `TDateTime` value with another millisecond.

Declaration:
```
function RecodeMilliSecond(const AValue: TDateTime;
                           const AMilliSecond: Word) : TDateTime
```

Visibility: default

Description: `RecodeMilliSecond` replaces the millisecond part of the timestamp `AValue` with `AMilliSecond`. All other parts of the date/time stamp are left untouched.

Errors: If the `AMilliSecond` value is not within a valid range (0..999) then an `EConvertError` exception is raised.

See also: RecodeYear (172), RecodeMonth (170), RecodeDay (167), RecodeHour (168), RecodeMinute (169), RecodeSecond (171), RecodeDate (166), RecodeTime (171), RecodeDateTime (166)

Listing: ./datutex/ex93.pp

```
Program Example93;

{ This program demonstrates the RecodeMilliSecond function }

Uses SysUtils, DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss.zzz';

Var
  S : AnsiString;

Begin
  S:=FormatDateTime(Fmt, RecodeMilliSecond(Now,0));
  Writeln('This moment, milliseconds stripped : ',S);
End.
```

44.4.105 RecodeMinute

Synopsis: Replace minutse part of a `TDateTime` value with another minute.

Declaration:
```
function RecodeMinute(const AValue: TDateTime;const AMinute: Word)
                      : TDateTime
```

Visibility: default

Description: `RecodeMinute` replaces the Minute part of the timestamp `AValue` with `AMinute`. All other parts of the date/time stamp are left untouched.

Errors: If the `AMinute` value is not within a valid range (0..59) then an `EConvertError` exception is raised.

See also: RecodeYear (172), RecodeMonth (170), RecodeDay (167), RecodeHour (168), RecodeSecond (171), RecodeMilliSecond (169), RecodeDate (166), RecodeTime (171), RecodeDateTime (166)

Listing: ./datutex/ex91.pp

```
Program Example91;

{ This program demonstrates the RecodeMinute function }

Uses SysUtils,DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var
  S : AnsiString;

Begin
  S:=FormatDateTime(Fmt,RecodeMinute(Now,0));
  Writeln('This moment in the first minute of the hour: ',S);
End.
```

44.4.106 RecodeMonth

Synopsis: Replace month part of a `TDateTime` value with another month.

Declaration:
```
function RecodeMonth(const AValue: TDateTime;const AMonth: Word)
                          : TDateTime
```

Visibility: default

Description: `RecodeMonth` replaces the Month part of the timestamp `AValue` with `AMonth`. All other parts of the date/time stamp are left untouched.

Errors: If the `AMonth` value is not within a valid range (1..12) then an `EConvertError` exception is raised.

See also: RecodeYear (172), RecodeDay (167), RecodeHour (168), RecodeMinute (169), RecodeSecond (171), RecodeMilliSecond (169), RecodeDate (166), RecodeTime (171), RecodeDateTime (166)

Listing: ./datutex/ex88.pp

```
Program Example88;

{ This program demonstrates the RecodeMonth function }

Uses SysUtils,DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var
  S : AnsiString;

Begin
  S:=FormatDateTime(Fmt,RecodeMonth(Now,5));
  Writeln('This moment in May : ',S);
End.
```

44.4.107 RecodeSecond

Synopsis: Replace seconds part of a `TDateTime` value with another second.

Declaration:
```
function RecodeSecond(const AValue: TDateTime;const ASecond: Word)
                          : TDateTime
```

Visibility: default

Description: `RecodeSecond` replaces the Second part of the timestamp `AValue` with `ASecond`. All other parts of the date/time stamp are left untouched.

Errors: If the `ASecond` value is not within a valid range (0..59) then an `EConvertError` exception is raised.

See also: RecodeYear (172), RecodeMonth (170), RecodeDay (167), RecodeHour (168), RecodeMinute (169), RecodeMilliSecond (169), RecodeDate (166), RecodeTime (171), RecodeDateTime (166)

Listing: ./datutex/ex92.pp

```
Program Example92;

{ This program demonstrates the RecodeSecond function }

Uses SysUtils, DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var
  S : AnsiString;

Begin
  S:=FormatDateTime(Fmt,RecodeSecond(Now,0));
  Writeln('This moment, seconds stripped : ',S);
End.
```

44.4.108 RecodeTime

Synopsis: Replace time part of a `TDateTime` value with another time.

Declaration:
```
function RecodeTime(const AValue: TDateTime;const AHour: Word;
                        const AMinute: Word;const ASecond: Word;
                        const AMilliSecond: Word) : TDateTime
```

Visibility: default

Description: `RecodeTime` replaces the time part of the timestamp `AValue` with the date specified in `AHour`, `AMinute`, `ASecond` and `AMilliSecond`. All other parts (the date part) of the date/time stamp are left untouched.

Errors: If one of the values `AHour`, `AMinute`, `ASecondAMilliSecond` is not within a valid range then an `EConvertError` exception is raised.

See also: RecodeYear (172), RecodeMonth (170), RecodeDay (167), RecodeHour (168), RecodeMinute (169), RecodeSecond (171), RecodeMilliSecond (169), RecodeDate (166), RecodeDateTime (166)

Listing: ./datutex/ex95.pp

```
Program Example95;

{ This program demonstrates the RecodeTime function }

Uses SysUtils,DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var
  S : AnsiString;

Begin
  S:=FormatDateTime(Fmt,RecodeTime(Now,8,0,0,0));
  Writeln('Today, 8 AM : ',S);
End.
```

44.4.109 RecodeYear

Synopsis: Replace year part of a `TDateTime` value with another year.

Declaration:
```
function RecodeYear(const AValue: TDateTime;const AYear: Word)
                          : TDateTime
```

Visibility: default

Description: `RecodeYear` replaces the year part of the timestamp `AValue` with `AYear`. All other parts of the date/time stamp are left untouched.

Errors: If the `AYear` value is not within a valid range (1..9999) then an `EConvertError` exception is raised.

See also: RecodeMonth (170), RecodeDay (167), RecodeHour (168), RecodeMinute (169), RecodeSecond (171), RecodeMilliSecond (169), RecodeDate (166), RecodeTime (171), RecodeDateTime (166)

Listing: ./datutex/ex87.pp

```
Program Example87;

{ This program demonstrates the RecodeYear function }

Uses SysUtils,DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var
  S : AnsiString;

Begin
  S:=FormatDateTime(Fmt,RecodeYear(Now,1999));
  Writeln('This moment in 1999 : ',S);
End.
```

44.4.110 SameDate

Synopsis: Check whether two `TDateTime` values have the same date part.

Declaration: `function SameDate(const A: TDateTime;const B: TDateTime) : Boolean`

Visibility: default

Description: `SameDate` compares the date parts of two timestamps `A` and `B` and returns `True` if they are equal, `False` if they are not.

The function simply checks whether CompareDate (118) returns zero.

See also: CompareDateTime (119), CompareDate (118), CompareTime (120), SameDateTime (173), SameTime (174)

Listing: ./datutex/ex102.pp

```
Program Example102;

{ This program demonstrates the SameDate function }

Uses SysUtils, DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss.zzz';

Procedure Test(D1,D2 : TDateTime);

begin
  Write(FormatDateTime(Fmt,D1),' is the same date as ');
  WriteIn(FormatDateTime(Fmt,D2),' : ',SameDate(D1,D2));
end;

Var
  D,N : TDateTime;

Begin
  D:=Today;
  N:=Now;
  Test(D,D);
  Test(N,N);
  Test(N+1,N);
  Test(N-1,N);
  Test(N+OneSecond,N);
  Test(N-OneSecond,N);
End.
```

44.4.111 SameDateTime

Synopsis: Check whether two `TDateTime` values have the same date and time parts.

Declaration: `function SameDateTime(const A: TDateTime;const B: TDateTime) : Boolean`

Visibility: default

Description: `SameDateTime` compares the date/time parts of two timestamps `A` and `B` and returns `True` if they are equal, `False` if they are not.

The function simply checks whether CompareDateTime (119) returns zero.

See also: CompareDateTime (119), CompareDate (118), CompareTime (120), SameDate (173), SameTime (174)

Listing: ./datutex/ex101.pp

```
Program Example101;

{ This program demonstrates the SameDateTime function }

Uses SysUtils,DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss.zzz';

Procedure Test(D1,D2 : TDateTime);

begin
  Write(FormatDateTime(Fmt,D1),' is the same datetime as ');
  WriteIn(FormatDateTime(Fmt,D2),' : ',SameDateTime(D1,D2));
end;

Var
  D,N : TDateTime;

Begin
  D:=Today;
  N:=Now;
  Test(D,D);
  Test(N,N);
  Test(N+1,N);
  Test(N-1,N);
  Test(N+OneSecond,N);
  Test(N-OneSecond,N);
End.
```

44.4.112 SameTime

Synopsis: Check whether two `TDateTime` values have the same time part.

Declaration: `function SameTime(const A: TDateTime;const B: TDateTime) : Boolean`

Visibility: default

Description: `SameTime` compares the time parts of two timestamps `A` and `B` and returns `True` if they are equal, `False` if they are not.

The function simply checks whether CompareTime (120) returns zero.

See also: CompareDateTime (119), CompareDate (118), CompareTime (120), SameDateTime (173), SameDate (173)

Listing: ./datutex/ex103.pp

```
Program Example102;

{ This program demonstrates the SameTime function }

Uses SysUtils,DateUtils;
```

```
Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss.zzz';

Procedure Test(D1,D2 : TDateTime);

begin
  Write(FormatDateTime(Fmt,D1),' is the same time as ');
  Writeln(FormatDateTime(Fmt,D2),' : ',SameTime(D1,D2));
end;

Var
  D,N : TDateTime;

Begin
  D:=Today;
  N:=Now;
  Test(D,D);
  Test(N,N);
  Test(N+1,N);
  Test(N-1,N);
  Test(N+OneSecond,N);
  Test(N-OneSecond,N);
End.
```

44.4.113 ScanDateTime

Synopsis: Scans a string for a DateTime pattern and returns the date/time

Declaration:
```
function ScanDateTime(const Pattern: string;const s: string;
                      const fmt: TFormatSettings;startpos: Integer)
                       : tdatetime;  Overload
function ScanDateTime(const Pattern: string;const s: string;
                      startpos: Integer) : tdatetime;  Overload
```

Visibility: default

Description: `ScanDateTime` scans string `S` for the date/time pattern `Pattern`, starting at position `StartPos` (default 1). Optionally, the format settings `fmt` can be specified.

In effect, this function does the opposite of what FormatDateTime (628) does. The `Pattern` variable must contain a valid date/time pattern: note that not all possible formatdatetime patterns can be recognized, e.g., `hn` cannot be detected properly.

Errors: In case of an error, a EConvertError (628) exception is raised.

See also: FormatDateTime (628)

44.4.114 SecondOf

Synopsis: Extract the second part from a DateTime value.

Declaration:
```
function SecondOf(const AValue: TDateTime) : Word
```

Visibility: default

Description: `SecondOf` returns the second of the minute part of the `AValue` date/time indication. It is a number between 0 and 59.

For an example, see YearOf (203)

See also: YearOf (203), WeekOf (191), MonthOf (162), DayOf (123), HourOf (138), MinuteOf (158), MilliSecondOf (154)

44.4.115 SecondOfTheDay

Synopsis: Calculate the number of seconds elapsed since the start of the day

Declaration: `function SecondOfTheDay(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `SecondOfTheDay` returns the number of seconds that have passed since the start of the Day (00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:00.999 return 0.

For an example, see the HourOfTheDay (138) function.

See also: SecondOfTheYear (177), SecondOfTheMonth (177), SecondOfTheWeek (177), SecondOfTheHour (176), SecondOfTheMinute (176), HourOfTheDay (138), MinuteOfTheDay (158), MilliSecondOfTheDay (154)

44.4.116 SecondOfTheHour

Synopsis: Calculate the number of seconds elapsed since the start of the hour

Declaration: `function SecondOfTheHour(const AValue: TDateTime) : Word`

Visibility: default

Description: `SecondOfTheHour` returns the number of seconds that have passed since the start of the Hour (HH:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. HH:00:00.999 return 0.

For an example, see the MinuteOfTheHour (159) function.

See also: SecondOfTheYear (177), SecondOfTheMonth (177), SecondOfTheWeek (177), SecondOfTheDay (176), SecondOfTheMinute (176), MinuteOfTheHour (159), MilliSecondOfTheHour (155)

44.4.117 SecondOfTheMinute

Synopsis: Calculate the number of seconds elapsed since the start of the minute

Declaration: `function SecondOfTheMinute(const AValue: TDateTime) : Word`

Visibility: default

Description: `SecondOfTheMinute` returns the number of seconds that have passed since the start of the minute (HH:MM:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. HH:MM:00.999 return 0.

See also: SecondOfTheYear (177), SecondOfTheMonth (177), SecondOfTheWeek (177), SecondOfTheDay (176), SecondOfTheHour (176), MilliSecondOfTheMinute (155)

Listing: ./datutex/ex45.pp

```
Program Example45;

{ This program demonstrates the SecondOfTheMinute function }

Uses SysUtils,DateUtils;

Var
  N : TDateTime;

Begin
  N:=Now;
  Writeln('Second of the Minute      : ',SecondOfTheMinute(N));
  Writeln('MilliSecond of the Minute : ',
          MilliSecondOfTheMinute(N));
End.
```

44.4.118 SecondOfTheMonth

Synopsis: Calculate number of seconds elapsed since the start of the month.

Declaration: `function SecondOfTheMonth(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `SecondOfTheMonth` returns the number of seconds that have passed since the start of the month (00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:00.999 on the first day of the month will return 0.

For an example, see the WeekOfTheMonth (192) function.

See also: WeekOfTheMonth (192), DayOfTheMonth (123), HourOfTheMonth (139), MinuteOfTheMonth (159), MilliSecondOfTheMonth (155)

44.4.119 SecondOfTheWeek

Synopsis: Calculate the number of seconds elapsed since the start of the week

Declaration: `function SecondOfTheWeek(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `SecondOfTheWeek` returns the number of seconds that have passed since the start of the week (00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:00.999 on the first day of the week will return 0.

For an example, see the DayOfTheWeek (124) function.

See also: SecondOfTheYear (177), SecondOfTheMonth (177), SecondOfTheDay (176), SecondOfTheHour (176), SecondOfTheMinute (176), DayOfTheWeek (124), HourOfTheWeek (139), MinuteOfTheWeek (160), MilliSecondOfTheWeek (156)

44.4.120 SecondOfTheYear

Synopsis: Calculate the number of seconds elapsed since the start of the year.

Declaration: `function SecondOfTheYear(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `SecondOfTheYear` returns the number of seconds that have passed since the start of the year (January 1, 00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. January 1 00:00:00.999 will return 0.

For an example, see the WeekOfTheYear (192) function.

See also: WeekOfTheYear (192), DayOfTheYear (124), HourOfTheYear (140), MinuteOfTheYear (160), SecondOfTheYear (177), MilliSecondOfTheYear (156)

44.4.121 SecondsBetween

Synopsis: Calculate the number of whole seconds between two DateTime values.

Declaration:
```
function SecondsBetween(const ANow: TDateTime;const AThen: TDateTime)
                                : Int64
```

Visibility: default

Description: `SecondsBetween` returns the number of whole seconds between `ANow` and `AThen`. This means the fractional part of a second (milliseconds etc.) is dropped.

See also: YearsBetween (204), MonthsBetween (163), WeeksBetween (193), DaysBetween (124), HoursBetween (140), MinutesBetween (160), MilliSecondsBetween (157)

Listing: ./datutex/ex61.pp

```
Program Example61;

{ This program demonstrates the SecondsBetween function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
  Write('Number of seconds between ');
  Write(TimeToStr(AThen),' and ',TimeToStr(ANow));
  Writeln(' : ',SecondsBetween(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=D1-(999*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(1001*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(2001*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(5001*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(5.4*OneSecond);
  Test(D1,D2);
  D2:=D1-(2.5*OneSecond);
  Test(D1,D2);
End.
```

44.4.122 SecondSpan

Synopsis: Calculate the approximate number of seconds between two DateTime values.

Declaration:
```
function SecondSpan(const ANow: TDateTime;const AThen: TDateTime)
                         : Double
```

Visibility: default

Description: `SecondSpan` returns the number of seconds between `ANow` and `AThen`, including any fractional parts of a second.

See also: YearSpan (205), MonthSpan (164), WeekSpan (195), DaySpan (127), HourSpan (141), MinuteSpan (161), MilliSecondSpan (157), SecondsBetween (178)

Listing: ./datutex/ex69.pp

```
Program Example69;

{ This program demonstrates the SecondSpan function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
  Write('Number of seconds between ');
  Write(TimeToStr(AThen),' and ',TimeToStr(ANow));
  Writeln(' : ',SecondSpan(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=D1-(999*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(1001*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(2001*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(5001*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(5.4*OneSecond);
  Test(D1,D2);
  D2:=D1-(2.5*OneSecond);
  Test(D1,D2);
End.
```

44.4.123 StartOfADay

Synopsis: Return the start of a day as a DateTime value, given a day indication

Declaration:
```
function StartOfADay(const AYear: Word;const AMonth: Word;
                     const ADay: Word) : TDateTime;  Overload
function StartOfADay(const AYear: Word;const ADayOfYear: Word)
                      : TDateTime;  Overload
```

Visibility: default

Description: `StartOfADay` returns a `TDateTime` value with the date/time indication of the start (0:0:0.000) of the day given by `AYear`, `AMonth`, `ADay`.

The day may also be indicated with a `AYear`, `ADayOfYear` pair.

See also: StartOfTheDay (182), StartOfTheWeek (183), StartOfAWeek (181), StartOfAMonth (180), StartOfTheMonth (182), EndOfTheWeek (137), EndOfAWeek (135), EndOfTheYear (138), EndOfAYear (136), EndOfTheMonth (137), EndOfAMonth (134), EndOfTheDay (136), EndOfADay (134)

Listing: ./datutex/ex38.pp

```
Program Example38;

{ This program demonstrates the StartOfADay function }

Uses SysUtils, DateUtils;

Const
  Fmt = '"Start of the day : "dd mmmm yyyy hh:nn:ss';

Var
  Y,M,D : Word;

Begin
  Y:=YearOf(Today);
  M:=MonthOf(Today);
  D:=DayOf(Today);
  WriteLn(FormatDateTime(Fmt,StartOfADay(Y,M,D)));
  DecodeDateDay(Today,Y,D);
  WriteLn(FormatDateTime(Fmt,StartOfADay(Y,D)));
End.
```

44.4.124 StartOfAMonth

Synopsis: Return first date of month, given a year/month pair.

Declaration: `function StartOfAMonth(const AYear: Word;const AMonth: Word) : TDateTime`

Visibility: default

Description: `StartOfAMonth` e returns a `TDateTime` value with the date of the first day of the month indicated by the `AYear`, `AMonth` pair.

See also: StartOfTheMonth (182), EndOfTheMonth (137), EndOfAMonth (134), EndOfTheYear (138), EndOfAYear (136), StartOfAWeek (181), StartOfTheWeek (183)

Listing: ./datutex/ex30.pp

```
Program Example30;

{ This program demonstrates the StartOfAMonth function }

Uses SysUtils, DateUtils;

Const
  Fmt = '"First day of this month : "dd mmmm yyyy';
```

```
Var
  Y,M : Word;

Begin
  Y:=YearOf(Today);
  M:=MonthOf(Today);
  WriteLn(FormatDateTime(Fmt,StartOfAMonth(Y,M)));
End.
```

44.4.125 StartOfAWeek

Synopsis: Return a day of the week, given a year, week and day in the week.

Declaration:
```
function StartOfAWeek(const AYear: Word;const AWeekOfYear: Word;
                      const ADayOfWeek: Word) : TDateTime
function StartOfAWeek(const AYear: Word;const AWeekOfYear: Word)
                       : TDateTime
```

Visibility: default

Description: `StartOfAWeek` returns a `TDateTime` value with the date of the indicated day of the week indicated by the `AYear`, `AWeek`, `ADayOfWeek` values.

The default value for `ADayOfWeek` is 1.

See also: StartOfTheWeek (183), EndOfTheWeek (137), EndOfAWeek (135), StartOfAMonth (180), EndOfTheYear (138), EndOfAYear (136), EndOfTheMonth (137), EndOfAMonth (134)

Listing: ./datutex/ex34.pp

```
Program Example34;

{ This program demonstrates the StartOfAWeek function }

Uses SysUtils,DateUtils;

Const
  Fmt = '"First day of this week : "dd mmmm yyyy hh:nn:ss';
  Fmt2 = '"Second day of this week : "dd mmmm yyyy hh:nn:ss';

Var
  Y,W : Word;

Begin
  Y:=YearOf(Today);
  W:=WeekOf(Today);
  WriteLn(FormatDateTime(Fmt,StartOfAWeek(Y,W)));
  WriteLn(FormatDateTime(Fmt2,StartOfAWeek(Y,W,2)));
End.
```

44.4.126 StartOfAYear

Synopsis: Return the first day of a given year.

Declaration:
```
function StartOfAYear(const AYear: Word) : TDateTime
```

Visibility: default

Description: `StartOfAYear` returns a `TDateTime` value with the date of the first day of the year `AYear` (January 1).

See also: StartOfTheYear (184), EndOfTheYear (138), EndOfAYear (136), EndOfTheMonth (137), EndOfAMonth (134), StartOfAWeek (181), StartOfTheWeek (183)

Listing: ./datutex/ex26.pp

```
Program Example26;

{ This program demonstrates the StartOfAYear function }

Uses SysUtils, DateUtils;

Const
  Fmt = '"First day of this year : "dd mmmm yyyy';

Begin
  WriteLn(FormatDateTime(Fmt, StartOfAYear(YearOf(Today))));
End.
```

44.4.127 StartOfTheDay

Synopsis: Calculate the start of the day as a DateTime value, given a moment in the day.

Declaration: `function StartOfTheDay(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `StartOfTheDay` extracts the date part of `AValue` and returns a `TDateTime` value with the date/time indication of the start (0:0:0.000) of this day.

See also: StartOfADay (179), StartOfTheWeek (183), StartOfAWeek (181), StartOfAMonth (180), StartOfTheMonth (182), EndOfTheWeek (137), EndOfAWeek (135), EndOfTheYear (138), EndOfAYear (136), EndOfTheMonth (137), EndOfAMonth (134), EndOftheDay (136), EndOfADay (134)

Listing: ./datutex/ex36.pp

```
Program Example36;

{ This program demonstrates the StartOfTheDay function }

Uses SysUtils, DateUtils;

Const
  Fmt = '"Start of the day : "dd mmmm yyyy hh:nn:ss';


Begin
  WriteLn(FormatDateTime(Fmt, StartOfTheDay(Today)));
End.
```

44.4.128 StartOfTheMonth

Synopsis: Calculate the first day of the month, given a date in that month.

Declaration: `function StartOfTheMonth(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `StartOfTheMonth` extracts the year and month parts of `AValue` and returns a `TDateTime` value with the date of the first day of that year and month as the StartOfAMonth (180) function.

See also: StartOfAMonth (180), EndOfTheYear (138), EndOfAYear (136), EndOfTheMonth (137), EndOfAMonth (134), StartOfAWeek (181), StartOfTheWeek (183)

Listing: ./datutex/ex28.pp

```
Program Example28;

{ This program demonstrates the StartOfTheMonth function }

Uses SysUtils, DateUtils;

Const
  Fmt = '"First day of this month : "dd mmmm yyyy';

Begin
  WriteLn(FormatDateTime(Fmt, StartOfTheMonth(Today)));
End.
```

44.4.129 StartOfTheWeek

Synopsis: Return the first day of the week, given a date.

Declaration: `function StartOfTheWeek(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `StartOfTheWeek` extracts the year and week parts of `AValue` and returns a `TDateTime` value with the date of the first day of that week as the StartOfAWeek (181) function.

See also: StartOfAWeek (181), EndOfTheWeek (137), EndOfAWeek (135), StartOfAMonth (180), EndOfTheYear (138), EndOfAYear (136), EndOfTheMonth (137), EndOfAMonth (134)

Listing: ./datutex/ex32.pp

```
Program Example32;

{ This program demonstrates the StartOfTheWeek function }

Uses SysUtils, DateUtils;

Const
  Fmt = '"First day of this week : "dd mmmm yyyy';

Begin
  WriteLn(FormatDateTime(Fmt, StartOfTheWeek(Today)));
End.
```

44.4.130 StartOfTheYear

Synopsis: Return the first day of the year, given a date in this year.

Declaration: `function StartOfTheYear(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `StartOfTheYear` extracts the year part of `AValue` and returns a `TDateTime` value with the date of the first day of that year (January 1), as the StartOfAYear (181) function.

See also: StartOfAYear (181), EndOfTheYear (138), EndOfAYear (136)

Listing: ./datutex/ex24.pp

```
Program Example24;

{ This program demonstrates the StartOfTheYear function }

Uses SysUtils, DateUtils;

Const
  Fmt = '"First day of this year : "dd mmmm yyyy';

Begin
  WriteLn(FormatDateTime(Fmt, StartOfTheYear(Today)));
End.
```

44.4.131 TimeOf

Synopsis: Extract the time part from a DateTime indication.

Declaration: `function TimeOf(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `TimeOf` extracts the time part from `AValue` and returns the result.

Since the `TDateTime` is actually a double with the time part encoded in the fractional part, this operation corresponds to a call to `Frac`.

See also: DateOf (121), YearOf (203), MonthOf (162), DayOf (123), HourOf (138), MinuteOf (158), SecondOf (175), MilliSecondOf (154)

Listing: ./datutex/ex2.pp

```
Program Example2;

{ This program demonstrates the TimeOf function }

Uses SysUtils, DateUtils;

Begin
  WriteLn('Time is : ',TimeToStr(TimeOf(Now)));
End.
```

44.4.132 Today

Synopsis: Return the current date

Declaration: `function Today : TDateTime`

Visibility: default

Description: `Today` is an alias for the Date (628) function in the sysutils (628) unit.

For an example, see Yesterday (206)

See also: Date (628), Yesterday (206), Tomorrow (185)

44.4.133 Tomorrow

Synopsis: Return the next day

Declaration: `function Tomorrow : TDateTime`

Visibility: default

Description: `Tomorrow` returns tomorrow's date. Tomorrow is determined from the system clock, i.e. it is Today (185) +1.

See also: Today (185), Yesterday (206)

Listing: ./datutex/ex19.pp

```
Program Example19;

{ This program demonstrates the Tomorrow function }

Uses SysUtils, DateUtils;

Begin
  WriteLn(FormatDateTime('"Today is" dd mmmm yyyy',Today));
  WriteLn(FormatDateTime('"Tomorrow will be" dd mmmm yyyy',Tomorrow));
End.
```

44.4.134 TryEncodeDateDay

Synopsis: Encode a year and day of year to a `TDateTime` value

Declaration:
```
function TryEncodeDateDay(const AYear: Word;const ADayOfYear: Word;
                          out AValue: TDateTime) : Boolean
```

Visibility: default

Description: `TryEncodeDateDay` encodes the values `AYear` and `ADayOfYear` to a date value and returns this value in `AValue`.

If the encoding was succcesful, `True` is returned. `False` is returned if any of the arguments is not valid.

See also: EncodeDateDay (132), EncodeDateTime (132), EncodeDateMonthWeek (132), EncodeDateWeek (133), TryEncodeDateTime (187), TryEncodeDateMonthWeek (186), TryEncodeDateWeek (187)

Listing: ./datutex/ex84.pp

```
Program Example84;

{ This program demonstrates the TryEncodeDateDay function }

Uses SysUtils,DateUtils;

Var
  Y,DoY : Word;
  TS : TDateTime;

Begin
  DecodeDateDay(Now,Y,DoY);
  If TryEncodeDateDay(Y,DoY,TS) then
    Writeln('Today is : ',DateToStr(TS))
  else
    Writeln('Wrong year/day of year indication');
End.
```

44.4.135 TryEncodeDateMonthWeek

Synopsis: Encode a year, month, week of month and day of week to a `TDateTime` value

Declaration:
```
function TryEncodeDateMonthWeek(const AYear: Word;const AMonth: Word;
                                 const AWeekOfMonth: Word;
                                 const ADayOfWeek: Word;
                                 out AValue: TDateTime) : Boolean
```

Visibility: default

Description: `TryEncodeDateTime` encodes the values `AYearAMonth`, `WeekOfMonth`,`ADayOfWeek`, to a date value and returns this value in `AValue`.

If the encoding was succesful, `True` is returned, `False` if any of the arguments is not valid.

See also: DecodeDateMonthWeek (129), EncodeDateTime (132), EncodeDateWeek (133), EncodeDateDay (132), EncodeDateMonthWeek (132), TryEncodeDateTime (187), TryEncodeDateWeek (187), TryEncodeDateDay (185), NthDayOfWeek (164)

Listing: ./datutex/ex86.pp

```
Program Example86;

{ This program demonstrates the TryEncodeDateMonthWeek function }

Uses SysUtils,DateUtils;

Var
  Y,M,Wom,Dow : Word;
  TS : TDateTime;

Begin
  DecodeDateMonthWeek(Now,Y,M,WoM,DoW);
  If TryEncodeDateMonthWeek(Y,M,WoM,Dow,TS) then
    Writeln('Today is : ',DateToStr(TS))
  else
    Writeln('Invalid year/month/week/dow indication');
End.
```

44.4.136 TryEncodeDateTime

Synopsis: Encode a Year, Month, Day, Hour, minute, seconds, milliseconds tuplet to a `TDateTime` value

Declaration:
```
function TryEncodeDateTime(const AYear: Word;const AMonth: Word;
                           const ADay: Word;const AHour: Word;
                           const AMinute: Word;const ASecond: Word;
                           const AMilliSecond: Word;
                           out AValue: TDateTime) : Boolean
```

Visibility: default

Description: `EncodeDateTime` encodes the values `AYearAMonth`, `ADay`,`AHour`, `AMinute`,`ASecond` and `AMilliSecond` to a date/time valueand returns this value in `AValue`.

If the date was encoded succesfully, `True` is returned, `False` is returned if one of the arguments is not valid.

See also: EncodeDateTime (132), EncodeDateMonthWeek (132), EncodeDateWeek (133), EncodeDateDay (132), TryEncodeDateDay (185), TryEncodeDateWeek (187), TryEncodeDateMonthWeek (186)

Listing: ./datutex/ex80.pp

```
Program Example79;

{ This program demonstrates the TryEncodeDateTime function }

Uses SysUtils, DateUtils;

Var
  Y,Mo,D,H,Mi,S,MS : Word;
  TS : TDateTime;

Begin
  DecodeDateTime(Now,Y,Mo,D,H,Mi,S,MS);
  If TryEncodeDateTime(Y,Mo,D,H,Mi,S,MS,TS) then
    Writeln('Now is : ',DateTimeToStr(TS))
  else
    Writeln('Wrong date/time indication');
End.
```

44.4.137 TryEncodeDateWeek

Synopsis: Encode a year, week and day of week triplet to a `TDateTime` value

Declaration:
```
function TryEncodeDateWeek(const AYear: Word;const AWeekOfYear: Word;
                           out AValue: TDateTime;const ADayOfWeek: Word)
                            : Boolean
function TryEncodeDateWeek(const AYear: Word;const AWeekOfYear: Word;
                           out AValue: TDateTime) : Boolean
```

Visibility: default

Description: `TryEncodeDateWeek` encodes the values `AYear`, `AWeekOfYear` and `ADayOfWeek` to a date value and returns this value in `AValue`.

If the encoding was succcesful, `True` is returned. `False` is returned if any of the arguments is not valid.

See also: EncodeDateMonthWeek (132), EncodeDateWeek (133), EncodeDateTime (132), EncodeDateDay (132), TryEncodeDateTime (187), TryEncodeDateMonthWeek (186), TryEncodeDateDay (185)

Listing: ./datutex/ex82.pp

```
Program Example82;

{ This program demonstrates the TryEncodeDateWeek function }

Uses SysUtils, DateUtils;

Var
  Y,W,Dow : Word;
  TS : TDateTime;

Begin
  DecodeDateWeek(Now,Y,W,Dow);
  If TryEncodeDateWeek(Y,W,TS,Dow) then
    WriteIn('Today is : ',DateToStr(TS))
  else
    WriteIn('Invalid date/week indication');
End.
```

44.4.138 TryEncodeDayOfWeekInMonth

Synopsis: Encode a year, month, week, day of week triplet to a `TDateTime` value

Declaration:
```
function TryEncodeDayOfWeekInMonth(const AYear: Word;const AMonth: Word;
                                   const ANthDayOfWeek: Word;
                                   const ADayOfWeek: Word;
                                   out AValue: TDateTime) : Boolean
```

Visibility: default

Description: `EncodeDayOfWeekInMonth` encodes `AYear`, `AMonth`, `ADayOfweek` and `ANthDayOfweek` to a valid date stamp and returns the result in `AValue`.

`ANthDayOfweek`is the N-th time that this weekday occurs in the month, e.g. the third saturday of the month.

The function returns `True` if the encoding was succesful, `False` if any of the values is not in range.

See also: NthDayOfWeek (164), EncodeDateMonthWeek (132), #rtl.sysutils.DayOfWeek (628), DecodeDayOfWeekInMonth (131), EncodeDayOfWeekInMonth (133)

Listing: ./datutex/ex106.pp

```
Program Example105;

{ This program demonstrates the DecodeDayOfWeekInMonth function }

Uses SysUtils, DateUtils;

Var
  Y,M,NDoW,DoW : Word;
  D : TDateTime;
Begin
  DecodeDayOfWeekInMonth(Date,Y,M,NDoW,DoW);
```

```
  If TryEncodeDayOfWeekInMonth(Y,M,NDoW,DoW,D) then
    begin
    Write(DateToStr(D),' is the ',NDow,'-th ');
    WriteIn(formatdateTime('dddd',D),' of the month.');
    end
  else
    WriteIn('Invalid year/month/NthDayOfweek combination');
End.
```

44.4.139 TryEncodeTimeInterval

Synopsis: Try to encode an interval as a `TDateTime` value.

Declaration:
```
function TryEncodeTimeInterval(Hour: Word;Min: Word;Sec: Word;
                               MSec: Word;out Time: TDateTime) : Boolean
```

Visibility: default

Description: `TryEncodeTimeInterval` encodes a time interval expressed in `Hour`, `Min`, `Sec`, `MSec` as a `TDateTime` value and returns the value in `Time`. It returns `True` if `Min`, `Sec`, `MSec` contain valid time values (i.e. less than 60, 60 resp. MSec). The number of hours may be larger than 24.

See also: EncodeTimeInterval (133)

44.4.140 TryJulianDateToDateTime

Synopsis: Convert a Julian date representation to a `TDateTime` value.

Declaration:
```
function TryJulianDateToDateTime(const AValue: Double;
                                 out ADateTime: TDateTime) : Boolean
```

Visibility: default

Description: Try to convert a Julian date to a regular `TDateTime` date/time representation.

See also: DateTimeToJulianDate (122), JulianDateToDateTime (153), DateTimeToModifiedJulianDate (123), TryModifiedJulianDateToDateTime (189)

44.4.141 TryModifiedJulianDateToDateTime

Synopsis: Convert a modified Julian date representation to a `TDateTime` value.

Declaration:
```
function TryModifiedJulianDateToDateTime(const AValue: Double;
                                         out ADateTime: TDateTime)
                                          : Boolean
```

Visibility: default

Description: Not yet implemented.

Errors: Currently, trying to use this function will raise an exception.

See also: DateTimeToJulianDate (122), JulianDateToDateTime (153), TryJulianDateToDateTime (189), DateTimeToModifiedJulianDate (123), ModifiedJulianDateToDateTime (162)

44.4.142 TryRecodeDateTime

Synopsis: Replace selected parts of a `TDateTime` value with other values

Declaration:
```
function TryRecodeDateTime(const AValue: TDateTime;const AYear: Word;
                           const AMonth: Word;const ADay: Word;
                           const AHour: Word;const AMinute: Word;
                           const ASecond: Word;const AMilliSecond: Word;
                           out AResult: TDateTime) : Boolean
```

Visibility: default

Description: `TryRecodeDateTime` replaces selected parts of the timestamp `AValue` with the date/time values specified in `AYear`, `AMonth`, `ADay`, `AHour`, `AMinute`, `ASecond` and `AMilliSecond`. If any of these values equals the pre-defined constant RecodeLeaveFieldAsIs (118), then the corresponding part of the date/time stamp is left untouched.

The resulting date/time is returned in `AValue`.

The function returns `True` if the encoding was succesful. It returns `False` if one of the values `AYear`, `AMonth`, `ADay`, `AHour`, `AMinute`, `ASecondAMilliSecond` is not within a valid range.

See also: RecodeYear (172), RecodeMonth (170), RecodeDay (167), RecodeHour (168), RecodeMinute (169), RecodeSecond (171), RecodeMilliSecond (169), RecodeDate (166), RecodeTime (171), RecodeDateTime (166)

Listing: ./datutex/ex97.pp

```
Program Example97;

{ This program demonstrates the TryRecodeDateTime function }

Uses SysUtils, DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var
  S : AnsiString;
  D : TDateTime ;

Begin
  If TryRecodeDateTime(Now,2000,2,RecodeLeaveFieldAsIs,0,0,0,0,D) then
    begin
    S:=FormatDateTime(Fmt,D);
    WriteIn('This moment in februari 2000 : ',S);
    end
  else
    WriteIn('This moment did/does not exist in februari 2000');
End.
```

44.4.143 UniversalTimeToLocal

Synopsis: Convert UTC time to local time

Declaration:
```
function UniversalTimeToLocal(UT: TDateTime) : TDateTime
function UniversalTimeToLocal(UT: TDateTime;TZOffset: Integer)
                              : TDateTime
```

Visibility: default

Description: `UniversalTimeToLocal` converts a universal time indication to a local time: it applies the `TZOffset` timezone offset to the `UT` Universal time (UTC). If no `TZOffset` is specified, the local time offset as returned by GetLocalTimeOffset (116) is used.

See also: GetLocalTimeOffset (116), LocalTimeToUniversal (153)

44.4.144 UnixTimeStampToMac

Synopsis: Convert Unix Timestamp to a Mac Timestamp

Declaration: `function UnixTimeStampToMac(const AValue: Int64) : Int64`

Visibility: default

Description: `UnixTimeStampToMac` converts the unix epoch time in `AValue` to a valid Mac timestamp indication and returns the result.

Errors: None.

See also: DateTimeToMac (122), MacToDateTime (154), MacTimeStampToUnix (154)

44.4.145 UnixToDateTime

Synopsis: Convert Unix epoch time to a `TDateTime` value

Declaration: `function UnixToDateTime(const AValue: Int64) : TDateTime`

Visibility: default

Description: `UnixToDateTime` converts epoch time (seconds elapsed since 1/1/1970) to a `TDateTime` value.

See also: DateTimeToUnix (123)

44.4.146 WeekOf

Synopsis: Extract week (of the year) from a given date.

Declaration: `function WeekOf(const AValue: TDateTime) : Word`

Visibility: default

Description: `WeekOf` returns the week-of-the-year part of the `AValue` date/time indication. It is a number between 1 and 53.

For an example, see YearOf (203)

See also: YearOf (203), DayOf (123), MonthOf (162), HourOf (138), MinuteOf (158), SecondOf (175), MilliSecondOf (154)

44.4.147 WeekOfTheMonth

Synopsis: Extract the week of the month (and optionally month and year) from a DateTime value

Declaration:
```
function WeekOfTheMonth(const AValue: TDateTime) : Word;  Overload
function WeekOfTheMonth(const AValue: TDateTime;out AYear: Word;
                        out AMonth: Word) : Word;  Overload
```

Visibility: default

Description: `WeekOfTheMonth` extracts the week of the month from `Avalue` and returns it, and optionally returns the year and month as well (in `AYear`, `AMonth` respectively).

Remark: Note that weeks are numbered from 1 using the ISO 8601 standard, and the day of the week as well. This means that the year and month may not be the same as the year part of the date, since the week may start in the previous year as the first week of the year is the week with at least 4 days in it.

See also: WeekOfTheYear (192), DayOfTheMonth (123), HourOfTheMonth (139), MinuteOfTheMonth (159), SecondOfTheMonth (177), MilliSecondOfTheMonth (155)

Listing: ./datutex/ex41.pp

```
Program Example41;

{ This program demonstrates the WeekOfTheMonth function }

Uses SysUtils, DateUtils;

Var
  N : TDateTime;

Begin
  N:=Now;
  Writeln('Week of the Month        : ',WeekOfTheMonth(N));
  Writeln('Day of the Month         : ',DayOfTheMonth(N));
  Writeln('Hour of the Month        : ',HourOfTheMonth(N));
  Writeln('Minute of the Month      : ',MinuteOfTheMonth(N));
  Writeln('Second of the Month      : ',SecondOfTheMonth(N));
  Writeln('MilliSecond of the Month : ',
          MilliSecondOfTheMonth(N));
End.
```

44.4.148 WeekOfTheYear

Synopsis: Extract the week of the year (and optionally year) of a DateTime indication.

Declaration:
```
function WeekOfTheYear(const AValue: TDateTime) : Word;  Overload
function WeekOfTheYear(const AValue: TDateTime;out AYear: Word) : Word
                       ;  Overload
```

Visibility: default

Description: `WeekOfTheYear` extracts the week of the year from `Avalue` and returns it, and optionally returns the year as well. It returns the same value as WeekOf (191).

Remark: Note that weeks are numbered from 1 using the ISO 8601 standard, and the day of the week as well. This means that the year may not be the same as the year part of the date, since the week may start in the previous year as the first week of the year is the week with at least 4 days in it.

See also: WeekOf (191), MonthOfTheYear (162), DayOfTheYear (124), HourOfTheYear (140), MinuteOfTheYear (160), SecondOfTheYear (177), MilliSecondOfTheYear (156)

Listing: ./datutex/ex40.pp

```
Program Example40;

{ This program demonstrates the WeekOfTheYear function }

Uses SysUtils, DateUtils;

Var
  N : TDateTime;

Begin
  N:=Now;
  Writeln('Month of the year       : ',MonthOfTheYear(N));
  Writeln('Week of the year        : ',WeekOfTheYear(N));
  Writeln('Day of the year         : ',DayOfTheYear(N));
  Writeln('Hour of the year        : ',HourOfTheYear(N));
  Writeln('Minute of the year      : ',MinuteOfTheYear(N));
  Writeln('Second of the year      : ',SecondOfTheYear(N));
  Writeln('MilliSecond of the year : ',
          MilliSecondOfTheYear(N));
End.
```

44.4.149 WeeksBetween

Synopsis: Calculate the number of whole weeks between two DateTime values

Declaration:
```
function WeeksBetween(const ANow: TDateTime;const AThen: TDateTime)
                            : Integer
```

Visibility: default

Description: `WeeksBetween` returns the number of whole weeks between `ANow` and `AThen`. This means the fractional part of a Week is dropped.

See also: YearsBetween (204), MonthsBetween (163), DaysBetween (124), HoursBetween (140), MinutesBetween (160), SecondsBetween (178), MilliSecondsBetween (157)

Listing: ./datutex/ex57.pp

```
Program Example57;

{ This program demonstrates the WeeksBetween function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
  Write('Number of weeks between ');
  Write(DateToStr(AThen),' and ',DateToStr(ANow));
  Writeln(' : ',WeeksBetween(ANow,AThen));
end;
```

```
Var
  D1,D2 : TDateTime;

Begin
  D1:=Today;
  D2:=Today-7;
  Test(D1,D2);
  D2:=Today-8;
  Test(D1,D2);
  D2:=Today-14;
  Test(D1,D2);
  D2:=Today-35;
  Test(D1,D2);
  D2:=Today-36;
  Test(D1,D2);
  D2:=Today-17;
  Test(D1,D2);
End.
```

44.4.150 WeeksInAYear

Synopsis: Return the number of weeks in a given year

Declaration: `function WeeksInAYear(const AYear: Word) : Word`

Visibility: default

Description: `WeeksInAYear` returns the number of weeks in the year `AYear`. The return value is either 52 or 53.

Remark: The first week of the year is determined according to the ISO 8601 standard: It is the first week that has at least 4 days in it, i.e. it includes a thursday.

See also: WeeksInYear (194), DaysInYear (127), DaysInAYear (126), DaysInMonth (126), DaysInAMonth (125)

Listing: ./datutex/ex13.pp

```
Program Example13;

{ This program demonstrates the WeeksInAYear function }

Uses SysUtils,DateUtils;

Var
  Y : Word;

Begin
  For Y:=1992 to 2010 do
    Writeln(Y,' has ',WeeksInAYear(Y),' weeks.');
End.
```

44.4.151 WeeksInYear

Synopsis: return the number of weeks in the year, given a date

Declaration: `function WeeksInYear(const AValue: TDateTime) : Word`

Visibility: default

Description: `WeeksInYear` returns the number of weeks in the year part of `AValue`. The return value is either 52 or 53.

Remark: The first week of the year is determined according to the ISO 8601 standard: It is the first week that has at least 4 days in it, i.e. it includes a thursday.

See also: WeeksInAYear (194), DaysInYear (127), DaysInAYear (126), DaysInMonth (126), DaysInAMonth (125)

Listing: ./datutex/ex12.pp

```
Program Example12;

{ This program demonstrates the WeeksInYear function }

Uses SysUtils, DateUtils;

Var
  Y : Word;

Begin
  For Y:=1992 to 2010 do
    WriteIn(Y,' has ',WeeksInYear(EncodeDate(Y,2,1)),' weeks.');
End.
```

44.4.152 WeekSpan

Synopsis: Calculate the approximate number of weeks between two DateTime values.

Declaration: `function WeekSpan(const ANow: TDateTime;const AThen: TDateTime) : Double`

Visibility: default

Description: `WeekSpan` returns the number of weeks between `ANow` and `AThen`, including any fractional parts of a week.

See also: YearSpan (205), MonthSpan (164), DaySpan (127), HourSpan (141), MinuteSpan (161), SecondSpan (179), MilliSecondSpan (157), WeeksBetween (193)

Listing: ./datutex/ex65.pp

```
Program Example57;

{ This program demonstrates the WeekSpan function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
 Write('Number of weeks between ');
 Write(DateToStr(AThen),' and ',DateToStr(ANow));
 WriteIn(' : ',WeekSpan(ANow,AThen));
end;

Var
```

```
  D1,D2 : TDateTime;

Begin
  D1:=Today;
  D2:=Today-7;
  Test(D1,D2);
  D2:=Today-8;
  Test(D1,D2);
  D2:=Today-14;
  Test(D1,D2);
  D2:=Today-35;
  Test(D1,D2);
  D2:=Today-36;
  Test(D1,D2);
  D2:=Today-17;
  Test(D1,D2);
End.
```

44.4.153 WithinPastDays

Synopsis: Check whether two datetimes are only a number of days apart

Declaration:
```
function WithinPastDays(const ANow: TDateTime;const AThen: TDateTime;
                        const ADays: Integer) : Boolean
```

Visibility: default

Description: `WithinPastDays` compares the timestamps `ANow` and `AThen` and returns `True` if the difference between them is at most `ADays` days apart, or `False` if they are further apart.

Remark: Since this function uses the DaysBetween (124) function to calculate the difference in days, this means that fractional days do not count, and the fractional part is simply dropped, so for two dates actually 2 and a half days apart, the result will also be `True`

See also: WithinPastYears (203), WithinPastMonths (200), WithinPastWeeks (202), WithinPastHours (197), WithinPastMinutes (199), WithinPastSeconds (201), WithinPastMilliSeconds (198)

Listing: ./datutex/ex50.pp

```
Program Example50;

{ This program demonstrates the WithinPastDays function }

Uses SysUtils,DateUtils;

Procedure Test(ANow,AThen : TDateTime; ADays : Integer);

begin
 Write(DateTimeToStr(AThen),' and ',DateTimeToStr(ANow));
 Write(' are within ',ADays,' days: ');
 WriteIn(WithinPastDays(ANow,AThen,ADays));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Now;
```

```
  D2:=Today-23/24;
  Test(D1,D2,1);
  D2:=Today-1;
  Test(D1,D2,1);
  D2:=Today-25/24;
  Test(D1,D2,1);
  D2:=Today-26/24;
  Test(D1,D2,5);
  D2:=Today-5.4;
  Test(D1,D2,5);
  D2:=Today-2.5;
  Test(D1,D2,1);
  Test(D1,D2,2);
  Test(D1,D2,3);
End.
```

44.4.154 WithinPastHours

Synopsis: Check whether two datetimes are only a number of hours apart

Declaration:
```
function WithinPastHours(const ANow: TDateTime;const AThen: TDateTime;
                         const AHours: Int64) : Boolean
```

Visibility: default

Description: `WithinPastHours` compares the timestamps `ANow` and `AThen` and returns `True` if the difference between them is at most `AHours` hours apart, or `False` if they are further apart.

Remark: Since this function uses the HoursBetween (140) function to calculate the difference in Hours, this means that fractional hours do not count, and the fractional part is simply dropped, so for two dates actually 2 and a half hours apart, the result will also be `True`

See also: WithinPastYears (203), WithinPastMonths (200), WithinPastWeeks (202), WithinPastDays (196), WithinPastMinutes (199), WithinPastSeconds (201), WithinPastMilliSeconds (198)

Listing: ./datutex/ex51.pp

```
Program Example51;

{ This program demonstrates the WithinPastHours function }

Uses SysUtils,DateUtils;

Procedure Test(ANow,AThen : TDateTime; AHours : Integer);

begin
  Write(DateTimeToStr(AThen),' and ',DateTimeToStr(ANow));
  Write(' are within ',AHours,' hours: ');
  WriteIn(WithinPastHours(ANow,AThen,AHours));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=D1-(59*OneMinute);
  Test(D1,D2,1);
```

```
  D2:=D1-(61*OneMinute);
  Test(D1,D2,1);
  D2:=D1-(122*OneMinute);
  Test(D1,D2,1);
  D2:=D1-(306*OneMinute);
  Test(D1,D2,5);
  D2:=D1-(5.4*OneHour);
  Test(D1,D2,5);
  D2:=D1-(2.5*OneHour);
  Test(D1,D2,1);
  Test(D1,D2,2);
  Test(D1,D2,3);
End.
```

44.4.155 WithinPastMilliSeconds

Synopsis: Check whether two datetimes are only a number of milliseconds apart

Declaration:
```
function WithinPastMilliSeconds(const ANow: TDateTime;
                                const AThen: TDateTime;
                                const AMilliSeconds: Int64) : Boolean
```

Visibility: default

Description: `WithinPastMilliSeconds` compares the timestamps `ANow` and `AThen` and returns `True` if the difference between them is at most `AMilliSeconds` milliseconds apart, or `False` if they are further apart.

Remark: Since this function uses the MilliSecondsBetween (157) function to calculate the difference in milliseconds, this means that fractional milliseconds do not count, and the fractional part is simply dropped, so for two dates actually 2 and a half milliseconds apart, the result will also be `True`

See also: WithinPastYears (203), WithinPastMonths (200), WithinPastWeeks (202), WithinPastDays (196), WithinPastHours (197), WithinPastMinutes (199), WithinPastSeconds (201)

Listing: ./datutex/ex54.pp

```
Program Example54;

{ This program demonstrates the WithinPastMilliSeconds function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime; AMilliSeconds : Integer);

begin
  Write(TimeToStr(AThen),' and ',TimeToStr(ANow));
  Write(' are within ',AMilliSeconds,' milliseconds: ');
  WriteIn(WithinPastMilliSeconds(ANow,AThen,AMilliSeconds));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=D1-(0.9*OneMilliSecond);
  Test(D1,D2,1);
```

```
  D2:=D1-(1.0*OneMilliSecond);
  Test(D1,D2,1);
  D2:=D1-(1.1*OneMilliSecond);
  Test(D1,D2,1);
  D2:=D1-(2.5*OneMilliSecond);
  Test(D1,D2,1);
  Test(D1,D2,2);
  Test(D1,D2,3);
End.
```

44.4.156 WithinPastMinutes

Synopsis: Check whether two datetimes are only a number of minutes apart

Declaration:
```
function WithinPastMinutes(const ANow: TDateTime;const AThen: TDateTime;
                           const AMinutes: Int64) : Boolean
```

Visibility: default

Description: `WithinPastMinutes` compares the timestamps `ANow` and `AThen` and returns `True` if the difference between them is at most `AMinutes` minutes apart, or `False` if they are further apart.

Remark: Since this function uses the MinutesBetween (160) function to calculate the difference in Minutes, this means that fractional minutes do not count, and the fractional part is simply dropped, so for two dates actually 2 and a half minutes apart, the result will also be `True`

See also: WithinPastYears (203), WithinPastMonths (200), WithinPastWeeks (202), WithinPastDays (196), WithinPastHours (197), WithinPastSeconds (201), WithinPastMilliSeconds (198)

Listing: ./datutex/ex52.pp

```
Program Example52;

{ This program demonstrates the WithinPastMinutes function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime; AMinutes : Integer);

begin
 Write(DateTimeToStr(AThen),' and ',DateTimeToStr(ANow));
 Write(' are within ',AMinutes,' Minutes: ');
 WriteIn(WithinPastMinutes(ANow,AThen,AMinutes));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=D1-(59*OneSecond);
  Test(D1,D2,1);
  D2:=D1-(61*OneSecond);
  Test(D1,D2,1);
  D2:=D1-(122*OneSecond);
  Test(D1,D2,1);
  D2:=D1-(306*OneSecond);
  Test(D1,D2,5);
```

```
  D2:=D1-(5.4*OneMinute);
  Test(D1,D2,5);
  D2:=D1-(2.5*OneMinute);
  Test(D1,D2,1);
  Test(D1,D2,2);
  Test(D1,D2,3);
End.
```

44.4.157 WithinPastMonths

Synopsis: Check whether two datetimes are only a number of months apart

Declaration:
```
function WithinPastMonths(const ANow: TDateTime;const AThen: TDateTime;
                          const AMonths: Integer) : Boolean
```

Visibility: default

Description: `WithinPastMonths` compares the timestamps `ANow` and `AThen` and returns `True` if the difference between them is at most `AMonths` months apart, or `False` if they are further apart.

Remark: Since this function uses the MonthsBetween (163) function to calculate the difference in Months, this means that fractional months do not count, and the fractional part is simply dropped, so for two dates actually 2 and a half months apart, the result will also be `True`

See also: WithinPastYears (203), WithinPastWeeks (202), WithinPastDays (196), WithinPastHours (197), WithinPastMinutes (199), WithinPastSeconds (201), WithinPastMilliSeconds (198)

Listing: ./datutex/ex48.pp

```
Program Example48;

{ This program demonstrates the WithinPastMonths function }

Uses SysUtils,DateUtils;

Procedure Test(ANow,AThen : TDateTime; AMonths : Integer);

begin
 Write(DateToStr(AThen),' and ',DateToStr(ANow));
 Write(' are within ',AMonths,' months: ');
 WriteIn(WithinPastMonths(ANow,AThen,AMonths));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Today;
  D2:=Today-364;
  Test(D1,D2,12);
  D2:=Today-365;
  Test(D1,D2,12);
  D2:=Today-366;
  Test(D1,D2,12);
  D2:=Today-390;
  Test(D1,D2,12);
  D2:=Today-368;
  Test(D1,D2,11);
```

```
  D2:=Today-1000;
  Test(D1,D2,31);
  Test(D1,D2,32);
  Test(D1,D2,33);
End.
```

44.4.158 WithinPastSeconds

Synopsis: Check whether two datetimes are only a number of seconds apart

Declaration:
```
function WithinPastSeconds(const ANow: TDateTime;const AThen: TDateTime;
                           const ASeconds: Int64) : Boolean
```

Visibility: default

Description: `WithinPastSeconds` compares the timestamps `ANow` and `AThen` and returns `True` if the difference between them is at most `ASeconds` seconds apart, or `False` if they are further apart.

Remark: Since this function uses the SecondsBetween (178) function to calculate the difference in seconds, this means that fractional seconds do not count, and the fractional part is simply dropped, so for two dates actually 2 and a half seconds apart, the result will also be `True`

See also: WithinPastYears (203), WithinPastMonths (200), WithinPastWeeks (202), WithinPastDays (196), WithinPastHours (197), WithinPastMinutes (199), WithinPastMilliSeconds (198)

Listing: ./datutex/ex53.pp

```
Program Example53;

{ This program demonstrates the WithinPastSeconds function }

Uses SysUtils,DateUtils;

Procedure Test(ANow,AThen : TDateTime; ASeconds : Integer);

begin
  Write(DateTimeToStr(AThen),' and ',DateTimeToStr(ANow));
  Write(' are within ',ASeconds,' seconds: ');
  WriteIn(WithinPastSeconds(ANow,AThen,ASeconds));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=D1-(999*OneMilliSecond);
  Test(D1,D2,1);
  D2:=D1-(1001*OneMilliSecond);
  Test(D1,D2,1);
  D2:=D1-(2001*OneMilliSecond);
  Test(D1,D2,1);
  D2:=D1-(5001*OneMilliSecond);
  Test(D1,D2,5);
  D2:=D1-(5.4*OneSecond);
  Test(D1,D2,5);
  D2:=D1-(2.5*OneSecond);
  Test(D1,D2,1);
```

```
  Test(D1,D2,2);
  Test(D1,D2,3);
End.
```

44.4.159 WithinPastWeeks

Synopsis: Check whether two datetimes are only a number of weeks apart

Declaration: `function WithinPastWeeks(const ANow: TDateTime;const AThen: TDateTime; const AWeeks: Integer) : Boolean`

Visibility: default

Description: `WithinPastWeeks` compares the timestamps `ANow` and `AThen` and returns `True` if the difference between them is at most `AWeeks` weeks apart, or `False` if they are further apart.

Remark: Since this function uses the WeeksBetween (193) function to calculate the difference in Weeks, this means that fractional Weeks do not count, and the fractional part is simply dropped, so for two dates actually 2 and a half weeks apart, the result will also be `True`

See also: WithinPastYears (203), WithinPastMonths (200), WithinPastDays (196), WithinPastHours (197), WithinPastMinutes (199), WithinPastSeconds (201), WithinPastMilliSeconds (198)

Listing: ./datutex/ex49.pp

```
Program Example49;

{ This program demonstrates the WithinPastWeeks function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime; AWeeks : Integer);

begin
  Write(DateToStr(AThen),' and ',DateToStr(ANow));
  Write(' are within ',AWeeks,' weeks: ');
  WriteIn(WithinPastWeeks(ANow,AThen,AWeeks));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Today;
  D2:=Today-7;
  Test(D1,D2,1);
  D2:=Today-8;
  Test(D1,D2,1);
  D2:=Today-14;
  Test(D1,D2,1);
  D2:=Today-35;
  Test(D1,D2,5);
  D2:=Today-36;
  Test(D1,D2,5);
  D2:=Today-17;
  Test(D1,D2,1);
  Test(D1,D2,2);
  Test(D1,D2,3);
```

```
End.
```

44.4.160 WithinPastYears

Synopsis: Check whether two datetimes are only a number of years apart

Declaration:
```
function WithinPastYears(const ANow: TDateTime;const AThen: TDateTime;
                         const AYears: Integer) : Boolean
```

Visibility: default

Description: `WithinPastYears` compares the timestamps `ANow` and `AThen` and returns `True` if the difference between them is at most `AYears` years apart, or `False` if they are further apart.

Remark: Since this function uses the YearsBetween (204) function to calculate the difference in years, this means that fractional years do not count, and the fractional part is simply dropped, so for two dates actually 2 and a half years apart, the result will also be `True`

See also: WithinPastMonths (200), WithinPastWeeks (202), WithinPastDays (196), WithinPastHours (197), WithinPastMinutes (199), WithinPastSeconds (201), WithinPastMilliSeconds (198)

Listing: ./datutex/ex47.pp

```
Program Example47;

{ This program demonstrates the WithinPastYears function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime; AYears : Integer);

begin
 Write(DateToStr(AThen),' and ',DateToStr(ANow));
 Write(' are within ',AYears,' years: ');
 WriteIn(WithinPastYears(ANow,AThen,AYears));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Today;
  D2:=Today-364;
  Test(D1,D2,1);
  D2:=Today-365;
  Test(D1,D2,1);
  D2:=Today-366;
  Test(D1,D2,1);
  D2:=Today-390;
  Test(D1,D2,1);
  D2:=Today-368;
  Test(D1,D2,1);
  D2:=Today-1000;
  Test(D1,D2,1);
  Test(D1,D2,2);
  Test(D1,D2,3);
End.
```

44.4.161 YearOf

Synopsis: Extract the year from a given date.

Declaration: `function YearOf(const AValue: TDateTime) : Word`

Visibility: default

Description: `YearOf` returns the year part of the `AValue` date/time indication. It is a number between 1 and 9999.

See also: MonthOf (162), DayOf (123), WeekOf (191), HourOf (138), MinuteOf (158), SecondOf (175), MilliSecondOf (154)

Listing: ./datutex/ex23.pp

```
Program Example23;

{ This program demonstrates the YearOf function }

Uses SysUtils, DateUtils;

Var
  D : TDateTime;

Begin
  D:=Now;
  Writeln('Year        : ',YearOf(D));
  Writeln('Month       : ',MonthOf(D));
  Writeln('Day         : ',DayOf(D));
  Writeln('Week        : ',WeekOf(D));
  Writeln('Hour        : ',HourOf(D));
  Writeln('Minute      : ',MinuteOf(D));
  Writeln('Second      : ',SecondOf(D));
  Writeln('MilliSecond : ',MilliSecondOf(D));
End.
```

44.4.162 YearsBetween

Synopsis: Calculate the number of whole years between two DateTime values

Declaration: `function YearsBetween(const ANow: TDateTime;const AThen: TDateTime) : Integer`

Visibility: default

Description: `YearsBetween` returns the number of whole years between `ANow` and `AThen`. This number is an approximation, based on an average number of days of 365.25 per year (average over 4 years). This means the fractional part of a year is dropped.

See also: MonthsBetween (163), WeeksBetween (193), DaysBetween (124), HoursBetween (140), MinutesBetween (160), SecondsBetween (178), MilliSecondsBetween (157), YearSpan (205)

Listing: ./datutex/ex55.pp

```
Program Example55;

{ This program demonstrates the YearsBetween function }
```

```
Uses SysUtils,DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
  Write('Number of years between ');
  Write(DateToStr(AThen),' and ',DateToStr(ANow));
  Writeln(' : ',YearsBetween(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Today;
  D2:=Today-364;
  Test(D1,D2);
  D2:=Today-365;
  Test(D1,D2);
  D2:=Today-366;
  Test(D1,D2);
  D2:=Today-390;
  Test(D1,D2);
  D2:=Today-368;
  Test(D1,D2);
  D2:=Today-1000;
  Test(D1,D2);
End.
```

44.4.163 YearSpan

Synopsis: Calculate the approximate number of years between two DateTime values.

Declaration: `function YearSpan(const ANow: TDateTime;const AThen: TDateTime) : Double`

Visibility: default

Description: `YearSpan` returns the number of years between `ANow` and `AThen`, including any fractional parts of a year. This number is an approximation, based on an average number of days of 365.25 per year (average over 4 years).

See also: MonthSpan (164), WeekSpan (195), DaySpan (127), HourSpan (141), MinuteSpan (161), SecondSpan (179), MilliSecondSpan (157), YearsBetween (204)

Listing: ./datutex/ex63.pp

```
Program Example63;

{ This program demonstrates the YearSpan function }

Uses SysUtils,DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
  Write('Number of years between ');
```

```
  Write(DateToStr(AThen),' and ',DateToStr(ANow));
  Writeln(' : ',YearSpan(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Today;
  D2:=Today-364;
  Test(D1,D2);
  D2:=Today-365;
  Test(D1,D2);
  D2:=Today-366;
  Test(D1,D2);
  D2:=Today-390;
  Test(D1,D2);
  D2:=Today-368;
  Test(D1,D2);
  D2:=Today-1000;
  Test(D1,D2);
End.
```

44.4.164 Yesterday

Synopsis: Return the previous day.

Declaration: `function Yesterday : TDateTime`

Visibility: default

Description: `Yesterday` returns yesterday's date. Yesterday is determined from the system clock, i.e. it is Today (185) -1.

See also: Today (185), Tomorrow (185)

Listing: ./datutex/ex18.pp

```
Program Example18;

{ This program demonstrates the Yesterday function }

Uses SysUtils,DateUtils;

Begin
  Writeln(FormatDateTime('"Today is " dd mmmm yyyy',Today));
  Writeln(FormatDateTime('"Yesterday was " dd mmmm yyyy',Yesterday));
End.
```

Chapter 45

Reference for unit 'Dos'

45.1 Used units

Table 45.1: Used units by unit 'Dos'

Name	Page
BaseUnix	51
System	622

45.2 Overview

The DOS unit gives access to some operating system calls related to files, the file system, date and time. Except for the PalmOS target, this unit is available to all supported platforms.

The unit was first written for dos by Florian Klaempfl. It was ported to linux by Mark May and enhanced by Michael Van Canneyt. The Amiga version was ported by Nils Sjoholm.

Under non-DOS systems, some of the functionality is lost, as it is either impossible or meaningless to implement it. Other than that, the functionality is the same for all operating systems.

Because the DOS unit is a Turbo Pascal compatibility unit, it is no longer actively developed: the interface is frozen and it is maintained only for the purpose of porting Turbo Pascal programs. For new development, it is recommended to use the sysutils (628) unit instead.

45.3 System information

Functions for retrieving and setting general system information such as date and time.

Table 45.2:

Name	Description
DosVersion (1)	Get OS version
GetCBreak (207)	Get setting of control-break handling flag
GetDate (207)	Get system date
GetIntVec (207)	Get interrupt vector status
GetTime (207)	Get system time
GetVerify (207)	Get verify flag
Intr (207)	Execute an interrupt
Keep (207)	Keep process in memory and exit
MSDos (207)	Execute MS-dos function call
PackTime (207)	Pack time for file time
SetCBreak (207)	Set control-break handling flag
SetDate (207)	Set system date
SetIntVec (207)	Set interrupt vectors
SetTime (207)	Set system time
SetVerify (207)	Set verify flag
SwapVectors (207)	Swap interrupt vectors
UnPackTime (207)	Unpack file time

45.4 Process handling

Functions to handle process information and starting new processes.

Table 45.3:

Name	Description
DosExitCode (1)	Exit code of last executed program
EnvCount (207)	Return number of environment variables
EnvStr (207)	Return environment string pair
Exec (207)	Execute program
GetEnv (207)	Return specified environment string

45.5 Directory and disk handling

Routines to handle disk information.

Table 45.4:

Name	Description
AddDisk (207)	Add disk to list of disks (UNIX only)
DiskFree (207)	Return size of free disk space
DiskSize (207)	Return total disk size

45.6 File handling

Routines to handle files on disk.

Table 45.5:

Name	Description
FExpand (207)	Expand filename to full path
FindClose (207)	Close finfirst/findnext session
FindFirst (207)	Start find of file
FindNext (207)	Find next file
FSearch (207)	Search for file in a path
FSplit (207)	Split filename in parts
GetFAttr (207)	Return file attributes
GetFTime (207)	Return file time
GetLongName (207)	Convert short filename to long filename (DOS only)
GetShortName (207)	Convert long filename to short filename (DOS only)
SetFAttr (207)	Set file attributes
SetFTime (207)	Set file time

45.7 File open mode constants.

These constants are used in the `Mode` field of the `TextRec` record. Gives information on the file-mode of the text I/O. For their definitions consult the following table:

Table 45.6: Possible mode constants

Constant	Description	Value
`fmclosed`	File is closed	$D7B0
`fminput`	File is read only	$D7B1
`fmoutput`	File is write only	$D7B2
`fminout`	File is read and write	$D7B3

45.8 File attributes

The File Attribute constants are used in FindFirst (207), FindNext (207) to determine what type of special file to search for in addition to normal files. These flags are also used in the SetFAttr (207) and GetFAttr (207) routines to set and retrieve attributes of files. For their definitions consult fileattributes (209).

Table 45.7: Possible file attributes

Constant	Description	Value
`readonly`	Read-Only file attribute	$01
`hidden`	Hidden file attribute	$02
`sysfile`	System file attribute	$04
`volumeid`	Volumd ID file attribute	$08
`directory`	Directory file attribute	$10
`archive`	Archive file attribute	$20
`anyfile`	Match any file attribute	$3F

45.9 Constants, types and variables

45.9.1 Constants

```
FileNameLen = 255
```

Maximum length of a filename

45.9.2 Types

```
SearchRec = packed record
  SearchPos : TOff;
  SearchNum : LongInt;
  DirPtr : Pointer;
  SearchType : Byte;
  SearchAttr : Byte;
  Mode : Word;
  Fill : Array[1..1] of Byte;
  Attr : Byte;
  Time : LongInt;
  Size : LongInt;
  Reserved : Word;
  Name : string;
  SearchSpec : string;
  NamePos : Word;
end
```

`SearchRec` is filled by the FindFirst (207) call and can be used in subsequent FindNext (207) calls to search for files. The structure of this record depends on the platform. Only the following fields are present on all platforms:

Attr File attributes.

Time File modification time.

Size File size

Name File name (name part only, no path)

Mode File access mode (linux only)

Chapter 46

Reference for unit 'dxeload'

46.1 Overview

The dxeload unit was implemented by Pierre Mueller for dos, it allows to load a DXE file (an object file with 1 entry point) into memory and return a pointer to the entry point.

It exists only for dos.

46.2 Procedures and functions

46.2.1 dxe_load

Synopsis: Load DXE file in memory

Declaration: `function dxe_load(filename: string) : pointer`

Visibility: default

Description: `dxe_load` loads the contents of the file `filename` into memory. It performs the necessary relocations in the object code, and returns then a pointer to the entry point of the code.

For an example, see the emu387 (213) unit in the RTL.

Errors: If an error occurs during the load or relocations, `Nil` is returned.

Chapter 47

Reference for unit 'dynlibs'

47.1 Overview

The Dynlibs unit provides support for dynamically loading shared libraries. It is available only on those platforms that support shared libraries. The functionality available here may only be a part of the functionality available on each separate platform, in the interest of portability.

On unix platforms, using this unit will cause the program to be linked to the C library, as most shared libraries are implemented in C and the dynamical linker too.

Chapter 48

Reference for unit 'emu387'

48.1 Overview

The emu387 unit was written by Pierre Mueller for dos. It sets up the coprocessor emulation for FPC under dos. It is not necessary to use this unit on other OS platforms because they either simply do not run on a machine without coprocessor, or they provide the coprocessor emulation themselves.

It shouldn't be necessary to use the function in this unit, it should be enough to place this unit in the `uses` clause of your program to enable the coprocessor emulation under dos. The unit initialization code will try and load the coprocessor emulation code and initialize it.

48.2 Procedures and functions

48.2.1 npxsetup

Synopsis: Set up coprocessor emulation.

Declaration: `procedure npxsetup(prog_name: string)`

Visibility: default

Description: `npxsetup` checks whether a coprocessor is found. If not, it loads the file wmemu387.dxe into memory and initializes the code in it.

If the environment variable `387` is set to `N`, then the emulation will be loaded, even if there is a coprocessor present. If the variable doesn't exist, or is set to any other value, the unit will try to detect the presence of a coprocessor unit.

The function searches the file wmemu387.dxe in the following way:

1. If the environment variable `EMU387` is set, then it is assumed to point at the wmemu387.dxe file.
2. if the environment variable `EMU387` does not exist, then the function will take the path part of `prog_name` and look in that directory for the file wmemu387.dxe.

It should never be necessary to call this function, because the initialization code of the unit contains a call to the function with as an argument `paramstr(0)`. This means that you should deliver the file `wmemu387.dxe` together with your program.

Errors: If there is an error, an error message is printed to standard error, and the program is halted, since any floating-point code is bound to fail anyhow.

Chapter 49

Reference for unit 'errors'

49.1 Used units

Table 49.1: Used units by unit 'errors'

Name	Page
System	622
unixtype	747

49.2 Overview

The errors unit contains routines to convert a unix system call error code to an error message: StrError (214). It is only available on unix platforms.

Chapter 50

Reference for unit 'exeinfo'

50.1 Overview

The exeinfo unit implements some cross-platform routines to examine the contents of an executable: information about sections, mapping addresses to loaded modules etc.

It is mainly used by the lineinfo (341) and lnfodwrf (370) unit to examine the binary for debug info.

50.2 Constants, types and variables

50.2.1 Types

```
TExeFile = record
  f : File;
  size : Int64;
  isopen : Boolean;
  nsects : LongInt;
  sechdrofs : ptruint;
  secstrofs : ptruint;
  processaddress : ptruint;
  FunctionRelative : Boolean;
  ImgOffset : ptruint;
  filename : string;
  buf : Array[0..4095] of Byte;
  bufsize : LongInt;
  bufcnt : LongInt;
end
```

`TExeFile` is a record used in the various calls of this unit. It contains a file descriptor, and various fields that describe the executable.

The structure of `TExeFile` is opaque, that is, one shouldn't rely on the exactness of this structure, it may change any time in the future.

50.3 Procedures and functions

50.3.1 CloseExeFile

Synopsis: Close a previously opened file.

Declaration:
```
function CloseExeFile(var e: TExeFile) : Boolean
```

Visibility: default

Description: `CloseExeFile` closes an executable file image previously opened with OpenExeFile (216), and represented by `e`.

The function returns `True` if the file was closed succesfully, or `False` if something went wrong.

Errors: In case of an error, `False` is returned.

See also: OpenExeFile (216)

50.3.2 FindExeSection

Synopsis: Find a section in the binary image.

Declaration:
```
function FindExeSection(var e: TExeFile;const secname: string;
                        var secofs: LongInt;var seclen: LongInt)
                         : Boolean
```

Visibility: default

Description: `FindExeSection` examines the binary that was opened with OpenExeFile (216) (represented by `e`) and searches for the section named `secname`. If found, the section offset is returned in `secofs` and the section length (in bytes) is returned in `seclen`.

The function returns `True` if the section was found, `False` if not.

See also: OpenExeFile (216)

50.3.3 GetModuleByAddr

Synopsis: Return the module name by address

Declaration:
```
procedure GetModuleByAddr(addr: pointer;var baseaddr: pointer;
                          var filename: string)
```

Visibility: default

Description: `GetModuleByAddr` returns the name of the module that contains address `addr`. If succesful, it returns `True` and returns the filename in `FileName` and the base address at which it is loaded in `BaseAddr`.

50.3.4 OpenExeFile

Synopsis: Open an executable file

Declaration:
```
function OpenExeFile(var e: TExeFile;const fn: string) : Boolean
```

Visibility: default

Description: `OpenExeFile` opens the executable file `fn` and initializes the structure `e` for subsequent calls to routines in the `exeinfo` unit.

The function returns `True` if the file was opened succesfully, false otherwise.

See also: FindExeSection (216), CloseExeFile (216), ReadDebugLink (217)

50.3.5 ReadDebugLink

Synopsis: Read the location of a debug info filename

Declaration: `function ReadDebugLink(var e: TExeFile;var dbgfn: string) : Boolean`

Visibility: default

Description: `ReadDebugLink` examines the `.gnu_debuglink` section to see if the debug information is stored in an external file. If so, then the name of the file with the debug information is returned in the `dbgfn` parameter.

The function returns `false` if there is no external debug information file, or if the file with debug information does not exist. It is searched next to the binary file or in the current directory.

See also: OpenExeFile (216), CloseExeFile (216)

Chapter 51

Reference for unit 'fgl'

51.1 Used units

Table 51.1: Used units by unit 'fgl'

Name	Page
System	622
sysutils	628
Types	637

51.2 Overview

The `fgl` unit contains some basic list-related generic classes.

51.3 Constants, types and variables

51.3.1 Constants

```
MaxGListSize = MaxInt div 1024
```

`MaxGListSize` is the maximum number of elements in the TFPGList (223) list.

```
MaxListSize = Maxint div 16
```

`MaxListSize` is the maximum number of elements a list can contain before the memory runs out.

51.3.2 Types

```
TFPSListCompareFunc = function(Key1: Pointer;Key2: Pointer) : Integer
                                of object
```

`TFPSListCompareFunc` is used in the TFPSList.Sort (247) method to compare 2 elements. The list passes 2 pointers to the actual items to the compare function. The result of this function determines how the pointers will be sorted:

- If the result of this function is negative, the first key (`key1`) is assumed to be 'less' than the second key (`key2`) and will be moved before the second in the list.
- If the function result is positive, the first key (`key1`) pointer is assumed to be 'greater than' the second key (`key2`)and will be moved after the second in the list.
- if the function result is zero, the keys are assumed to be 'equal' and no moving will take place.

51.4 EListError

51.4.1 Description

`EListError` is the exception used in the TFPSList (242) class to indicate errors such as a list index out of bounds, wrong capacity etc.

See also: TFPSList.Capacity (248), TFPSList.Exchange (245), TFPSList.Items (248)

51.5 TFPGInterfacedObjectList

51.5.1 Description

`TFPGList` can be used to specialize a list for any class type `T` that requires reference counting (all objects that implement `IInterface` or `IUnknown`)). It will specialize to a list with the same methods as TFPSList (242) or classes.TFPList (218) or `TFPObjectList`

Classes that implement `IInterface` or `IUnknown` require special care to maintain the reference count. The `TFPGInterfacedObjectList` list provides the necessary functionality to deal with this.

See also: TFPSList (242), classes.TFPList (218)

51.5.2 Method overview

Page	Method	Description
220	Add	Add new object of class `T` to the list.
221	Assign	Copy objects from Source list
220	Create	Instantiate a new interfaced object list.
220	Extract	Extract an item from the list
220	GetEnumerator	Return a list enumerator for `T`
221	IndexOf	Index of object
221	Insert	Insert a new object in the list
221	Remove	Remove an object from the list.
222	Sort	Sort the objects in the list

51.5.3 Property overview

Page	Properties	Access	Description
222	First	rw	First non-nil object
223	Items	rw	Indexed access to objects in the list.
222	Last	rw	Last non-`Nil` object
223	List	r	Internal list pointer

51.5.4 TFPGInterfacedObjectList.Create

Synopsis: Instantiate a new interfaced object list.

Declaration: `constructor Create`

Visibility: public

Description: `Create` instantiates a new object list. It will simply call the inherited constructor with the correct item size.

See also: TFPSList.Destroy (243)

51.5.5 TFPGInterfacedObjectList.Add

Synopsis: Add new object of class `T` to the list.

Declaration: `function Add(const Item: T) : Integer`

Visibility: public

Description: `Add` adds a new item `Item` of class type `T` to the list and returns the position at which the item was added. Add will increase the reference count of the object.

Errors: If the item could not be added, an EListError (219) exception is raised.

See also: TFPGInterfacedObjectList.Extract (220), TFPGInterfacedObjectList.Items (223), TFPGInterfacedObjectList.IndexOf (221)

51.5.6 TFPGInterfacedObjectList.Extract

Synopsis: Extract an item from the list

Declaration: `function Extract(const Item: T) : T`

Visibility: public

Description: `Extract` removes `Item` from the list and returns the removed item, or `Nil` if it was not found.

The extracted object will not be destroyed.

Errors: None.

See also: TFPSList.Delete (244)

51.5.7 TFPGInterfacedObjectList.GetEnumerator

Synopsis: Return a list enumerator for `T`

Declaration: `function GetEnumerator : TFPGListEnumeratorSpec`

Visibility: public

Description: `GetEnumerator` returns an enumerator for the elements in the list. It is a specialized version of TFPGListEnumerator (227).

See also: TFPGListEnumerator (227)

51.5.8 TFPGInterfacedObjectList.IndexOf

Synopsis: Index of object

Declaration: `function IndexOf(const Item: T) : Integer`

Visibility: public

Description: `IndexOf` returns the index of `Item` in the list, or -1 if the object does not appear in the list.

Errors: None.

See also: TFPGInterfacedObjectList.Items (223), TFPGInterfacedObjectList.Insert (221), TFPGInterfacedObjectList.Add (220)

51.5.9 TFPGInterfacedObjectList.Insert

Synopsis: Insert a new object in the list

Declaration: `procedure Insert(Index: Integer;const Item: T)`

Visibility: public

Description: `Insert` inserts a new object (`Item`) in the list at position `Index`. The index is zero based and must be less than Count (248).

Errors: If an invalid index is specified, an EListError (219) exception is raised.

See also: TFPGInterfacedObjectList.Items (223), TFPGInterfacedObjectList.Add (220)

51.5.10 TFPGInterfacedObjectList.Assign

Synopsis: Copy objects from Source list

Declaration: `procedure Assign(Source: TFPGInterfacedObjectList)`

Visibility: public

Description: `Assign` clears the list and copies all items in `Source` to the list. The source list must be of the same type as the destination list.

See also: TFPSList.Clear (244), TFPGObjectList.Add (238)

51.5.11 TFPGInterfacedObjectList.Remove

Synopsis: Remove an object from the list.

Declaration: `function Remove(const Item: T) : Integer`

Visibility: public

Description: `Remove` removes the object `Item` from the list, and returns the index of the removed item. If no item was removed, `-1` is returned. Only the first object is removed.

Removing an object from the list may cause the object to be freed.

Errors: None.

See also: TFPGInterfacedObjectList.IndexOf (221), TFPSList.Delete (244), TFPGInterfacedObjectList.FreeObjects (219)

51.5.12 TFPGInterfacedObjectList.Sort

Synopsis: Sort the objects in the list

Declaration: `procedure Sort(Compare: TCompareFunc)`

Visibility: public

Description: `Sort` sorts the elements in the list using the provided `Compare` function. The list passes 2 items to the compare function. The result of this function determines how the items will be sorted:

- If the result of this function is negative, the first object (`Item1`) is assumed to be 'less' than the second object (`Item2`) and will be moved before the second in the list.
- If the function result is positive, the first object (`Item1`) is assumed to be 'greater than' the second object (`Item2`)and will be moved after the second in the list.
- if the function result is zero, the objects are assumed to be 'equal' and no moving will take place.

Errors: None.

See also: TFPSList.Sorted (242)

51.5.13 TFPGInterfacedObjectList.First

Synopsis: First non-nil object

Declaration: `Property First : T`

Visibility: public

Access: Read,Write

Description: `First` returns the first non-nil object. If no such element is present, `Nil` is returned.

See also: TFPSList.First (249), TFPGInterfacedObjectList.Last (222), TFPSList.Pack (247)

51.5.14 TFPGInterfacedObjectList.Last

Synopsis: Last non-`Nil` object

Declaration: `Property Last : T`

Visibility: public

Access: Read,Write

Description: `Last` returns the last non-`Nil` object. If no such element is present, `Nil` is returned.

See also: TFPGInterfacedObjectList.First (222), TFPSList.Last (249)

51.5.15 TFPGInterfacedObjectList.Items

Synopsis: Indexed access to objects in the list.

Declaration: `Property Items[Index: Integer]: T; default`

Visibility: public

Access: Read,Write

Description: `Items` provides indexed access to the objects in the list. The objects can be get or set.

The index `Index` is zero based, and has a maximum value of Count-1 (248).

The previous object at position `Index` may be freed when setting the property, depending on its reference count.

Errors: If an invalid index is used, an EListError (219) exception is raised.

See also: TFPSList.Count (248)

51.5.16 TFPGInterfacedObjectList.List

Synopsis: Internal list pointer

Declaration: `Property List : PTypeList`

Visibility: public

Access: Read

Description: `List` is the internal list of objects. It should not be used directly.

See also: TFPGInterfacedObjectList.Items (223)

51.6 TFPGList

51.6.1 Description

`TFPGList` can be used to specialize a list for any type `T` that does not require reference counting (such as interfaced objects). It will specialize to a list with the same methods as TFPSList (242) or classes.TFPList (218)

See also: TFPSList (242), classes.TFPList (218)

51.6.2 Method overview

Page	Method	Description
224	Add	Add new item of type `T` to the list.
225	Assign	Copy elements from Source list
224	Create	Instantiate a new list
224	Extract	Extract an item from the list
224	GetEnumerator	Return a list enumerator for `T`.
225	IndexOf	Index of item
225	Insert	Insert a new item in the list
225	Remove	Remove an item from the list.
226	Sort	Sort the list

51.6.3 Property overview

Page	Properties	Access	Description
226	First	rw	First non-empty item
226	Items	rw	Indexed access to items in the list
226	Last	rw	Last non-empty item
227	List	r	Internal list object

51.6.4 TFPGList.Create

Synopsis: Instantiate a new list

Declaration: `constructor Create`

Visibility: public

Description: `Create` instantiates a new list. It will simply call the inherited constructor with the correct item size: `sizeof(T)`.

51.6.5 TFPGList.Add

Synopsis: Add new item of type `T` to the list.

Declaration: `function Add(const Item: T) : Integer`

Visibility: public

Description: `Add` adds a new item `Item` of generic type `T` to the list and returns the position at which the item was added.

Errors: If the item could not be added, an EListError (219) exception is raised.

See also: TFPGList.Extract (224), TFPGList.Items (226), TFPGList.IndexOf (225)

51.6.6 TFPGList.Extract

Synopsis: Extract an item from the list

Declaration: `function Extract(const Item: T) : T`

Visibility: public

Description: `Extract` removes `Item` from the list and returns the removed item, or an expression equivalent to `T(0)` if it was not found.

Errors: None.

See also: TFPSList.Delete (244)

51.6.7 TFPGList.GetEnumerator

Synopsis: Return a list enumerator for `T`.

Declaration: `function GetEnumerator : TFPGListEnumeratorSpec`

Visibility: public

Description: `GetEnumerator` returns an enumerator for the elements in the list. It is a specialized version of TFPGListEnumerator (227).

See also: TFPGListEnumerator (227)

51.6.8 TFPGList.IndexOf

Synopsis: Index of item

Declaration: `function IndexOf(const Item: T) : Integer`

Visibility: public

Description: `IndexOf` returns the index of `Item` in the list, or -1 if the item does not appear in the list.

Errors: None.

See also: TFPGList.Items (226), TFPGList.Insert (225), TFPSList.Add (244)

51.6.9 TFPGList.Insert

Synopsis: Insert a new item in the list

Declaration: `procedure Insert(Index: Integer;const Item: T)`

Visibility: public

Description: `Insert` inserts a new `item` in the list at position `Index`. The index is zero based and must be less than Count (248).

Errors: If an invalid index is specified, an EListError (219) exception is raised.

See also: TFPGList.Items (226), TFPGList.Insert (225), TFPSList.Add (244)

51.6.10 TFPGList.Assign

Synopsis: Copy elements from Source list

Declaration: `procedure Assign(Source: TFPGList)`

Visibility: public

Description: `Assign` clears the list and copies all items in `Source` to the list. The source list must be of the same type as the destination list.

See also: TFPSList.Clear (244), TFPGList.Add (224)

51.6.11 TFPGList.Remove

Synopsis: Remove an item from the list.

Declaration: `function Remove(const Item: T) : Integer`

Visibility: public

Description: `Remove` removes the item `Item` from the list, and returns the index of the removed item. If no item was removed, `-1` is returned. Only the first item is removed.

Errors: None.

See also: TFPGList.IndexOf (225)

51.6.12 TFPGList.Sort

Synopsis: Sort the list

Declaration: `procedure Sort(Compare: TCompareFunc)`

Visibility: public

Description: `Sort` sorts the elements in the list using the provided `Compare` function. The list passes 2 items to the compare function. The result of this function determines how the items will be sorted:

- If the result of this function is negative, the first item (`Item1`) is assumed to be 'less' than the second item (`Item2`) and will be moved before the second in the list.
- If the function result is positive, the first item (`Item1`) is assumed to be 'greater than' the second item (`Item2`)and will be moved after the second in the list.
- if the function result is zero, the items are assumed to be 'equal' and no moving will take place.

51.6.13 TFPGList.First

Synopsis: First non-empty item

Declaration: `Property First : T`

Visibility: public

Access: Read,Write

Description: `First` returns the first non-empty item, which means the first item not equal to `T(0)`. If no such element is present, `T(0)`. is returned.

See also: TFPSList.First (249), TFPGList.Last (226)

51.6.14 TFPGList.Last

Synopsis: Last non-empty item

Declaration: `Property Last : T`

Visibility: public

Access: Read,Write

Description: `Last` returns the last non-empty item, which means the last item not equal to `T(0)`. If no such element is present, `T(0)`. is returned.

See also: TFPGList.First (226), TFPSList.Last (249)

51.6.15 TFPGList.Items

Synopsis: Indexed access to items in the list

Declaration: `Property Items[Index: Integer]: T; default`

Visibility: public

Access: Read,Write

Description: `Items` provides indexed access to the items in the list. The items can be get or set.

The index `Index` is zero based, and has a maximum value of Count-1 (248).

Errors: If an invalid index is used, an EListError (219) exception is raised.

See also: TFPSList.Count (248)

51.6.16 TFPGList.List

Synopsis: Internal list object

Declaration: `Property List : PTypeList`

Visibility: public

Access: Read

Description: `List` is the internal list of items. It should not be used directly.

See also: TFPGList.Items (226)

51.7 TFPGListEnumerator

51.7.1 Description

`TFPGListEnumerator` is a generic list enumerator. It is used in the TFPGList (223) class to implement the enumerator for the list.

Normally there should be no need to instantiate or use this class directly.

See also: TFPGList (223)

51.7.2 Method overview

Page	Method	Description
227	Create	Create a new list enumerator
228	MoveNext	Move to next element in the list

51.7.3 Property overview

Page	Properties	Access	Description
228	Current	r	Current enumerated element

51.7.4 TFPGListEnumerator.Create

Synopsis: Create a new list enumerator

Declaration: `constructor Create(AList: TFPSList)`

Visibility: public

Description: `Create` is called by the list `AList` to initialize a new enumerator. There should be no need to call this directly.

See also: TFPGList (223)

51.7.5 TFPGListEnumerator.MoveNext

Synopsis: Move to next element in the list

Declaration: `function MoveNext : Boolean`

Visibility: public

Description: `MoveNext` moves to the next element in the list.

See also: TFPGListEnumerator.Current (228)

51.7.6 TFPGListEnumerator.Current

Synopsis: Current enumerated element

Declaration: `Property Current : T`

Visibility: public

Access: Read

Description: `Current` returns the currently enumerated element. It is only valid after TFPGListEnumerator.MoveNext (228) was called and returned `True`.

See also: TFPGListEnumerator.MoveNext (228)

51.8 TFPGMap

51.8.1 Description

`TFPGMap` is a generic map class. It can be used to specialize a map for any key type and data type that do not require manual reference counting: For reference counted interface objects, TFPGMapInterfacedObjectData (233) must be used.

See also: TFPGMapInterfacedObjectData (233)

51.8.2 Method overview

Page	Method	Description
229	Add	Add a key and value to the map
229	Create	Create a new instance of the map
229	Find	Find item based on key
230	IndexOf	Find index of a key in the list.
230	IndexOfData	Find index of data value in the list.
230	InsertKey	Insert a new key in the list
230	InsertKeyData	Insert a new key with associated data in the list
231	Remove	Remove a key from the list

51.8.3 Property overview

Page	Properties	Access	Description
231	Data	rw	Indexed access to the data in the list
231	KeyData	rw	Access to data based on key
231	Keys	rw	Indexed access to the keys in the list.
232	OnCompare	rw	Alias for `OnKeyCompare`
232	OnDataCompare	rw	Compare function for data values.
232	OnKeyCompare	rw	Compare function for key values.

51.8.4 TFPGMap.Create

Synopsis: Create a new instance of the map

Declaration:
```
constructor Create
```

Visibility: public

Description: `Create` instantiates a new map. It mainly initializes the TFPSMap (250) parent with the sized of the key and data.

See also: TFPSMap.Create (250)

51.8.5 TFPGMap.Add

Synopsis: Add a key and value to the map

Declaration:
```
function Add(const AKey: TKey;const AData: TData) : Integer
function Add(const AKey: TKey) : Integer
```

Visibility: public

Description: `Add` adds a new key `AKey` of generic type `TKey` with data value `AData` to the list and returns the position at which the key was added.

Errors: If the item could not be added, an EListError (219) exception is raised. If Duplicates (253) is set to `dupError` and a duplicate key is added, an EListError (219) exception is raised.

See also: TFPGMap.Keys (231), TFPGMap.IndexOf (230), TFPGMap.KeyData (231), TFPGMap.Data (231), TFPSMap.Duplicates (253)

51.8.6 TFPGMap.Find

Synopsis: Find item based on key

Declaration:
```
function Find(const AKey: TKey;out Index: Integer) : Boolean
```

Visibility: public

Description: `Find` will search the first key smaller than or equal to `AKey` and return its index in `AIndex`. If the key was not found then -1 is returned. The return value of the function is `True` if an exact match for `AKey` is found, `False` otherwise.

See also: TFPGMap.IndexOf (230), TFPGMap.IndexOfData (230)

51.8.7 TFPGMap.IndexOf

Synopsis: Find index of a key in the list.

Declaration:
```
function IndexOf(const AKey: TKey) : Integer
```

Visibility: public

Description: `IndexOf` returns the index of `AKey` in the list, or -1 if the key was not found in the list.

Errors: None.

See also: TFPGMap.Find (229), TFPGMap.IndexOfData (230)

51.8.8 TFPGMap.IndexOfData

Synopsis: Find index of data value in the list.

Declaration:
```
function IndexOfData(const AData: TData) : Integer
```

Visibility: public

Description: `IndexOfData` returns the index of `AData` in the list, or -1 if the data was not found in the list.

Errors: None.

See also: TFPGMap.Find (229), TFPGMap.IndexOf (230)

51.8.9 TFPGMap.InsertKey

Synopsis: Insert a new key in the list

Declaration:
```
procedure InsertKey(Index: Integer;const AKey: TKey)
```

Visibility: public

Description: `InsertKey` inserts key `AKey` at position `Index` in the list. It is not allowed to insert a key in a sorted list.

Errors: If the index `AIndex` is out of range [0..Count-1], or the list is sorted, an EListError (219) exception will be raised.

See also: TFPGMap.InsertKeyData (230), TFPGMap.Add (229), TFPSMap.Delete (250), TFPSMap.Remove (253)

51.8.10 TFPGMap.InsertKeyData

Synopsis: Insert a new key with associated data in the list

Declaration:
```
procedure InsertKeyData(Index: Integer;const AKey: TKey;
                        const AData: TData)
```

Visibility: public

Description: `InsertKey` inserts key `AKey` with associated data `AData` at position `Index` in the list. It is not allowed to insert a key in a sorted list.

Errors: If the index `AIndex` is out of range [0..Count-1], or the list is sorted, an EListError (219) exception will be raised.

See also: TFPGMap.InsertKey (230), TFPGMap.Add (229), TFPSMap.Delete (250), TFPGMap.Remove (231)

51.8.11 TFPGMap.Remove

Synopsis: Remove a key from the list

Declaration: `function Remove(const AKey: TKey) : Integer`

Visibility: public

Description: `Remove` removes the key `AKey` from the list, together with its associated data. The function returns the index of `AKey` prior to removal from the list, or -1 if `AKey` was not present in the list.

Errors: None.

See also: TFPGMap.InsertKey (230), TFPGMap.InsertKeyData (230), TFPGMap.Add (229), TFPSMap.Delete (250)

51.8.12 TFPGMap.Keys

Synopsis: Indexed access to the keys in the list.

Declaration: `Property Keys[Index: Integer]: TKey`

Visibility: public

Access: Read,Write

Description: `Keys` provides indexed access to the key values in the list. Valid values for `Index` are in the range `[0..Count-1]`. Key values can always be read, but can only be written if the list is unsorted.

Errors: If the index `AIndex` is out of range [0..Count-1], an EListError (219) exception will be raised. The same exception is raised if a key is written and the list is sorted.

See also: TFPSList.Count (248), TFPGMap.Data (231), TFPGMap.KeyData (231)

51.8.13 TFPGMap.Data

Synopsis: Indexed access to the data in the list

Declaration: `Property Data[Index: Integer]: TData`

Visibility: public

Access: Read,Write

Description: `Data` provides indexed access to the data values in the list. Valid values for `Index` are in the range `[0..Count-1]`. Data can always be read or written.

Errors: If the index `AIndex` is out of range [0..Count-1], an EListError (219) exception will be raised.

See also: TFPSList.Count (248), TFPGMap.Keys (231), TFPGMap.KeyData (231)

51.8.14 TFPGMap.KeyData

Synopsis: Access to data based on key

Declaration: `Property KeyData[AKey: TKey]: TData; default`

Visibility: public

Access: Read,Write

Description: `KeyData` allows access to the data based on the key value `AKey`. The data can be read and written. When writing, writing using an existing key will overwrite the current data. If it does not exist yet, it will be created. When reading, if the key is not present, an EListError (219) will be raised.

Errors: If the key does not exist, an EListError (219) exception will be raised.

See also: TFPSList.Count (248), TFPGMap.Keys (231), TFPGMap.Data (231)

51.8.15 TFPGMap.OnCompare

Synopsis: Alias for `OnKeyCompare`

Declaration: `Property OnCompare : TKeyCompareFunc`

Visibility: public

Access: Read,Write

Description: `OnCompare` is a deprecated property, use TFPGMap.OnKeyCompare (232) instead.

See also: TFPGMap.OnKeyCompare (232)

51.8.16 TFPGMap.OnKeyCompare

Synopsis: Compare function for key values.

Declaration: `Property OnKeyCompare : TKeyCompareFunc`

Visibility: public

Access: Read,Write

Description: `OnKeyCompare` can be set to a function that compares key values. The default value for this event is a function that compares keys based on a byte-by-byte comparison of the memory block. The function must have the following semantics:

- If the result of this function is negative, the first key (`key1`) is assumed to be 'less' than the second key (`key2`) and will be moved before the second in the list.
- If the function result is positive, the first key (`key1`) pointer is assumed to be 'greater than' the second key (`key2`)and will be moved after the second in the list.
- if the function result is zero, the keys are assumed to be 'equal' and no moving will take place.

See also: TFPGMap.OnDataCompare (232)

51.8.17 TFPGMap.OnDataCompare

Synopsis: Compare function for data values.

Declaration: `Property OnDataCompare : TDataCompareFunc`

Visibility: public

Access: Read,Write

Description: `OnDataCompare` can be set to a function that compares data values. The default value for this event is a function that compares data based on a byte-by-byte comparison of the memory block. The function must have the following semantics:

- If the result of this function is negative, the first data item (`Data1`) is assumed to be 'less' than the second data item (`Data2`) and will be moved before the second in the list.
- If the function result is positive, the first data item (`Data1`) pointer is assumed to be 'greater than' the second data item (`Data2`)and will be moved after the second in the list.
- if the function result is zero, the data items are assumed to be 'equal' and no moving will take place.

See also: TFPGMap.OnKeyCompare (232)

51.9 TFPGMapInterfacedObjectData

51.9.1 Description

`TFPGInterfacedObjectMap` is a generic map class. It can be used to specialize a map for any key type, with associated data type that requires manual reference counting: any type which implements `IInterface`. For non-reference counted objects, TFPGMap (228) should be used.

This map class is entirely equivalend to TFPGMap (228), but operates on data items that require additional reference counting code on the data.

See also: TFPGMap (228)

51.9.2 Method overview

Page	Method	Description
234	Add	Add a key and value to the map
233	Create	Create a new instance of the map
234	Find	Find item based on key
234	IndexOf	Find index of a key in the list.
235	IndexOfData	Find index of data value in the list.
235	InsertKey	Insert a new key in the list
235	InsertKeyData	Insert a new key with associated data in the list
235	Remove	Remove a key from the list

51.9.3 Property overview

Page	Properties	Access	Description
236	Data	rw	Indexed access to the data in the list
236	KeyData	rw	Access to data based on key
236	Keys	rw	Indexed access to the keys in the list.
237	OnCompare	rw	Alias for `OnKeyCompare`
237	OnDataCompare	rw	Compare function for data values.
237	OnKeyCompare	rw	Compare function for key values.

51.9.4 TFPGMapInterfacedObjectData.Create

Synopsis: Create a new instance of the map

Declaration: `constructor Create`

Visibility: public

Description: `Create` instantiates a new map. It mainly initializes the TFPSMap (250) parent with the sized of the key and data.

See also: TFPSMap.Create (250)

51.9.5 TFPGMapInterfacedObjectData.Add

Synopsis: Add a key and value to the map

Declaration:
```
function Add(const AKey: TKey;const AData: TData) : Integer
function Add(const AKey: TKey) : Integer
```

Visibility: public

Description: `Add` adds a new key `AKey` of generic type `TKey` with data value `AData` to the list and returns the position at which the key was added.

Errors: If the item could not be added, an EListError (219) exception is raised. If Duplicates (253) is set to `dupError` and a duplicate key is added, an EListError (219) exception is raised.

See also: TFPGMapInterfacedObjectData.Keys (236), TFPGMapInterfacedObjectData.IndexOf (234), TFPGMapInterfacedObjectData.KeyData (236), TFPGMapInterfacedObjectData.Data (236), TFPS.Duplicates (218)

51.9.6 TFPGMapInterfacedObjectData.Find

Synopsis: Find item based on key

Declaration:
```
function Find(const AKey: TKey;out Index: Integer) : Boolean
```

Visibility: public

Description: `Find` will search the first key smaller than or equal to `AKey` and return its index in `AIndex`. If the key was not found then -1 is returned. The return value of the function is `True` if an exact match for `AKey` is found, `False` otherwise.

See also: TFPGMapInterfacedObjectData.IndexOf (234), TFPGMapInterfacedObjectData.IndexOfData (235)

51.9.7 TFPGMapInterfacedObjectData.IndexOf

Synopsis: Find index of a key in the list.

Declaration:
```
function IndexOf(const AKey: TKey) : Integer
```

Visibility: public

Description: `IndexOf` returns the index of `AKey` in the list, or -1 if the key was not found in the list.

Errors: None.

See also: TFPGMapInterfacedObjectData.Find (234), TFPGMapInterfacedObjectData.IndexOfData (235)

51.9.8 TFPGMapInterfacedObjectData.IndexOfData

Synopsis: Find index of data value in the list.

Declaration:
```
function IndexOfData(const AData: TData) : Integer
```

Visibility: public

Description: `IndexOfData` returns the index of `AData` in the list, or -1 if the data was not found in the list.

Errors: None.

See also: TFPGMapInterfacedObjectData.Find (234), TFPGMapInterfacedObjectData.IndexOf (234)

51.9.9 TFPGMapInterfacedObjectData.InsertKey

Synopsis: Insert a new key in the list

Declaration:
```
procedure InsertKey(Index: Integer;const AKey: TKey)
```

Visibility: public

Description: `InsertKey` inserts key `AKey` at position `Index` in the list. It is not allowed to insert a key in a sorted list.

Errors: If the index `AIndex` is out of range [0..Count-1], or the list is sorted, an EListError (219) exception will be raised.

See also: TFPGMapInterfacedObjectData.InsertKeyData (235), TFPGMapInterfacedObjectData.Add (234), TFPSMapInterfacedObjectData.Delete (218), TFPSMapInterfacedObjectData.Remove (218)

51.9.10 TFPGMapInterfacedObjectData.InsertKeyData

Synopsis: Insert a new key with associated data in the list

Declaration:
```
procedure InsertKeyData(Index: Integer;const AKey: TKey;
                        const AData: TData)
```

Visibility: public

Description: `InsertKey` inserts key `AKey` with associated data `AData` at position `Index` in the list. It is not allowed to insert a key in a sorted list.

Errors: If the index `AIndex` is out of range [0..Count-1], or the list is sorted, an EListError (219) exception will be raised.

See also: TFPGMapInterfacedObjectData.InsertKey (235), TFPGMapInterfacedObjectData.Add (234), TFPSMapInterfacedObjectData.Delete (218), TFPGMapInterfacedObjectData.Remove (235)

51.9.11 TFPGMapInterfacedObjectData.Remove

Synopsis: Remove a key from the list

Declaration:
```
function Remove(const AKey: TKey) : Integer
```

Visibility: public

Description: `Remove` removes the key `AKey` from the list, together with its associated data. The function returns the index of `AKey` prior to removal from the list, or -1 if `AKey` was not present in the list.

Errors: None.

See also: TFPGMap.InsertKey (230), TFPGMap.InsertKeyData (230), TFPGMap.Add (229), TFPGMapInterfacedObjectData.Delete (233)

51.9.12 TFPGMapInterfacedObjectData.Keys

Synopsis: Indexed access to the keys in the list.

Declaration: `Property Keys[Index: Integer]: TKey`

Visibility: public

Access: Read,Write

Description: `Keys` provides indexed access to the key values in the list. Valid values for `Index` are in the range `[0..Count-1]`. Key values can always be read, but can only be written if the list is unsorted.

Errors: If the index `AIndex` is out of range [0..Count-1], an EListError (219) exception will be raised. The same exception is raised if a key is written and the list is sorted.

See also: TFPSList.Count (248), TFPGMapInterfacedObjectData.Data (236), TFPGMapInterfacedObjectData.KeyData (236)

51.9.13 TFPGMapInterfacedObjectData.Data

Synopsis: Indexed access to the data in the list

Declaration: `Property Data[Index: Integer]: TData`

Visibility: public

Access: Read,Write

Description: `Data` provides indexed access to the data values in the list. Valid values for `Index` are in the range `[0..Count-1]`. Data can always be read or written.

Errors: If the index `AIndex` is out of range [0..Count-1], an EListError (219) exception will be raised.

See also: TFPSList.Count (248), TFPGMapInterfacedObjectData.Keys (236), TFPGMapInterfacedObjectData.KeyData (236)

51.9.14 TFPGMapInterfacedObjectData.KeyData

Synopsis: Access to data based on key

Declaration: `Property KeyData[AKey: TKey]: TData; default`

Visibility: public

Access: Read,Write

Description: `KeyData` allows access to the data based on the key value `AKey`. The data can be read and written. When writing, writing using an existing key will overwrite the current data. If it does not exist yet, it will be created. When reading, if the key is not present, an EListError (219) will be raised.

Errors: If the key does not exist, an EListError (219) exception will be raised.

See also: TFPSList.Count (248), TFPGMapInterfacedObjectData.Keys (236), TFPGMapInterfacedObjectData.Data (236)

51.9.15 TFPGMapInterfacedObjectData.OnCompare

Synopsis: Alias for `OnKeyCompare`

Declaration: `Property OnCompare : TKeyCompareFunc`

Visibility: public

Access: Read,Write

Description: `OnCompare` is a deprecated property, use TFPGMapInterfacedObjectData.OnKeyCompare (237) instead.

See also: TFPGMapInterfacedObjectData.OnKeyCompare (237)

51.9.16 TFPGMapInterfacedObjectData.OnKeyCompare

Synopsis: Compare function for key values.

Declaration: `Property OnKeyCompare : TKeyCompareFunc`

Visibility: public

Access: Read,Write

Description: `OnKeyCompare` can be set to a function that compares key values. The default value for this event is a function that compares keys based on a byte-by-byte comparison of the memory block. The function must have the following semantics:

- If the result of this function is negative, the first key (`key1`) is assumed to be 'less' than the second key (`key2`) and will be moved before the second in the list.
- If the function result is positive, the first key (`key1`) pointer is assumed to be 'greater than' the second key (`key2`)and will be moved after the second in the list.
- if the function result is zero, the keys are assumed to be 'equal' and no moving will take place.

See also: TFPGMapInterfacedObjectData.OnDataCompare (237)

51.9.17 TFPGMapInterfacedObjectData.OnDataCompare

Synopsis: Compare function for data values.

Declaration: `Property OnDataCompare : TDataCompareFunc`

Visibility: public

Access: Read,Write

Description: `OnDataCompare` can be set to a function that compares data values. The default value for this event is a function that compares data based on a byte-by-byte comparison of the memory block. The function must have the following semantics:

- If the result of this function is negative, the first data item (`Data1`) is assumed to be 'less' than the second data item (`Data2`) and will be moved before the second in the list.
- If the function result is positive, the first data item (`Data1`) pointer is assumed to be 'greater than' the second data item (`Data2`)and will be moved after the second in the list.
- if the function result is zero, the data items are assumed to be 'equal' and no moving will take place.

See also: TFPGMapInterfacedObjectData.OnKeyCompare (237)

51.10 TFPGObjectList

51.10.1 Description

`TFPGList` can be used to specialize a list for any class type `T` that does not require reference counting (such as interfaced objects). It will specialize to a list with the same methods as TFPSList (242) or classes.TFPList (218) or `TFPObjectList`

See also: TFPSList (242), classes.TFPList (218)

51.10.2 Method overview

Page	Method	Description
238	Add	Add new object of class `T` to the list.
240	Assign	Copy objects from Source list
238	Create	Instantiate a new object list.
239	Extract	Extract an item from the list
239	GetEnumerator	Return a list enumerator for `T`.
239	IndexOf	Index of item
240	Insert	Insert a new object in the list
240	Remove	Remove an object from the list.
240	Sort	Sort the objects in the list

51.10.3 Property overview

Page	Properties	Access	Description
241	First	rw	First non-nil item
242	FreeObjects	rw	Does the list own the objects or not?
241	Items	rw	Indexed access to objects in the list.
241	Last	rw	Last non-`Nil` object
242	List	r	Internal list pointer

51.10.4 TFPGObjectList.Create

Synopsis: Instantiate a new object list.

Declaration: `constructor Create(FreeObjects: Boolean)`

Visibility: public

Description: `Create` instantiates a new object list. It will simply call the inherited constructor with the correct item size and will initialize TFPGObjectList.FreeObjects (242) with `FreeObjects`.

If `FreeObjects` is true, then the list owns the objects. Freeing or clearing the list will remove all objects from memory by calling the destructor. Deleting or removing an item from the list will also dispose of the element.

See also: TFPGObjectList.FreeObjects (242)

51.10.5 TFPGObjectList.Add

Synopsis: Add new object of class `T` to the list.

Declaration: `function Add(const Item: T) : Integer`

Visibility: public

Description: `Add` adds a new item `Item` of class type `T` to the list and returns the position at which the item was added.

Errors: If the item could not be added, an EListError (219) exception is raised.

See also: TFPGObjectList.Extract (239), TFPGObjectList.Items (241), TFPGObjectList.IndexOf (239)

51.10.6 TFPGObjectList.Extract

Synopsis: Extract an item from the list

Declaration: `function Extract(const Item: T) : T`

Visibility: public

Description: `Extract` removes `Item` from the list and returns the removed item, or `Nil` if it was not found.

The extracted object will not be destroyed even if the list owns the objects.

Errors: None.

See also: TFPSList.Delete (244)

51.10.7 TFPGObjectList.GetEnumerator

Synopsis: Return a list enumerator for `T`.

Declaration: `function GetEnumerator : TFPGListEnumeratorSpec`

Visibility: public

Description: `GetEnumerator` returns an enumerator for the elements in the list. It is a specialized version of TFPGListEnumerator (227).

See also: TFPGListEnumerator (227)

51.10.8 TFPGObjectList.IndexOf

Synopsis: Index of item

Declaration: `function IndexOf(const Item: T) : Integer`

Visibility: public

Description: `IndexOf` returns the index of `Item` in the list, or -1 if the item does not appear in the list.

Errors: None.

See also: TFPGObjectList.Items (241), TFPGObjectList.Insert (240), TFPGObjectList.Add (238)

51.10.9 TFPGObjectList.Insert

Synopsis: Insert a new object in the list

Declaration: `procedure Insert(Index: Integer;const Item: T)`

Visibility: public

Description: `Insert` inserts a new object (`Item`) in the list at position `Index`. The index is zero based and must be less than Count (248).

Errors: If an invalid index is specified, an EListError (219) exception is raised.

See also: TFPGList.Items (226), TFPGList.Insert (225), TFPSList.Add (244)

51.10.10 TFPGObjectList.Assign

Synopsis: Copy objects from Source list

Declaration: `procedure Assign(Source: TFPGObjectList)`

Visibility: public

Description: `Assign` clears the list and copies all items in `Source` to the list. The source list must be of the same type as the destination list.

Take care if both the list owns the objects (i.e. have TFPGObjectList.FreeObjects (242) set to `True`), this may result to access violations.

See also: TFPSList.Clear (244), TFPGObjectList.Add (238)

51.10.11 TFPGObjectList.Remove

Synopsis: Remove an object from the list.

Declaration: `function Remove(const Item: T) : Integer`

Visibility: public

Description: `Remove` removes the object `Item` from the list, and returns the index of the removed item. If no item was removed, `-1` is returned. Only the first object is removed.

If the list owns the objects, (TFPGObjectList.FreeObjects (242) is set to `True`) then the object is freed.

Errors: None.

See also: TFPGObjectList.IndexOf (239), TFPSList.Delete (244), TFPGObjectList.FreeObjects (242)

51.10.12 TFPGObjectList.Sort

Synopsis: Sort the objects in the list

Declaration: `procedure Sort(Compare: TCompareFunc)`

Visibility: public

Description: `Sort` sorts the elements in the list using the provided `Compare` function. The list passes 2 items to the compare function. The result of this function determines how the items will be sorted:

- If the result of this function is negative, the first object (`Item1`) is assumed to be 'less' than the second object (`Item2`) and will be moved before the second in the list.
- If the function result is positive, the first object (`Item1`) is assumed to be 'greater than' the second object (`Item2`)and will be moved after the second in the list.
- if the function result is zero, the objects are assumed to be 'equal' and no moving will take place.

Errors: None.

51.10.13 TFPGObjectList.First

Synopsis: First non-nil item

Declaration: `Property First : T`

Visibility: public

Access: Read,Write

Description: `First` returns the first non-nil item. If no such element is present, `Nil` is returned.

See also: TFPSList.First (249), TFPGList.Last (226), TFPSList.Pack (247)

51.10.14 TFPGObjectList.Last

Synopsis: Last non-`Nil` object

Declaration: `Property Last : T`

Visibility: public

Access: Read,Write

Description: `Last` returns the last non-`Nil` object. If no such element is present, `Nil`. is returned.

See also: TFPGObjectList.First (241), TFPSList.Last (249)

51.10.15 TFPGObjectList.Items

Synopsis: Indexed access to objects in the list.

Declaration: `Property Items[Index: Integer]: T; default`

Visibility: public

Access: Read,Write

Description: `Items` provides indexed access to the objects in the list. The objects can be get or set.

The index `Index` is zero based, and has a maximum value of Count-1 (248).

If the list owns the objects, (TFPGObjectList.FreeObjects (242) is set to `True`) then the previous object at position `Index` is freed when setting the property.

Errors: If an invalid index is used, an EListError (219) exception is raised.

See also: TFPSList.Count (248), TFPGObjectList.FreeObjects (242)

51.10.16 TFPGObjectList.List

Synopsis: Internal list pointer

Declaration: `Property List : PTypeList`

Visibility: public

Access: Read

Description: `List` is the internal list of objects. It should not be used directly.

See also: TFPGObjectList.Items (241)

51.10.17 TFPGObjectList.FreeObjects

Synopsis: Does the list own the objects or not?

Declaration: `Property FreeObjects : Boolean`

Visibility: public

Access: Read,Write

Description: `FreeObjects` indicates whether the list owns the objects or not. If set to `True`, freeing or clearing the list will remove all objects from memory by calling the destructor. Deleting or removing an item from the list will also dispose of the element.

The initial value for this property is set in the constructor and is `True` by default.

See also: TFPGObjectList.FreeObjects (242)

51.11 TFPSList

51.11.1 Description

`TFPSList` can be seen as the generalized equivalent of the classes unit TFPList (218) list. It is used as a base class for the TFPGList (223), TFPGMap (228), TFPGObjectList (238), TFPGInterfacedObjectList (219) and TFPGMapInterfacedObjectData (233) generic classes.

This list is not meant to be used directly, it is an auxiliary class for the actual generic list classes.

See also: classes.TFPList (218), TFPGMap (228), TFPGObjectList (238), TFPGInterfacedObjectList (219), TFPGMapInterfacedObjectData (233)

51.11.2 Method overview

Page	Method	Description
244	Add	Add a new item to the list
246	Assign	Copy one list to another
244	Clear	Clear the list
243	Create	Create a new instance of `TFPSList`
244	Delete	Delete an item from the list
243	Destroy	Destroy the list instance.
244	Error	Raise an `EListError` exception.
245	Exchange	Exchange two items in the list
245	Expand	Expand the capacity of the list
245	Extract	delete an element from the list
245	IndexOf	Search an item in the list
246	Insert	Insert a new item in the list.
246	Move	Moves an item from one position in the list to another.
247	Pack	Remove empty items from the list
247	Remove	Remove the item from the list
247	Sort	Sort the list

51.11.3 Property overview

Page	Properties	Access	Description
248	Capacity	rw	Current capacity of the list
248	Count	rw	Current element count
249	First	rw	Pointer to first non-empty item in the list
248	Items	rw	Items in the list
249	ItemSize	r	Size of the items in the list
249	Last	rw	Pointer to last non-empty item in the list
249	List	r	Internal list pointer

51.11.4 TFPSList.Create

Synopsis: Create a new instance of `TFPSList`

Declaration: `constructor Create(AItemSize: Integer)`

Visibility: public

Description: `Create` creates a new instance of `TFPSList`, and initializes the item size to `ItemSize`, which defaults to the size of a pointer.

See also: TFPSList.ItemSize (249), TFPSList.Destroy (243)

51.11.5 TFPSList.Destroy

Synopsis: Destroy the list instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` clears and cleans up the list instance. Depending on the descendant, this may also remove the items in the list from memory.

See also: TFPSList.Clear (244)

51.11.6 TFPSList.Add

Synopsis: Add a new item to the list

Declaration: `function Add(Item: Pointer) : Integer`

Visibility: public

Description: `Add` adds the item pointed to by `Item` to the list. It is not the pointer `Item` itself which is added to the list, but rather the TFPSList.ItemSize (249) bytes of memory to which `Item` is pointing.

The function returns the index of the newly added item.

Errors: If the maximum list size is reached, an EListError (219) is raised.

See also: TFPSList.Delete (244), TFPSList.Items (248), TFPSList.Clear (244)

51.11.7 TFPSList.Clear

Synopsis: Clear the list

Declaration: `procedure Clear`

Visibility: public

Description: `Clear` removes all the items in the list. Depending on the descendent, the list items themselves may be cleared as well.

See also: TFPSList.Delete (244), TFPSList.Items (248), TFPSList.Add (244)

51.11.8 TFPSList.Delete

Synopsis: Delete an item from the list

Declaration: `procedure Delete(Index: Integer)`

Visibility: public

Description: `Delete` deletes the item at position `Index` from the list. Depending on the descendent, the list items itself may be cleared as well.

Errors: If `Index` is out of bounds, an EListError (219) exception is raised.

See also: TFPSList.Clear (244), TFPSList.Items (248), TFPSList.Add (244)

51.11.9 TFPSList.Error

Synopsis: Raise an `EListError` exception.

Declaration: `class procedure Error(const Msg: string;Data: PtrInt)`

Visibility: public

Description: `Error` is an auxiliary routine which raises an EListError (219) exception formatted from `Msg` and `Data`.

See also: EListError (219)

51.11.10 TFPSList.Exchange

Synopsis: Exchange two items in the list

Declaration: `procedure Exchange(Index1: Integer;Index2: Integer)`

Visibility: public

Description: `Exchange` will exchange 2 items at positions `Index1` and `Index2` in the list.

Errors: If `Index1` or `Index2` are out of bounds, an EListError (219) exception is raised.

51.11.11 TFPSList.Expand

Synopsis: Expand the capacity of the list

Declaration: `function Expand : TFPSList`

Visibility: public

Description: `Expand` will expand the capacity of the list, and returns itself.

Errors: If the size will become larger than MaxListSize (218) an EListError (219) exception will be raised.

See also: MaxListSize (218), TFPSList.Capacity (248)

51.11.12 TFPSList.Extract

Synopsis: delete an element from the list

Declaration: `procedure Extract(Item: Pointer;ResultPtr: Pointer)`

Visibility: public

Description: `Extract` removes the item pointed to by `Item` from the list. It returns a pointer to the actually removed item from the list. The item is searched using TFPSList.IndexOf (245). If the item is not found, nil is returned.

Some descendents own the items in the list. `Extract` will not dispose of the item, as TFPSList.Delete (244) does.

Errors: None.

See also: TFPSList.Delete (244), TFPSList.Add (244), TFPSList.IndexOf (245)

51.11.13 TFPSList.IndexOf

Synopsis: Search an item in the list

Declaration: `function IndexOf(Item: Pointer) : Integer`

Visibility: public

Description: `IndexOf` searches in the list for the item pointed to by `Item`, and returns the position (0-based index) of the item in the list, or -1 if it was not found. The items are compared using system.CompareMem (218), so an exact memory layout match.

See also: TFPSList.Extract (245)

51.11.14 TFPSList.Insert

Synopsis: Insert a new item in the list.

Declaration:
```
procedure Insert(Index: Integer;Item: Pointer)
function Insert(Index: Integer) : Pointer
```

Visibility: public

Description: `Insert` comes in 2 overloaded version. The version without `Item` creates a slot for a new item at position `Index` in the list, and returns a pointer to the new slot. The slot will be of size TFPSList.ItemSize (249).

The version with `Item` argument will allocate a slot in the list at position `Index` and will copy the item pointed to by `Item` to the slot in the list.

In both cases, `Index` must be 0-based.

Errors: If `Index` is invalid, an EListError (219) exception will be raised.

See also: TFPSList.Add (244), TFPSList.Delete (244), TFPSList.Extract (245)

51.11.15 TFPSList.Move

Synopsis: Moves an item from one position in the list to another.

Declaration:
```
procedure Move(CurIndex: Integer;NewIndex: Integer)
```

Visibility: public

Description: `Move` moves the item at position `CurIndex` to position `NewIndex`. This is done by storing the value at position `CurIndex`, deleting the pointer at position `CurIndex`, and reinserting the value at position `NewIndex`

Errors: If `CurIndex` or `Newindex` are not inside the valid range of indices, an EListError (219) exception is raised.

See also: Exchange (245)

51.11.16 TFPSList.Assign

Synopsis: Copy one list to another

Declaration:
```
procedure Assign(Obj: TFPSList)
```

Visibility: public

Description: `Assign` clears the list and will add all items from `Obj` to the list. The items are copied one by one.

Errors: If the TFPSList.ItemSize (249) differs for the two lists, an EListError (219) exception is raised.

See also: TFPSList.Add (244)

51.11.17 TFPSList.Remove

Synopsis: Remove the item from the list

Declaration: `function Remove(Item: Pointer) : Integer`

Visibility: public

Description: `Remove` searches `Item` in the list, and deletes it from the list. It returns the index of the item that was removed, or -1 if it was not found. Only the first match is removed.

If a descendent of `TFPSList` owns the object of the list, the item is removed from memory. If this is not desired, then TFPSList.Extract (245) must be used instead.

See also: TFPSList.Extract (245), TFPSList.IndexOf (245)

51.11.18 TFPSList.Pack

Synopsis: Remove empty items from the list

Declaration: `procedure Pack`

Visibility: public

Description: `Pack` will remove all empty items from the list. An item is considered to be empty if the memory location where the item is stored contains only zero bytes.

See also: TFPSList.Clear (244), TFPSList.Sort (247)

51.11.19 TFPSList.Sort

Synopsis: Sort the list

Declaration: `procedure Sort(Compare: TFPSListCompareFunc)`

Visibility: public

Description: `Sort>` sorts the items in the list. Two pointers are compared by passing them to the `Compare` function. The result of this function determines how the pointers will be sorted:

- If the result of this function is negative, the first item is assumed to be 'less' than the second and will be moved before the second item in the list.
- If the function result is positive, the first item is assumed to be 'greater than' the second and will be moved after the second item in the list.
- if the function result is zero, the pointers are assumed to be 'equal' and no moving will take place.

The sort is done using a quicksort algorithm.

See also: TFPSListCompareFunc (218)

51.11.20 TFPSList.Capacity

Synopsis: Current capacity of the list

Declaration: `Property Capacity : Integer`

Visibility: public

Access: Read,Write

Description: `Capacity` is the current capacity (maximum amount of elements) of the list. The list capacity will expand automatically if an item is added and the capacity is reached (i.e. TFPSList.Count (248) equals `capacity`. Expanding the list is an expensive operation involving reallocation of memory and moving of list data in memort, so capacity can be set to a large amount to avoid frequent reallocations.

See also: TFPSList.Count (248), TFPSList.Expand (245), TFPSList.Items (248)

51.11.21 TFPSList.Count

Synopsis: Current element count

Declaration: `Property Count : Integer`

Visibility: public

Access: Read,Write

Description: `Count` is the current amount of elements in the list. It is initially zero. A valid item index is always a value between zero and `Count-1`.

See also: TFPSList.Items (248), TFPSList.Capacity (248)

51.11.22 TFPSList.Items

Synopsis: Items in the list

Declaration: `Property Items[Index: Integer]: Pointer; default`

Visibility: public

Access: Read,Write

Description: `Items` provides indexed access to the items in the list, the returned pointers are not the actual list items, but pointers to the elements in the list. The items can be get or set.

When assigning to the `Items` property, the memory area pointed to by the assigned pointer is copied to the list. Exactly TFPSList.ItemSize (249) bytes are copied.

The index `Index` is zero based, and has a maximum value of Count-1 (248).

Errors: If an invalid index is used, an EListError (219) exception is raised.

See also: TFPSList.ItemSize (249), TFPSList.Count (248)

51.11.23 TFPSList.ItemSize

Synopsis: Size of the items in the list

Declaration: `Property ItemSize : Integer`

Visibility: public

Access: Read

Description: `ItemSize` is the memory size of the items in the list. It is specified in the constructor and cannot be changed during the lifetime of the list.

See also: TFPSList.Create (243)

51.11.24 TFPSList.List

Synopsis: Internal list pointer

Declaration: `Property List : PByte`

Visibility: public

Access: Read

Description: `List` is a pointer to the memory area used for the items in the list. It should not be manipulated.

51.11.25 TFPSList.First

Synopsis: Pointer to first non-empty item in the list

Declaration: `Property First : Pointer`

Visibility: public

Access: Read,Write

Description: `First` returns the value of the first non-empty item in the list. An item is considered empty if consists of TFPSList.ItemSize (249) zero bytes.

If there are no non-empty items in the list, then `Nil` is returned.

See also: TFPSList.Last (249), TFPSList.Pack (247)

51.11.26 TFPSList.Last

Synopsis: Pointer to last non-empty item in the list

Declaration: `Property Last : Pointer`

Visibility: public

Access: Read,Write

Description: `Last` returns the value of the last non-empty item in the list. An item is considered empty if consists of TFPSList.ItemSize (249) zero bytes.

If there are no non-empty items in the list, then `Nil` is returned.

See also: TFPSList.Last (249), TFPSList.Pack (247)

51.12 TFPSMap

51.12.1 Description

`TFPSMap` can be used to create a map for any type `T` that does not require reference counting (such as interfaced objects). It will specialize to a map which is a generalized list with an arbitrary type as the list index (called the key).

This class should normally not be used directly, instead use one of the generic map objects such as TFPGMap (228).

See also: TFPGMap (228)

51.12.2 Method overview

Page	Method	Description
251	Add	Add a key, value pair to the map.
250	Create	Create a new map with given key and data size.
251	Find	Find data using the associated key
251	IndexOf	Index of key pointed to by `AKey`
252	IndexOfData	Index of data item `AData`
252	Insert	Insert a new slot for key and associated data item in the list
252	InsertKey	Insert a key in the list
252	InsertKeyData	Insert a key and associated in the list
253	Remove	Remove a key/value pair from the map.
253	Sort	Sort the list according to key

51.12.3 Property overview

Page	Properties	Access	Description
254	Data	rw	Indexed access to the locations of all data items
254	DataSize	r	Size (in bytes) for the data associated with keys
253	Duplicates	rw	What to do with duplicate key values
255	KeyData	rw	Access to data locations using key
254	Keys	rw	Indexed access to the locations of all keys
254	KeySize	r	Size (in bytes) for the key
256	OnDataPtrCompare	rw	Callback to compare 2 data items
255	OnKeyPtrCompare	rw	Callback to compare 2 keys
255	OnPtrCompare	rw	Alias for `OnKeyPtrCompare`
255	Sorted	rw	Is the map permanently sorted on key ?

51.12.4 TFPSMap.Create

Synopsis: Create a new map with given key and data size.

Declaration: `constructor Create(AKeySize: Integer;ADataSize: Integer)`

Visibility: public

Description: `Create` instantiates a new `TFPSMap` instance and initializes TFPSMap.KeySize (254) and TFPSMap.DataSize (254) with `AKeySize` and `ADataSize`, respectively. It also initializes the TFPSMap.OnDataPtrCompare (256) and TFPSMap.OnKeyPtrCompare (255) properties to functions that compare memory blocks.

Errors: None.

See also: TFPSMap.Destroy (250), TFPSMap.KeySize (254), TFPSMap.DataSize (254), TFPSMap.OnDataPtrCompare (256), TFPSMap.OnKeyPtrCompare (255)

51.12.5 TFPSMap.Add

Synopsis: Add a key, value pair to the map.

Declaration:
```
function Add(AKey: Pointer;AData: Pointer) : Integer
function Add(AKey: Pointer) : Integer
```

Visibility: public

Description: `Add` adds the memory pointed to by `AData` to the map using the memory pointed to by `AKey` as the key. If no data is specified, it allocates a slot fot `AKey` and returns a pointer to that slot.

Errors: If the maximum amount of values is reached, `Add` will raise an EListError (219) exception.

See also: TFPSMap.Insert (252), TFPSMap.IndexOf (251)

51.12.6 TFPSMap.Find

Synopsis: Find data using the associated key

Declaration:
```
function Find(AKey: Pointer;out Index: Integer) : Boolean
```

Visibility: public

Description: `Find` searches for the first key less than or equal to `AKey` and returns its index in the list in `Index`. It returns `True` if an exact match for the key was found. It returns `False` if the key was not found, and `Index` is then set to `-1`

This function performs a binary search using the key comparing function specified in OnKeyPtrCompare (255).

Errors: if `OnKeyPtrCompare` is not set, an access violation will occur.

See also: TFPSMap.OnKeyPtrCompare (255)

51.12.7 TFPSMap.IndexOf

Synopsis: Index of key pointed to by `AKey`

Declaration:
```
function IndexOf(AKey: Pointer) : Integer
```

Visibility: public

Description: `IndexOf` returns the index of the element with a key pointed to by `AKey`. It returns -1 if the key was not found.

If the list is sorted, then a binary search is performed, otherwise a linear search is used to find the key.

Errors: None.

See also: TFPSMap.Find (251), TFPSMap.IndexOfData (252), TFPSMap.Keys (254), TFPSMap.Data (254)

51.12.8 TFPSMap.IndexOfData

Synopsis: Index of data item `AData`

Declaration:
```
function IndexOfData(AData: Pointer) : Integer
```

Visibility: public

Description: `IndexOfData` returns the index of the element with data pointed to by `AData`. It returns -1 if the data item was not found. The search is always performed using a linear search.

See also: TFPSMap.Find (251), TFPSMap.IndexOf (251), TFPSMap.Keys (254), TFPSMap.Data (254)

51.12.9 TFPSMap.Insert

Synopsis: Insert a new slot for key and associated data item in the list

Declaration:
```
function Insert(Index: Integer) : Pointer
procedure Insert(Index: Integer;out AKey: Pointer;out AData: Pointer)
```

Visibility: public

Description: `Insert` will allocate a new slot in the list. It returns a pointer to the new slot. If `Akey` and `AData` are given, then they will point to the positions in the slot for the key and data items.

Errors: If the maximum amount of values is reached or an invalid index is specified, `Insert` will raise an EListError (219) exception.

See also: TFPSMap.Add (251), TFPSMap.InsertKey (252), TFPSMap.InsertKeyData (252)

51.12.10 TFPSMap.InsertKey

Synopsis: Insert a key in the list

Declaration:
```
procedure InsertKey(Index: Integer;AKey: Pointer)
```

Visibility: public

Description: `InsertKey` will allocate a new slot in the list for a key value as pointed to by `AKey`, and copy the key pointed to by `AKey` to the slot.

Errors: If the maximum amount of values is reached or an invalid index is specified, `InsertKey` will raise an EListError (219) exception.

See also: TFPSMap.Add (251), TFPSMap.Insert (252), TFPSMap.InsertKeyData (252)

51.12.11 TFPSMap.InsertKeyData

Synopsis: Insert a key and associated in the list

Declaration:
```
procedure InsertKeyData(Index: Integer;AKey: Pointer;AData: Pointer)
```

Visibility: public

Description: `InsertKeyData` will allocate a new slot in the list for a key value as pointed to by `AKey`, and copy the key pointed to by `AKey` as well as the data pointed to by `AData` to the newly allocated slot.

Errors: If the maximum amount of values is reached or an invalid index is specified, `InsertKeyData` will raise an EListError (219) exception.

See also: TFPSMap.Add (251), TFPSMap.Insert (252), TFPSMap.InsertKey (252)

51.12.12 TFPSMap.Remove

Synopsis: Remove a key/value pair from the map.

Declaration: `function Remove(AKey: Pointer) : Integer`

Visibility: public

Description: `Remove` removes the key/value pair pointing to by `AKey` from the map. It returns the index of `Akey` prior to removal from the list. If `AKey` was not found, -1 is returned.

Errors: None.

See also: TFPSList.Delete (244), TFPSMap.IndexOf (251)

51.12.13 TFPSMap.Sort

Synopsis: Sort the list according to key

Declaration: `procedure Sort`

Visibility: public

Description: `Sort` sorts the list according to the key value, using the compare function provided in TFPSMap.OnKeyPtrCompare (255)

Errors: None.

See also: TFPSMap.OnKeyPtrCompare (255), TFPSMap.OnDataPtrCompare (256)

51.12.14 TFPSMap.Duplicates

Synopsis: What to do with duplicate key values

Declaration: `Property Duplicates : TDuplicates`

Visibility: public

Access: Read,Write

Description: `Duplicates` can be set to determine what to do with duplicate key values in the map:

dupIgnoreIgnore the new item, do not add it to the list.

dupAcceptAccept duplicates, adding them to the list.

dupErrorRaise an error when an attempt is made to add a duplicate.

The value is ignored if Sorted (255) is `False`.

See also: TFPSMap.Sorted (255), types.TDuplicates (218)

51.12.15 TFPSMap.KeySize

Synopsis: Size (in bytes) for the key

Declaration: `Property KeySize : Integer`

Visibility: public

Access: Read

Description: `KeySize` is the size (in bytes) for the keys in the map. This size is initialized during list construction and defaults to the size of a pointer.

See also: TFPSMap.Create (250), TFPSMap.DataSize (254), TFPSMap.Keys (254), TFPSMap.KeyData (255)

51.12.16 TFPSMap.DataSize

Synopsis: Size (in bytes) for the data associated with keys

Declaration: `Property DataSize : Integer`

Visibility: public

Access: Read

Description: `DataSize` is the size (in bytes) for the data in the map. This size is initialized during list construction and defaults to the size of a pointer.

See also: TFPSMap.Create (250), TFPSMap.KeySize (254), TFPSMap.Keys (254), TFPSMap.KeyData (255), TFPSMap.Data (254)

51.12.17 TFPSMap.Keys

Synopsis: Indexed access to the locations of all keys

Declaration: `Property Keys[Index: Integer]: Pointer`

Visibility: public

Access: Read,Write

Description: `Keys` provides indexed access to all keys. It does not return the key, but returns a pointer to the key value. When setting the `Keys` property, a pointer to the key value must be set, and the key value is copied from the pointer.

See also: TFPSMap.KeySize (254), TFPSMap.KeyData (255), TFPSMap.Data (254)

51.12.18 TFPSMap.Data

Synopsis: Indexed access to the locations of all data items

Declaration: `Property Data[Index: Integer]: Pointer`

Visibility: public

Access: Read,Write

Description: `Data` provides indexed access to all data. It does not return the actual data, but returns a pointer to the data. When setting the `Data` property, a pointer to the data value must be set, and the data value is copied from the pointer.

See also: TFPSMap.DataSize (254), TFPSMap.KeyData (255), TFPSMap.Keys (254)

51.12.19 TFPSMap.KeyData

Synopsis: Access to data locations using key

Declaration: `Property KeyData[Key: Pointer]: Pointer; default`

Visibility: public

Access: Read,Write

Description: `KeyData` provides access to the data items, using their key value (as pointed to by `AKey`) as an index. When reading a non-existent key value, `Nil` is returned. If the key is found, a pointer to the associated data's location is returned. When writing, the key pointed to by `Key` is added if it was not present, and the data data is copied from the written pointer.

See also: TFPSMap.DataSize (254), TFPSMap.KeyData (255), TFPSMap.Data (254), TFPSMap.Keys (254), TFPSMap.KeySize (254)

51.12.20 TFPSMap.Sorted

Synopsis: Is the map permanently sorted on key ?

Declaration: `Property Sorted : Boolean`

Visibility: public

Access: Read,Write

Description: `Sorted` can be set to true to keep the map permanently sorted on key value. The sorting happens using TFPSMap.OnKeyPtrCompare (255).

See also: TFPSMap.OnKeyPtrCompare (255), TFPSMap.Sort (253), TFPSMap.Duplicates (253)

51.12.21 TFPSMap.OnPtrCompare

Synopsis: Alias for `OnKeyPtrCompare`

Declaration: `Property OnPtrCompare : TFPSListCompareFunc`

Visibility: public

Access: Read,Write

Description: `OnPtrCompare` is a deprecated alias for OnKeyPtrCompare (255).

See also: TFPSMap.OnKeyPtrCompare (255), TFPSMap.OnDataPtrCompare (256)

51.12.22 TFPSMap.OnKeyPtrCompare

Synopsis: Callback to compare 2 keys

Declaration: `Property OnKeyPtrCompare : TFPSListCompareFunc`

Visibility: public

Access: Read,Write

Description: `OnKeyPtrCompare` is used to compare the values of 2 keys. By default it simply compares the byte values of the key memory block. It can be set to any function that performs another comparison. (e.g. a function that treats the memory blocks as a string pointer and compare the actual strings).

This function is used to sort the list or find a key.

See also: TFPSListCompareFunc (218), TFPSMap.OnDataPtrCompare (256), TFPSMap.Sort (253), TFPSMap.Find (251), TFPSMap.IndexOf (251)

51.12.23 TFPSMap.OnDataPtrCompare

Synopsis: Callback to compare 2 data items

Declaration: `Property OnDataPtrCompare : TFPSListCompareFunc`

Visibility: public

Access: Read,Write

Description: `OnKeyPtrCompare` is used to compare the values of 2 keys. By default it simply compares the byte values of the key memory block. It can be set to any function that performs another comparison. (e.g. a function that treats the memory blocks as a string pointer and compare the actual strings).

This function is used to find a data item (IndexOf (252)).

See also: TFPSListCompareFunc (218), TFPSMap.OnKeyPtrCompare (255), TFPSMap.IndexOfData (252)

Chapter 52

Reference for unit 'fpwidestring'

52.1 Used units

Table 52.1: Used units by unit 'fpwidestring'

Name	Page
System	622
unicodedata	702

52.2 Overview

`fpwidestring` implements unicode string support for the Free Pascal RTL using native Object Pascal routines. It is meant to be used on operating systems where the operating system does not natively support Unicode transformations and operations.

In general, it is sufficient to include the unit in the uses clause of a program. The initialization code of the unit will set the unicode string manager of the system unit to the object pascal implementation contained in this unit.

This unit needs unicode collation and character set tables in order to be able to do its work correctly. These must be registered using the routines of the unicodedata (257) unit: the FPC project distributes some unicode collation data in `.bco` files which can be loaded using the LoadCollation (711) routine.

In order for sorting and comparing of strings to work, a collation must be used. The collation in general depends on the internationalization of the application. Since the system unit does not know about collations, the collation must be set in the `fpWideString` unit using the SetActiveCollation (257) function. The collation can be set on a per-thread basis.

New threads get DefaultCollationName (257) as the active collation name.

The fpwidestring unit performs conversions between unicode and single-byte ansistring conversions (excluding UTF8). Support for various single-byte encodings are based on the charset (257) unit. This unit can be used to load single byte code pages. Various code page units such as cp895, cp932, cp950 are provided by the "rtl-unicode" package.

The fpwidestring requires at least the Default Unicode Collation Element Table to be registered (called `DUCET`). The DUCET encoding is provided by the unicodeducet unit. More information can be found in the unicodedata (257) unit.

Chapter 53

Reference for unit 'getopts'

53.1 Overview

This document describes the GETOPTS unit for Free Pascal. It was written for linux by Michael Van Canneyt. It now also works for all supported platforms.

The getopts unit provides a mechanism to handle command-line options in a structured way, much like the GNU getopts mechanism. It allows you to define the valid options for you program, and the unit will then parse the command-line options for you, and inform you of any errors.

53.2 Constants, types and variables

53.2.1 Constants

```
EndOfOptions = #255
```

Returned by getopt (260),getlongopts (260) to indicate that there are no more options.

```
No_Argument = 0
```

Specifies that a long option does not take an argument.

```
Optional_Argument = 2
```

Specifies that a long option optionally takes an argument.

```
OptSpecifier : Set of Char = ['-']
```

Character indicating an option on the command-line.

```
Required_Argument = 1
```

Specifies that a long option needs an argument.

53.2.2 Types

```
Orderings = (require_order,permute,return_in_order)
```

Table 53.1: Enumeration values for type Orderings

Value	Explanation
permute	Change command-line options.
require_order	Don't touch the ordering of the command-line options
return_in_order	Return options in the correct order.

Command-line ordering options.

```
POption = ^TOption
```

Pointer to TOption (259) record.

```
TOption = record
  Name : string;
  Has_arg : Integer;
  Flag : PChar;
  Value : Char;
end
```

The `TOption` type is used to communicate the long options to GetLongOpts (260). The `Name` field is the name of the option. `Has_arg` specifies if the option wants an argument, `Flag` is a pointer to a `char`, which is set to `Value`, if it is non-`nil`.

53.2.3 Variables

```
OptArg : string
```

Set to the argument of an option, if the option needs one.

```
OptErr : Boolean
```

Indicates whether `getopt()` prints error messages.

```
OptInd : LongInt
```

when all options have been processed, `optind` is the index of the first non-option parameter. This is a read-only variable. Note that it can become equal to `paramcount+1`.

```
OptOpt : Char
```

In case of an error, contains the character causing the error.

53.3 Procedures and functions

53.3.1 GetLongOpts

Synopsis: Return next long option.

Declaration:
```
function GetLongOpts(ShortOpts: string;LongOpts: POption;
                     var Longind: LongInt) : Char
```

Visibility: default

Description: Returns the next option found on the command-line, taking into account long options as well. If no more options are found, returns `EndOfOptions`. If the option requires an argument, it is returned in the `OptArg` variable.

`ShortOptions` is a string containing all possible one-letter options. (see Getopt (260) for its description and use) `LongOpts` is a pointer to the first element of an array of `Option` records, the last of which needs a name of zero length.

The function tries to match the names even partially (i.e. `-app` will match e.g. the `append` option), but will report an error in case of ambiguity. If the option needs an argument, set `Has_arg` to `Required_argument`, if the option optionally has an argument, set `Has_arg` to `Optional_argument`. If the option needs no argument, set `Has_arg` to zero.

Required arguments can be specified in two ways :

1. Pasted to the option : `-option=value`
2. As a separate argument : `-option value`

Optional arguments can only be specified through the first method.

Errors: see Getopt (260).

See also: Getopt (260)

53.3.2 GetOpt

Synopsis: Get next short option.

Declaration:
```
function GetOpt(ShortOpts: string) : Char
```

Visibility: default

Description: Returns the next option found on the command-line. If no more options are found, returns `EndOfOptions`. If the option requires an argument, it is returned in the `OptArg` variable.

`ShortOptions` is a string containing all possible one-letter options. If a letter is followed by a colon (:), then that option needs an argument. If a letter is followed by 2 colons, the option has an optional argument. If the first character of `shortoptions` is a `'+'` then options following a non-option are regarded as non-options (standard Unix behavior). If it is a `'-'`, then all non-options are treated as arguments of a option with character `#0`. This is useful for applications that require their options in the exact order as they appear on the command-line. If the first character of `shortoptions` is none of the above, options and non-options are permuted, so all non-options are behind all options. This allows options and non-options to be in random order on the command line.

Errors: Errors are reported through giving back a `'?'` character. `OptOpt` then gives the character which caused the error. If `OptErr` is `True` then getopt prints an error-message to `stdout`.

See also: GetLongOpts (260)

Listing: ./optex/optex.pp

```
program testopt;

{ Program to depmonstrate the getopts function. }

{
  Valid calls to this program are
  optex --verbose --add me --delete you
  optex --append --create child
  optex -ab -c me -d you
  and so on
}
uses getopts;

var c : char;
    optionindex : Longint;
    theopts : array[1..7] of TOption;

begin
  with theopts[1] do
   begin
    name:='add';
    has_arg:=1;
    flag:=nil;
    value:=#0;
  end;
  with theopts[2] do
   begin
    name:='append';
    has_arg:=0;
    flag:=nil;
    value:=#0;
  end;
  with theopts[3] do
   begin
    name:='delete';
    has_arg:=1;
    flag:=nil;
    value:=#0;
  end;
  with theopts[4] do
   begin
    name:='verbose';
    has_arg:=0;
    flag:=nil;
    value:=#0;
  end;
  with theopts[5] do
   begin
    name:='create';
    has_arg:=1;
    flag:=nil;
    value:='c'
  end;
  with theopts[6] do
   begin
    name:='file';
    has_arg:=1;
```

```
    flag:=nil;
    value:=#0;
  end;
  with theopts[7] do
   begin
    name:='';
    has_arg:=0;
    flag:=nil;
  end;
  c:=#0;
  repeat
    c:=getlongopts('abc:d:012',@theopts[1],optionindex);
    case c of
      '1','2','3','4','5','6','7','8','9' :
        begin
        writeln ('Got optind : ',c)
        end;
      #0 : begin
           write ('Long option : ',theopts[optionindex].name);
           if theopts[optionindex].has_arg>0 then
             writeln (' With value  : ',optarg)
           else
             writeln
           end;
      'a' : writeln ('Option a.');
      'b' : writeln ('Option b.');
      'c' : writeln ('Option c : ', optarg);
      'd' : writeln ('Option d : ', optarg);
      '?',':' : writeln ('Error with opt : ',optopt);
   end; { case }
 until c=endofoptions;
 if optind<=paramcount then
    begin
    write ('Non options : ');
    while optind<=paramcount do
      begin
      write (paramstr(optind),' ');
      inc(optind)
      end;
    writeln
    end
end.
```

Chapter 54

Reference for unit 'go32'

54.1 Overview

This document describes the GO32 unit for the Free Pascal compiler under dos. It was donated by Thomas Schatzl (tom_at_work@geocities.com), for which my thanks. This unit was first written for dos by Florian Klaempfl.

Only the GO32V2 DPMI mode is discussed by me here due to the fact that new applications shouldn't be created with the older GO32V1 model. The go32v2 version is much more advanced and better. Additionally a lot of functions only work in DPMI mode anyway. I hope the following explanations and introductions aren't too confusing at all. If you notice an error or bug send it to the FPC mailing list or directly to me. So let's get started and happy and error free coding I wish you.... Thomas Schatzl, 25. August 1998

54.2 Real mode callbacks

The callback mechanism can be thought of as the converse of calling a real mode procedure (i.e. interrupt), which allows your program to pass information to a real mode program, or obtain services from it in a manner that's transparent to the real mode program. In order to make a real mode callback available, you must first get the real mode callback address of your procedure and the selector and offset of a register data structure. This real mode callback address (this is a segment:offset address) can be passed to a real mode program via a software interrupt, a dos memory block or any other convenient mechanism. When the real mode program calls the callback (via a far call), the DPMI host saves the registers contents in the supplied register data structure, switches into protected mode, and enters the callback routine with the following settings:

- interrupts disabled
- `%CS:%EIP` = 48 bit pointer specified in the original call to get_rm_callback (287)
- `%DS:%ESI` = 48 bit pointer to real mode `SS:SP`
- `%ES:%EDI` = 48 bit pointer of real mode register data structure.
- `%SS:%ESP` = locked protected mode stack
- All other registers undefined

The callback procedure can then extract its parameters from the real mode register data structure and/or copy parameters from the real mode stack to the protected mode stack. Recall that the segment

register fields of the real mode register data structure contain segment or paragraph addresses that are not valid in protected mode. Far pointers passed in the real mode register data structure must be translated to virtual addresses before they can be used with a protected mode program. The callback procedure exits by executing an IRET with the address of the real mode register data structure in %ES:%EDI, passing information back to the real mode caller by modifying the contents of the real mode register data structure and/or manipulating the contents of the real mode stack. The callback procedure is responsible for setting the proper address for resumption of real mode execution into the real mode register data structure; typically, this is accomplished by extracting the return address from the real mode stack and placing it into the %CS:%EIP fields of the real mode register data structure. After the IRET, the DPMI host switches the CPU back into real mode, loads ALL registers with the contents of the real mode register data structure, and finally returns control to the real mode program. All variables and code touched by the callback procedure MUST be locked to prevent page faults.

See also: get_rm_callback (287), free_rm_callback (282), lock_code (295), lock_data (295)

54.3 Executing software interrupts

Simply execute a realintr() call with the desired interrupt number and the supplied register data structure. But some of these interrupts require you to supply them a pointer to a buffer where they can store data to or obtain data from in memory. These interrupts are real mode functions and so they only can access the first Mb of linear address space, not FPC's data segment. For this reason FPC supplies a pre-initialized dos memory location within the GO32 unit. This buffer is internally used for dos functions too and so it's contents may change when calling other procedures. It's size can be obtained with tb_size (305) and it's linear address via transfer_buffer (305). Another way is to allocate a completely new dos memory area via the global_dos_alloc (292) function for your use and supply its real mode address.

See also: tb_size (305), transfer_buffer (305), global_dos_alloc (292), global_dos_free (293), realintr (298)

Listing: ./go32ex/softint.pp

```
uses
        go32;

var
        r : trealregs;

begin
        r.ah := $30;
        r.al := $01;
        realintr($21, r);
        Writeln('DOS v', r.al,'.',r.ah, ' detected');
end.
```

Listing: ./go32ex/rmpmint.pp

```
uses
        crt,
        go32;

var
        r : trealregs;
        axreg : Word;

        oldint21h : tseginfo;
```

```
        newint21h : tseginfo;
procedure int21h_handler; assembler;
asm
        cmpw $0x3001, %ax
        jne .LCallOld
        movw $0x3112, %ax
        iret

.LCallOld:
        ljmp %cs:oldint21h
end;

procedure resume;
begin
        Writeln;
        Write('-- press any key to resume --'); readkey;
        gotoxy(1, wherey); clreol;
end;

begin
        clrscr;
        Writeln('Executing real mode interrupt');
        resume;
        r.ah := $30; r.al := $01;  realintr($21, r);
        Writeln('DOS v', r.al,'.',r.ah, ' detected');
        resume;
        Writeln('Executing protected mode interrupt without our own',
                ' handler');
        Writeln;
        asm
                movb $0x30, %ah
                movb $0x01, %al
                int $0x21
                movw %ax, axreg
        end;
        Writeln('DOS v', r.al,'.',r.ah, ' detected');
        resume;
        Writeln('As you can see the DPMI hosts default protected mode',
                'handler');
        Writeln('simply redirects it to the real mode handler');
        resume;
        Writeln('Now exchanging the protected mode interrupt with our ',
                'own handler');
        resume;

        newint21h.offset := @int21h_handler;
        newint21h.segment := get_cs;
        get_pm_interrupt($21, oldint21h);
        set_pm_interrupt($21, newint21h);

        Writeln('Executing real mode interrupt again');
        resume;
        r.ah := $30; r.al := $01; realintr($21, r);
        Writeln('DOS v', r.al,'.',r.ah, ' detected');
        Writeln;
        Writeln('See, it didn''t change in any way.');
        resume;
        Writeln('Now calling protected mode interrupt');
```

```
        resume;
        asm
                movb $0x30, %ah
                movb $0x01, %al
                int $0x21
                movw %ax, axreg
        end;
        Writeln('DOS v', lo(axreg),'.',hi(axreg), ' detected');
        Writeln;
        Writeln('Now you can see that there''s a distinction between ',
                'the two ways of calling interrupts...');
        set_pm_interrupt($21, oldint21h);
end.
```

54.4 Software interrupts

Ordinarily, a handler installed with set_pm_interrupt (302) only services software interrupts that are executed in protected mode; real mode software interrupts can be redirected by set_rm_interrupt (303).

See also: set_rm_interrupt (303), get_rm_interrupt (290), set_pm_interrupt (302), get_pm_interrupt (286), lock_data (295), lock_code (295), enable (281), disable (279), outportb (296)

54.5 Hardware interrupts

Hardware interrupts are generated by hardware devices when something unusual happens; this could be a keypress or a mouse move or any other action. This is done to minimize CPU time, else the CPU would have to check all installed hardware for data in a big loop (this method is called 'polling') and this would take much time. A standard IBM-PC has two interrupt controllers, that are responsible for these hardware interrupts: both allow up to 8 different interrupt sources (IRQs, interrupt requests). The second controller is connected to the first through IRQ 2 for compatibility reasons, e.g. if controller 1 gets an IRQ 2, he hands the IRQ over to controller 2. Because of this up to 15 different hardware interrupt sources can be handled. IRQ 0 through IRQ 7 are mapped to interrupts 8h to Fh and the second controller (IRQ 8 to 15) is mapped to interrupt 70h to 77h. All of the code and data touched by these handlers MUST be locked (via the various locking functions) to avoid page faults at interrupt time. Because hardware interrupts are called (as in real mode) with interrupts disabled, the handler has to enable them before it returns to normal program execution. Additionally a hardware interrupt must send an EOI (end of interrupt) command to the responsible controller; this is acomplished by sending the value 20h to port 20h (for the first controller) or A0h (for the second controller). The following example shows how to redirect the keyboard interrupt.

Listing: ./go32ex/keyclick.pp

```
{$ASMMODE ATT}
{$MODE FPC}

uses
        crt,
        go32;

const
        kbdint = $9;
```

```
var
        oldint9_handler : tseginfo;
        newint9_handler : tseginfo;

        clickproc : pointer;
        backupDS : Word; external name '___v2prt0_ds_alias';

procedure int9_handler; assembler;
asm
        cli
        pushl %ds
        pushl %es
        pushl %fs
        pushl %gs
        pushal
        movw %cs:backupDS, %ax
        movw %ax, %ds
        movw %ax, %es
        movw dosmemselector, %ax
        movw %ax, %fs
        call *clickproc
        popal
        popl %gs
        popl %fs
        popl %es
        popl %ds
        ljmp %cs:oldint9_handler
end;
procedure int9_dummy; begin end;

procedure clicker;
begin
        sound(500); delay(10); nosound;
end;
procedure clicker_dummy; begin end;

procedure install_click;
begin
        clickproc := @clicker;
        lock_data(clickproc, sizeof(clickproc));
        lock_data(dosmemselector, sizeof(dosmemselector));

        lock_code(@clicker,
                longint(@clicker_dummy) - longint(@clicker));
        lock_code(@int9_handler,
                longint(@int9_dummy)-longint(@int9_handler));
        newint9_handler.offset := @int9_handler;
        newint9_handler.segment := get_cs;
        get_pm_interrupt(kbdint, oldint9_handler);
        set_pm_interrupt(kbdint, newint9_handler);
end;

procedure remove_click;
begin
        set_pm_interrupt(kbdint, oldint9_handler);
        unlock_data(dosmemselector, sizeof(dosmemselector));
        unlock_data(clickproc, sizeof(clickproc));
```

```
        unlock_code(@clicker,
                longint(@clicker_dummy)-longint(@clicker));
        unlock_code(@int9_handler,
                longint(@int9_dummy)-longint(@int9_handler));
end;

var
        ch : char;

begin
        install_click;
        Writeln('Enter any message. Press return when finished');
        while (ch <> #13) do begin
                ch := readkey; write(ch);
        end;
        remove_click;
end.
```

54.6 Disabling interrupts

The GO32 unit provides the two procedures disable() and enable() to disable and enable all interrupts.

54.7 Creating your own interrupt handlers

Interrupt redirection with FPC pascal is done via the set_pm_interrupt() for protected mode interrupts or via the set_rm_interrupt() for real mode interrupts.

54.8 Protected mode interrupts vs. Real mode interrupts

As mentioned before, there's a distinction between real mode interrupts and protected mode interrupts; the latter are protected mode programs, while the former must be real mode programs. To call a protected mode interrupt handler, an assembly 'int' call must be issued, while the other is called via the realintr() or intr() function. Consequently, a real mode interrupt then must either reside in dos memory (<1MB) or the application must allocate a real mode callback address via the get_rm_callback() function.

54.9 Handling interrupts with DPMI

The interrupt functions are real-mode procedures; they normally can't be called in protected mode without the risk of an protection fault. So the DPMI host creates an interrupt descriptor table for the application. Initially all software interrupts (except for int 31h, 2Fh and 21h function 4Ch) or external hardware interrupts are simply directed to a handler that reflects the interrupt in real-mode, i.e. the DPMI host's default handlers switch the CPU to real-mode, issue the interrupt and switch back to protected mode. The contents of general registers and flags are passed to the real mode handler and the modified registers and flags are returned to the protected mode handler. Segment registers and stack pointer are not passed between modes.

54.10 Interrupt redirection

Interrupts are program interruption requests, which in one or another way get to the processor; there's a distinction between software and hardware interrupts. The former are explicitely called by an 'int' instruction and are a bit comparable to normal functions. Hardware interrupts come from external devices like the keyboard or mouse. Functions that handle hardware interrupts are called handlers.

54.11 Processor access

These are some functions to access various segment registers (%cs, %ds, %ss) which makes your work a bit easier.

See also: get_cs (282), get_ds (283), get_ss (292)

54.12 I/O port access

The I/O port access is done via the various inportb (294), outportb (296) functions which are available. Additionally Free Pascal supports the Turbo Pascal PORT[]-arrays but it is by no means recommened to use them, because they're only for compatibility purposes.

See also: outportb (296), inportb (294)

54.13 dos memory access

Dos memory is accessed by the predefined `dosmemselector` selector; the GO32 unit additionally provides some functions to help you with standard tasks, like copying memory from heap to dos memory and the likes. Because of this it is strongly recommened to use them, but you are still free to use the provided standard memory accessing functions which use 48 bit pointers. The third, but only thought for compatibility purposes, is using the `mem[]`-arrays. These arrays map the whole 1 Mb dos space. They shouldn't be used within new programs. To convert a segment:offset real mode address to a protected mode linear address you have to multiply the segment by 16 and add its offset. This linear address can be used in combination with the DOSMEMSELECTOR variable.

See also: dosmemget (271), dosmemput (271), dosmemmove (271), dosmemfillchar (270), dosmemfillword (271), seg_move (300), seg_fillchar (299), seg_fillword (300)

54.14 FPC specialities

The %ds and %es selector MUST always contain the same value or some system routines may crash when called. The %fs selector is preloaded with the DOSMEMSELECTOR variable at startup, and it MUST be restored after use, because again FPC relys on this for some functions. Luckily we asm programmers can still use the %gs selector for our own purposes, but for how long ?

See also: get_cs (282), get_ds (283), get_ss (292), allocate_ldt_descriptors (276), free_ldt_descriptor (281), segment_to_descriptor (299), get_next_selector_increment_value (285), get_segment_base_address (291), set_segment_base_address (303), set_segment_limit (304), create_code_segment_alias_descriptor (279)

54.15 Selectors and descriptors

Descriptors are a bit like real mode segments; they describe (as the name implies) a memory area in protected mode. A descriptor contains information about segment length, its base address and the attributes of it (i.e. type, access rights, ...). These descriptors are stored internally in a so-called descriptor table, which is basically an array of such descriptors. Selectors are roughly an index into this table. Because these 'segments' can be up to 4 GB in size, 32 bits aren't sufficient anymore to describe a single memory location like in real mode. 48 bits are now needed to do this, a 32 bit address and a 16 bit sized selector. The GO32 unit provides the tseginfo record to store such a pointer. But due to the fact that most of the time data is stored and accessed in the %ds selector, FPC assumes that all pointers point to a memory location of this selector. So a single pointer is still only 32 bits in size. This value represents the offset from the data segment base address to this memory location.

54.16 What is DPMI

The dos Protected Mode Interface helps you with various aspects of protected mode programming. These are roughly divided into descriptor handling, access to dos memory, management of interrupts and exceptions, calls to real mode functions and other stuff. Additionally it automatically provides swapping to disk for memory intensive applications. A DPMI host (either a Windows dos box or CWSDPMI.EXE) provides these functions for your programs.

54.17 Constants, types and variables

54.17.1 Constants

```
auxcarryflag = $010
```

Check for auxiliary carry flag in trealregs (275)

```
carryflag = $001
```

Check for carry flag in trealregs (275)

```
directionflag = $400
```

Check for direction flag in trealregs (275)

```
dosmemfillchar : procedure(seg: Word;ofs: Word;count: LongInt;c: Char) = @dpmi_dosmem
```

Sets a region of dos memory to a specific byte value.

Parameters:

seg real mode segment.

ofs real mode offset.

count number of bytes to set.

c value to set memory to.

Notes: No range check is performed.

```
dosmemfillword : procedure(seg: Word;ofs: Word;count: LongInt;w: Word) = @dpmi_dosmem
```

Sets a region of dos memory to a specific word value.
Parameters:

seg real mode segment.

ofs real mode offset.

count number of words to set.

w value to set memory to.

Notes: No range check is performed.

```
dosmemget : procedure(seg: Word;ofs: Word;var data;count: LongInt) = @dpmi_dosmemget
```

Copies data from the dos memory onto the heap.
Parameters:

seg source real mode segment.

ofs source real mode offset.

data destination.

count number of bytes to copy.

Notes: No range checking is performed.
For an example, see global_dos_alloc (292).

```
dosmemmove : procedure(sseg: Word;sofs: Word;dseg: Word;dofs: Word;count: LongInt) =
```

Copies count bytes of data between two dos real mode memory locations.
Parameters:

sseg source real mode segment.

sofs source real mode offset.

dseg destination real mode segment.

dofs destination real mode offset.

count number of bytes to copy.

Notes: No range check is performed in any way.

```
dosmemput : procedure(seg: Word;ofs: Word;var data;count: LongInt) = @dpmi_dosmemput
```

Copies heap data to dos real mode memory.
Parameters:

seg destination real mode segment.

ofs destination real mode offset.

data source.

count number of bytes to copy.

Notes: No range checking is performed.
For an example, see global_dos_alloc (292).

```
interruptflag = $200
```

Check for interrupt flag in trealregs (275)

```
overflowflag = $800
```

Check for overflow flag in trealregs (275)

```
parityflag = $004
```

Check for parity flag in trealregs (275)

```
rm_dpmi = 4
```

get_run_mode (290) return value: DPMI (e.g. dos box or 386Max)

```
rm_raw = 1
```

get_run_mode (290) return value: raw (without HIMEM)

```
rm_unknown = 0
```

get_run_mode (290) return value: Unknown runmode

```
rm_vcpi = 3
```

get_run_mode (290) return value: VCPI (with HIMEM and EMM386)

```
rm_xms = 2
```

get_run_mode (290) return value: XMS (with HIMEM, without EMM386)

```
signflag = $080
```

Check for sign flag in trealregs (275)

```
trapflag = $100
```

Check for trap flag in trealregs (275)

```
zeroflag = $040
```

Check for zero flag in trealregs (275)

54.17.2 Types

```
registers = trealregs
```

Alias for trealregs (275)

```
tdpmiversioninfo = record
  major : Byte;
  minor : Byte;
  flags : Word;
  cpu : Byte;
  master_pic : Byte;
  slave_pic : Byte;
end
```

`tdpmiversioninfo` describes the dpmi version information, as returned by get_dpmi_version (283). The CPU field can have the following values:

$02H 80286

$03H 80386

$04H 80486

$05H- Newer than 80486

The flags field is a bitmask with the following bits:

0 0 for 16 bit DPMI, 1 for 32-bit

1 0 for virtual 86 mode for reflected interrupts, 1 for return to real mode.

2 0 for no virtual memory support, 1 for virtual memory support.

```
tmeminfo = record
  available_memory : LongInt;
  available_pages : LongInt;
  available_lockable_pages : LongInt;
  linear_space : LongInt;
  unlocked_pages : LongInt;
  available_physical_pages : LongInt;
  total_physical_pages : LongInt;
  free_linear_space : LongInt;
  max_pages_in_paging_file : LongInt;
  reserved0 : LongInt;
  reserved1 : LongInt;
  reserved2 : LongInt;
end
```

`tmeminfo` Holds information about the memory allocation, etc.

NOTE: The value of a field is -1 (0ffffffffh) if the value is unknown, it's only guaranteed, that `available_memory` contains a valid value. The size of the pages can be determined by the get_page_size() function.

```
trealregs = record
case Integer of
1: (
  EDI : LongInt;
  ESI : LongInt;
  EBP : LongInt;
  Res : LongInt;
  EBX : LongInt;
  EDX : LongInt;
  ECX : LongInt;
  EAX : LongInt;
  Flags : Word;
  ES : Word;
  DS : Word;
  FS : Word;
  GS : Word;
  IP : Word;
  CS : Word;
  SP : Word;
  SS : Word;
);
2: (
  DI : Word;
  DI2 : Word;
  SI : Word;
  SI2 : Word;
  BP : Word;
  BP2 : Word;
  R1 : Word;
  R2 : Word;
  BX : Word;
  BX2 : Word;
  DX : Word;
  DX2 : Word;
  CX : Word;
  CX2 : Word;
  AX : Word;
  AX2 : Word;
);
3: (
  stuff : Array[1..4] of LongInt;
  BL : Byte;
  BH : Byte;
  BL2 : Byte;
  BH2 : Byte;
  DL : Byte;
  DH : Byte;
  DL2 : Byte;
  DH2 : Byte;
  CL : Byte;
  CH : Byte;
  CL2 : Byte;
  CH2 : Byte;
  AL : Byte;
```

```
  AH : Byte;
  AL2 : Byte;
  AH2 : Byte;
);
4: (
  RealEDI : LongInt;
  RealESI : LongInt;
  RealEBP : LongInt;
  RealRES : LongInt;
  RealEBX : LongInt;
  RealEDX : LongInt;
  RealECX : LongInt;
  RealEAX : LongInt;
  RealFlags : Word;
  RealES : Word;
  RealDS : Word;
  RealFS : Word;
  RealGS : Word;
  RealIP : Word;
  RealCS : Word;
  RealSP : Word;
  RealSS : Word;
);
end
```

The `trealregs` type contains the data structure to pass register values to a interrupt handler or real mode callback.

```
tseginfo = record
  offset : pointer;
  segment : Word;
end
```

This record is used to store a full 48-bit pointer. This may be either a protected mode selector:offset address or in real mode a segment:offset address, depending on application.

See also: Selectors and descriptors, dos memory access, Interrupt redirection

54.17.3 Variables

```
dosmemselector : Word
```

Selector to the dos memory. The whole dos memory is automatically mapped to this single descriptor at startup. This selector is the recommened way to access dos memory.

```
int31error : Word
```

This variable holds the result of a DPMI interrupt call. Any nonzero value must be treated as a critical failure.

54.18 Procedures and functions

54.18.1 allocate_ldt_descriptors

Synopsis: Allocate a number of descriptors

Declaration: `function allocate_ldt_descriptors(count: Word) : Word`

Visibility: default

Description: Allocates a number of new descriptors.

Parameters:

count:\specifies the number of requested unique descriptors.

Return value: The base selector.

Remark: Notes: The descriptors allocated must be initialized by the application with other function calls. This function returns descriptors with a limit and size value set to zero. If more than one descriptor was requested, the function returns a base selector referencing the first of a contiguous array of descriptors. The selector values for subsequent descriptors in the array can be calculated by adding the value returned by the get_next_selector_increment_value (285) function.

Errors: Check the int31error (275) variable.

See also: free_ldt_descriptor (281), get_next_selector_increment_value (285), segment_to_descriptor (299), create_code_segment_alias_descriptor (279), set_segment_limit (304), set_segment_base_address (303)

Listing: ./go32ex/seldes.pp

```
{$mode delphi}
uses
        crt,
        go32;

const
        maxx = 80;
        maxy = 25;
        bytespercell = 2;
        screensize = maxx * maxy * bytespercell;

        linB8000 = $B800 * 16;

type
        string80 = string[80];

var
        text_save : array[0..screensize-1] of byte;
        text_oldx, text_oldy : Word;

        text_sel : Word;

procedure status(s : string80);
begin
     gotoxy(1, 1); clreol; write(s); readkey;
end;

procedure selinfo(sel : Word);
```

```
begin
     gotoxy(1, 24);
     clreol; writeln('Descriptor base address : $',
        hexstr(get_segment_base_address(sel), 8));
     clreol; write('Descriptor limit : ', get_segment_limit(sel));
end;

function makechar(ch : char; color : byte) : Word;
begin
     result := byte(ch) or (color shl 8);
end;

begin
     seg_move(dosmemselector, linB8000, get_ds, longint(@text_save),
        screensize);
     text_oldx := wherex; text_oldy := wherey;
     seg_fillword(dosmemselector, linB8000, screensize div 2,
        makechar(' ', Black or (Black shl 4)));
     status('Creating selector ''text_sel'' to a part of ' +
        'text screen memory');
     text_sel := allocate_ldt_descriptors(1);
     set_segment_base_address(text_sel,
        linB8000 + bytespercell * maxx * 1);
     set_segment_limit(text_sel, screensize - 1 - bytespercell *
        maxx * 3);
     selinfo(text_sel);

     status('and clearing entire memory selected by ''text_sel''' +
        ' descriptor');
     seg_fillword(text_sel, 0, (get_segment_limit(text_sel)+1) div 2,
        makechar(' ', LightBlue shl 4));

     status('Notice that only the memory described by the' +
        ' descriptor changed, nothing else');

     status('Now reducing it''s limit and base and setting it''s ' +
        'described memory');
     set_segment_base_address(text_sel,
        get_segment_base_address(text_sel) + bytespercell * maxx);
     set_segment_limit(text_sel,
        get_segment_limit(text_sel) - bytespercell * maxx * 2);
     selinfo(text_sel);
     status('Notice that the base addr increased by one line but ' +
        'the limit decreased by 2 lines');
     status('This should give you the hint that the limit is ' +
        'relative to the base');
     seg_fillword(text_sel, 0, (get_segment_limit(text_sel)+1) div 2,
        makechar(#176, LightMagenta or Brown shl 4));

     status('Now let''s get crazy and copy 10 lines of data from ' +
        'the previously saved screen');
     seg_move(get_ds, longint(@text_save), text_sel,
        maxx * bytespercell * 2, maxx * bytespercell * 10);

     status('At last freeing the descriptor and restoring the old '+
        ' screen contents..');
     status('I hope this little program may give you some hints on '+
        'working with descriptors');
```

```
    free_ldt_descriptor(text_sel);
    seg_move(get_ds, longint(@text_save), dosmemselector,
      linB8000, screensize);
    gotoxy(text_oldx, text_oldy);
end.
```

54.18.2 allocate_memory_block

Synopsis: Allocate a block of linear memory

Declaration: `function allocate_memory_block(size: LongInt) : LongInt`

Visibility: default

Description: Allocates a block of linear memory.

Parameters:

size: Size of requested linear memory block in bytes.

Returned values: blockhandle - the memory handle to this memory block. Linear address of the requested memory.

Remark: *warning* According to my DPMI docs this function is not implemented correctly. Normally you should also get a blockhandle to this block after successful operation. This handle can then be used to free the memory block afterwards or use this handle for other purposes. Since the function isn't implemented correctly, and doesn't return a blockhandle, the block can't be deallocated and is hence unusuable ! This function doesn't allocate any descriptors for this block, it's the applications resposibility to allocate and initialize for accessing this memory.

Errors: Check the int31error (275) variable.

See also: free_memory_block (282)

54.18.3 copyfromdos

Synopsis: Copy data from DOS to heap

Declaration: `procedure copyfromdos(var addr;len: LongInt)`

Visibility: default

Description: Copies data from the pre-allocated dos memory transfer buffer to the heap.

Parameters:

addrdata to copy to.

lennumber of bytes to copy to heap.

Notes: Can only be used in conjunction with the dos memory transfer buffer.

Errors: Check the int31error (275) variable.

See also: tb_size (305), transfer_buffer (305), copytodos (279)

54.18.4 copytodos

Synopsis: Copy data from heap to DOS memory

Declaration: `procedure copytodos(var addr;len: LongInt)`

Visibility: default

Description: Copies data from heap to the pre-allocated dos memory buffer.

Parameters:

addrdata to copy from.

lennumber of bytes to copy to dos memory buffer.

Notes: This function fails if you try to copy more bytes than the transfer buffer is in size. It can only be used in conjunction with the transfer buffer.

Errors: Check the int31error (275) variable.

See also: tb_size (305), transfer_buffer (305), copyfromdos (278)

54.18.5 create_code_segment_alias_descriptor

Synopsis: Create new descriptor from existing descriptor

Declaration: `function create_code_segment_alias_descriptor(seg: Word) : Word`

Visibility: default

Description: Creates a new descriptor that has the same base and limit as the specified descriptor.

Parameters:

segDescriptor.

Return values: The data selector (alias).

Notes: In effect, the function returns a copy of the descriptor. The descriptor alias returned by this function will not track changes to the original descriptor. In other words, if an alias is created with this function, and the base or limit of the original segment is then changed, the two descriptors will no longer map the same memory.

Errors: Check the int31error (275) variable.

See also: allocate_ldt_descriptors (276), set_segment_limit (304), set_segment_base_address (303)

54.18.6 disable

Synopsis: Disable hardware interrupts

Declaration: `procedure disable`

Visibility: default

Description: Disables all hardware interrupts by execution a CLI instruction.

Errors: None.

See also: enable (281)

54.18.7 dpmi_dosmemfillchar

Synopsis: Fill DOS memory with a character

Declaration:
```
procedure dpmi_dosmemfillchar(seg: Word;ofs: Word;count: LongInt;
                              c: Char)
```

Visibility: default

Description: `dpmi_dosmemfillchar` fills the DOS memory reagion indicated by `seg`,`ofs` with `count` characters `c`.

See also: dpmi_dosmemput (281), dpmi_dosmemget (280), dpmi_dosmemmove (280), dpmi_dosmemfillword (280)

54.18.8 dpmi_dosmemfillword

Synopsis: Fill DOS memory with a word value

Declaration:
```
procedure dpmi_dosmemfillword(seg: Word;ofs: Word;count: LongInt;
                              w: Word)
```

Visibility: default

Description: `dpmi_dosmemfillword` fills the DOS memory reagion indicated by `seg`,`ofs` with `count` words `W`.

See also: dpmi_dosmemput (281), dpmi_dosmemget (280), dpmi_dosmemfillchar (280), dpmi_dosmemmove (280)

54.18.9 dpmi_dosmemget

Synopsis: Move data from DOS memory to DPMI memory

Declaration:
```
procedure dpmi_dosmemget(seg: Word;ofs: Word;var data;count: LongInt)
```

Visibility: default

Description: `dpmi_dosmemput` moves `count` bytes of data from the DOS memory location indicated by `seg` and `ofs` to DPMI memory indicated by `data`.

See also: dpmi_dosmemput (281), dpmi_dosmemmove (280), dpmi_dosmemfillchar (280), dpmi_dosmemfillword (280)

54.18.10 dpmi_dosmemmove

Synopsis: Move DOS memory

Declaration:
```
procedure dpmi_dosmemmove(sseg: Word;sofs: Word;dseg: Word;dofs: Word;
                          count: LongInt)
```

Visibility: default

Description: `dpmi_dosmemmove` moves `count` bytes from DOS memory `sseg`,`sofs` to `dseg`,`dofs`.

See also: dpmi_dosmemput (281), dpmi_dosmemget (280), dpmi_dosmemfillchar (280), dpmi_dosmemfillword (280)

54.18.11 dpmi_dosmemput

Synopsis: Move data from DPMI memory to DOS memory.

Declaration: `procedure dpmi_dosmemput(seg: Word;ofs: Word;var data;count: LongInt)`

Visibility: default

Description: `dpmi_dosmemput` moves `count` bytes of data from `data` to the DOS memory location indicated by `seg` and `ofs`.

See also: dpmi_dosmemget (280), dpmi_dosmemmove (280), dpmi_dosmemfillchar (280), dpmi_dosmemfillword (280)

54.18.12 enable

Synopsis: Enable hardware interrupts

Declaration: `procedure enable`

Visibility: default

Description: Enables all hardware interrupts by executing a STI instruction.

Errors: None.

See also: disable (279)

54.18.13 free_ldt_descriptor

Synopsis: Free a descriptor

Declaration: `function free_ldt_descriptor(d: Word) : Boolean`

Visibility: default

Description: Frees a previously allocated descriptor.

Parameters:

desThe descriptor to be freed.

Return value: `True` if successful, `False` otherwise. Notes: After this call this selector is invalid and must not be used for any memory operations anymore. Each descriptor allocated with allocate_ldt_descriptors (276) must be freed individually with this function, even if it was previously allocated as a part of a contiguous array of descriptors.

For an example, see allocate_ldt_descriptors (276).

Errors: Check the int31error (275) variable.

See also: allocate_ldt_descriptors (276), get_next_selector_increment_value (285)

54.18.14 free_linear_addr_mapping

Synopsis: ? No description available

Declaration: `function free_linear_addr_mapping(linear_addr: dword) : Boolean`

Visibility: default

54.18.15 free_memory_block

Synopsis: Free allocated memory block

Declaration: `function free_memory_block(blockhandle: LongInt) : Boolean`

Visibility: default

Description: Frees a previously allocated memory block.

Parameters:

blockhandlethe handle to the memory area to free.

Return value: `True` if successful, `false` otherwise. Notes: Frees memory that was previously allocated with allocate_memory_block (278) . This function doesn't free any descriptors mapped to this block, it's the application's responsibility.

Errors: Check int31error (275) variable.

See also: allocate_memory_block (278)

54.18.16 free_rm_callback

Synopsis: Release real mode callback.

Declaration: `function free_rm_callback(var intaddr: tseginfo) : Boolean`

Visibility: default

Description: Releases a real mode callback address that was previously allocated with the get_rm_callback (287) function.

Parameters:

intaddrreal mode address buffer returned by get_rm_callback (287) .

Return values: `True` if successful, `False` if not

For an example, see get_rm_callback (287).

Errors: Check the int31error (275) variable.

See also: set_rm_interrupt (303), get_rm_callback (287)

54.18.17 get_cs

Synopsis: Get CS selector

Declaration: `function get_cs : Word`

Visibility: default

Description: Returns the cs selector.

Return value: The content of the cs segment register.

For an example, see set_pm_interrupt (302).

Errors: None.

See also: get_ds (283), get_ss (292)

54.18.18 get_descriptor_access_right

Synopsis: Get descriptor's access rights

Declaration: `function get_descriptor_access_right(d: Word) : LongInt`

Visibility: default

Description: Gets the access rights of a descriptor.

Parameters:

dselector to descriptor.

Return value: Access rights bit field.

Errors: Check the int31error (275) variable.

See also: set_descriptor_access_right (301)

54.18.19 get_dpmi_version

Synopsis: Return DPMI information

Declaration: `function get_dpmi_version(var version: tdpmiversioninfo) : Boolean`

Visibility: default

Description: `get_dpmi_version` returns version information (Int $31 Function $0400) in `Version` and returns `True` if the information was retrieved successfully, `false` if the call failed.

Errors: The call returns `false` if the information could not be retrieved.

See also: tdpmiversioninfo (273)

54.18.20 get_ds

Synopsis: Get DS Selector

Declaration: `function get_ds : Word`

Visibility: default

Description: Returns the ds selector.

Return values: The content of the ds segment register.

Errors: None.

See also: get_cs (282), get_ss (292)

54.18.21 get_exception_handler

Synopsis: Return current exception handler

Declaration: `function get_exception_handler(e: Byte;var intaddr: tseginfo) : Boolean`

Visibility: default

Description: `get_exception_handler` returns the exception handler for exception `E` in `intaddr`. It returns `True` if the call was successful, `False` if not.

See also: set_exception_handler (301), get_pm_exception_handler (286)

54.18.22 get_linear_addr

Synopsis: Convert physical to linear address

Declaration: `function get_linear_addr(phys_addr: LongInt;size: LongInt) : LongInt`

Visibility: default

Description: Converts a physical address into a linear address.

Parameters:

phys_addrphysical address of device.

sizeSize of region to map in bytes.

Return value: Linear address that can be used to access the physical memory. Notes: It's the applications resposibility to allocate and set up a descriptor for access to the memory. This function shouldn't be used to map real mode addresses.

Errors: Check the int31error (275) variable.

See also: allocate_ldt_descriptors (276), set_segment_limit (304), set_segment_base_address (303)

54.18.23 get_meminfo

Synopsis: Return information on the available memory

Declaration: `function get_meminfo(var meminfo: tmeminfo) : Boolean`

Visibility: default

Description: Returns information about the amount of available physical memory, linear address space, and disk space for page swapping.

Parameters:

meminfobuffer to fill memory information into.

Return values: Due to an implementation bug this function always returns `False`, but it always succeeds.

Remark: Notes: Only the first field of the returned structure is guaranteed to contain a valid value. Any fields that are not supported by the DPMI host will be set by the host to `-1 (0FFFFFFFFH)` to indicate that the information is not available. The size of the pages used by the DPMI host can be obtained with the get_page_size (286) function.

Errors: Check the int31error (275) variable.

See also: get_page_size (286)

Listing: ./go32ex/meminfo.pp

```
uses
        go32;

var
        meminfo : tmeminfo;

begin
        get_meminfo(meminfo);
        if (int31error <> 0) then begin
```

```
                Writeln('Error getting DPMI memory information... Halting');
                Writeln('DPMI error number : ', int31error);
        end else begin
                with meminfo do begin
                        Writeln('Largest available free block : ',
                                available_memory div 1024, ' kbytes');
                        if (available_pages <> -1) then
                                Writeln('Maximum available unlocked pages : ',
                                        available_pages);
                        if (available_lockable_pages <> -1) then
                                Writeln('Maximum lockable available pages : ',
                                        available_lockable_pages);
                        if (linear_space <> -1) then
                                Writeln('Linear address space size : ',
                                        linear_space*get_page_size div 1024, ' kbytes');
                        if (unlocked_pages <> -1) then
                                Writeln('Total number of unlocked pages : ',
                                        unlocked_pages);
                        if (available_physical_pages <> -1) then
                                Writeln('Total number of free pages : ',
                                        available_physical_pages);
                        if (total_physical_pages <> -1) then
                                Writeln('Total number of physical pages : ',
                                        total_physical_pages);
                        if (free_linear_space <> -1) then
                                Writeln('Free linear address space : ',
                                        free_linear_space*get_page_size div 1024,
                                        ' kbytes');
                        if (max_pages_in_paging_file <> -1) then
                                Writeln('Maximum size of paging file : ',
                                        max_pages_in_paging_file*get_page_size div 1024,
                                        ' kbytes');
                end;
        end;
end.
```

54.18.24 get_next_selector_increment_value

Synopsis: Return selector increment value

Declaration: `function get_next_selector_increment_value : Word`

Visibility: default

Description: Returns the selector increment value when allocating multiple subsequent descriptors via allocate_ldt_descriptors (276).

Return value: Selector increment value.

Remark: Notes: Because allocate_ldt_descriptors (276) only returns the selector for the first descriptor and so the value returned by this function can be used to calculate the selectors for subsequent descriptors in the array.

Errors: Check the int31error (275) variable.

See also: allocate_ldt_descriptors (276), free_ldt_descriptor (281)

54.18.25 get_page_attributes

Synopsis: ? No description available

Declaration:
```
function get_page_attributes(handle: dword;offset: dword;
                             pagecount: dword;buf: pointer) : Boolean
```

Visibility: default

54.18.26 get_page_size

Synopsis: Return the page size

Declaration:
```
function get_page_size : LongInt
```

Visibility: default

Description: Returns the size of a single memory page.

Return value: Size of a single page in bytes.

Remark: The returned size is typically 4096 bytes.

For an example, see get_meminfo (284).

Errors: Check the int31error (275) variable.

See also: get_meminfo (284)

54.18.27 get_pm_exception_handler

Synopsis: Get protected mode exception handler

Declaration:
```
function get_pm_exception_handler(e: Byte;var intaddr: tseginfo)
                                  : Boolean
```

Visibility: default

Description: `get_pm_exception_handler` returns the protected mode exception handler for exception `E` in `intaddr`. It returns `True` if the call was successful, `False` if not.

See also: get_exception_handler (283), set_pm_exception_handler (302)

54.18.28 get_pm_interrupt

Synopsis: Return protected mode interrupt handler

Declaration:
```
function get_pm_interrupt(vector: Byte;var intaddr: tseginfo) : Boolean
```

Visibility: default

Description: Returns the address of a current protected mode interrupt handler.

Parameters:

vectorinterrupt handler number you want the address to.

intaddrbuffer to store address.

Return values: `True` if successful, `False` if not.

Remark: The returned address is a protected mode selector:offset address.

For an example, see set_pm_interrupt (302).

Errors: Check the int31error (275) variable.

See also: set_pm_interrupt (302), set_rm_interrupt (303), get_rm_interrupt (290)

54.18.29 get_rm_callback

Synopsis: Return real mode callback

Declaration:
```
function get_rm_callback(pm_func: pointer;const reg: trealregs;
                         var rmcb: tseginfo) : Boolean
```

Visibility: default

Description: Returns a unique real mode `segment:offset` address, known as a "real mode callback," that will transfer control from real mode to a protected mode procedure.

Parameters:

pm_funcpointer to the protected mode callback function.

regsupplied registers structure.

rmcbbuffer to real mode address of callback function.

Return values: `True` if successful, otherwise `False`.

Remark: Callback addresses obtained with this function can be passed by a protected mode program for example to an interrupt handler, device driver, or TSR, so that the real mode program can call procedures within the protected mode program or notify the protected mode program of an event. The contents of the supplied regs structure is not valid after function call, but only at the time of the actual callback.

Errors: Check the int31error (275) variable.

See also: free_rm_callback (282)

Listing: ./go32ex/callback.pp

```
{$ASMMODE ATT}
{$MODE FPC}

uses
        crt,
        go32;

const
        mouseint = $33;

var
        mouse_regs     : trealregs; external name '___v2prt0_rmcb_regs';
        mouse_seginfo  : tseginfo;

var
        mouse_numbuttons : longint;

        mouse_action : word;
        mouse_x, mouse_y : Word;
        mouse_b : Word;

        userproc_installed : Longbool;
        userproc_length : Longint;
```

```
        userproc_proc : pointer;

procedure callback_handler; assembler;
asm
    pushw %ds
    pushl %eax
    movw %es, %ax
    movw %ax, %ds

    cmpl $1, USERPROC_INSTALLED
    jne .LNoCallback
    pushal
    movw DOSmemSELECTOR, %ax
    movw %ax, %fs
    call *USERPROC_PROC
    popal
.LNoCallback:

    popl %eax
    popw %ds

    pushl %eax
    movl (%esi), %eax
    movl %eax, %es:42(%edi)
    addw $4, %es:46(%edi)
    popl %eax
    iret
end;
procedure mouse_dummy; begin end;

procedure textuserproc;
begin
        mouse_b := mouse_regs.bx;
        mouse_x := (mouse_regs.cx shr 3) + 1;
        mouse_y := (mouse_regs.dx shr 3) + 1;
end;

procedure install_mouse(userproc : pointer; userproclen : longint);
var r : trealregs;
begin
        r.eax := $0; realintr(mouseint, r);
        if (r.eax <> $FFFF) then begin
                Writeln('No Microsoft compatible mouse found');
                Writeln('A Microsoft compatible mouse driver is necessary ',
                        'to run this example');
                halt;
        end;
        if (r.bx = $ffff) then mouse_numbuttons := 2
        else mouse_numbuttons := r.bx;
        Writeln(mouse_numbuttons, ' button Microsoft compatible mouse ',
                ' found.');
        if (userproc <> nil) then begin
                userproc_proc := userproc;
                userproc_installed := true;
                userproc_length := userproclen;
                lock_code(userproc_proc, userproc_length);
        end else begin
                userproc_proc := nil;
```

```
                userproc_length := 0;
                userproc_installed := false;
        end;
        lock_data(mouse_x, sizeof(mouse_x));
        lock_data(mouse_y, sizeof(mouse_y));
        lock_data(mouse_b, sizeof(mouse_b));
        lock_data(mouse_action, sizeof(mouse_action));

        lock_data(userproc_installed, sizeof(userproc_installed));
        lock_data(userproc_proc, sizeof(userproc_proc));

        lock_data(mouse_regs, sizeof(mouse_regs));
        lock_data(mouse_seginfo, sizeof(mouse_seginfo));
        lock_code(@callback_handler,
                longint(@mouse_dummy)-longint(@callback_handler));
        get_rm_callback(@callback_handler, mouse_regs, mouse_seginfo);
        r.eax := $0c; r.ecx := $7f;
        r.edx := longint(mouse_seginfo.offset);
        r.es := mouse_seginfo.segment;
        realintr(mouseint, r);
        r.eax := $01;
        realintr(mouseint, r);
end;

procedure remove_mouse;
var
        r : trealregs;
begin
        r.eax := $02; realintr(mouseint, r);
        r.eax := $0c; r.ecx := 0; r.edx := 0; r.es := 0;
        realintr(mouseint, r);
        free_rm_callback(mouse_seginfo);
        if (userproc_installed) then begin
                unlock_code(userproc_proc, userproc_length);
                userproc_proc := nil;
                userproc_length := 0;
                userproc_installed := false;
        end;
        unlock_data(mouse_x, sizeof(mouse_x));
        unlock_data(mouse_y, sizeof(mouse_y));
        unlock_data(mouse_b, sizeof(mouse_b));
        unlock_data(mouse_action, sizeof(mouse_action));

        unlock_data(userproc_proc, sizeof(userproc_proc));
        unlock_data(userproc_installed, sizeof(userproc_installed));

        unlock_data(mouse_regs, sizeof(mouse_regs));
        unlock_data(mouse_seginfo, sizeof(mouse_seginfo));
        unlock_code(@callback_handler,
                longint(@mouse_dummy)-longint(@callback_handler));
        fillchar(mouse_seginfo, sizeof(mouse_seginfo), 0);
end;


begin
        install_mouse(@textuserproc, 400);
        Writeln('Press any key to exit...');
        while (not keypressed) do begin
```

```
                gotoxy(1, wherey);
                write('MouseX : ', mouse_x:2, ' MouseY : ', mouse_y:2,
                      ' Buttons : ', mouse_b:2);
        end;
        remove_mouse;
end.
```

54.18.30 get_rm_interrupt

Synopsis: Get real mode interrupt vector

Declaration: `function get_rm_interrupt(vector: Byte;var intaddr: tseginfo) : Boolean`

Visibility: default

Description: Returns the contents of the current machine's real mode interrupt vector for the specified interrupt.

Parameters:

vectorinterrupt vector number.

intaddrbuffer to store real mode `segment:offset` address.

Return values: `True` if successful, `False` otherwise.

Remark: The returned address is a real mode segment address, which isn't valid in protected mode.

Errors: Check the int31error (275) variable.

See also: set_rm_interrupt (303), set_pm_interrupt (302), get_pm_interrupt (286)

54.18.31 get_run_mode

Synopsis: Return current run mode

Declaration: `function get_run_mode : Word`

Visibility: default

Description: Returns the current mode your application runs with.

Return values: One of the constants used by this function.

Errors: None.

See also: get_run_mode (290)

Listing: ./go32ex/getrunmd.pp

```
uses
        go32;

begin
        case (get_run_mode) of
                rm_unknown :
                        Writeln('Unknown environment found');
                rm_raw     :
                        Writeln('You are currently running in raw mode ',
                                '(without HIMEM)');
                rm_xms     :
```

```
                WriteLn('You are currently using HIMEM.SYS only');
        rm_vcpi :
                WriteLn('VCPI server detected. You''re using HIMEM and ',
                        'EMM386');
        rm_dpmi :
                WriteLn('DPMI detected. You''re using a DPMI host like ',
                        'a windows DOS box or CWSDPMI');
    end;
end.
```

54.18.32 get_segment_base_address

Synopsis: Return base address from descriptor table

Declaration: `function get_segment_base_address(d: Word) : LongInt`

Visibility: default

Description: Returns the 32-bit linear base address from the descriptor table for the specified segment.

Parameters:

dselector of the descriptor you want the base address of.

Return values: Linear base address of specified descriptor.

For an example, see allocate_ldt_descriptors (276).

Errors: Check the int31error (275) variable.

See also: allocate_ldt_descriptors (276), set_segment_base_address (303), allocate_ldt_descriptors (276), set_segment_limit (304), get_segment_limit (291)

54.18.33 get_segment_limit

Synopsis: Return segment limite from descriptor

Declaration: `function get_segment_limit(d: Word) : LongInt`

Visibility: default

Description: Returns a descriptors segment limit.

Parameters:

dselector.

Return value: Limit of the descriptor in bytes.

Errors: Returns zero if descriptor is invalid.

See also: allocate_ldt_descriptors (276), set_segment_limit (304), set_segment_base_address (303), get_segment_base_address (291)

54.18.34 get_ss

Synopsis: Return SS selector

Declaration: `function get_ss : Word`

Visibility: default

Description: Returns the ss selector.

Return values: The content of the ss segment register.

Errors: None.

See also: get_ds (283), get_cs (282)

54.18.35 global_dos_alloc

Synopsis: Allocate DOS real mode memory

Declaration: `function global_dos_alloc(bytes: LongInt) : LongInt`

Visibility: default

Description: Allocates a block of dos real mode memory.

Parameters:

bytessize of requested real mode memory.

Return values: The low word of the returned value contains the selector to the allocated dos memory block, the high word the corresponding real mode segment value. The offset value is always zero. This function allocates memory from dos memory pool, i.e. memory below the 1 MB boundary that is controlled by dos. Such memory blocks are typically used to exchange data with real mode programs, TSRs, or device drivers. The function returns both the real mode segment base address of the block and one descriptor that can be used by protected mode applications to access the block. This function should only used for temporary buffers to get real mode information (e.g. interrupts that need a data structure in ES:(E)DI), because every single block needs an unique selector. The returned selector should only be freed by a global_dos_free (293) call.

Errors: Check the int31error (275) variable.

See also: global_dos_free (293)

Listing: ./go32ex/buffer.pp

```
uses
        go32;

procedure dosalloc(var selector : word;
        var segment : word; size : longint);
var
        res : longint;
begin
        res := global_dos_alloc(size);
        selector := word(res);
        segment := word(res shr 16);
end;

procedure dosfree(selector : word);
begin
```

```
        global_dos_free(selector);
end;

type
        VBEInfoBuf = packed record
                Signature : array[0..3] of char;
                Version : Word;
                reserved : array[0..505] of byte;
        end;

var
        selector,
        segment : Word;

        r : trealregs;
        infobuf : VBEInfoBuf;

begin
        fillchar(r, sizeof(r), 0);
        fillchar(infobuf, sizeof(VBEInfoBuf), 0);
        dosalloc(selector, segment, sizeof(VBEInfoBuf));
        if (int31error<>0) then begin
                Writeln('Error while allocating real mode memory, halting');
                halt;
        end;
        infobuf.Signature := 'VBE2';
        dosmemput(segment, 0, infobuf, sizeof(infobuf));
        r.ax := $4f00; r.es := segment;
        realintr($10, r);
        dosmemget(segment, 0, infobuf, sizeof(infobuf));
        dosfree(selector);
        if (r.ax <> $4f) then begin
                Writeln('VBE BIOS extension not available, function call ',
                        'failed');
                halt;
        end;
        if (infobuf.signature[0] = 'V') and
                (infobuf.signature[1] = 'E') and
                (infobuf.signature[2] = 'S') and
                (infobuf.signature[3] = 'A') then begin
                Writeln('VBE version ', hi(infobuf.version), '.',
                        lo(infobuf.version), ' detected');
        end;
end.
```

54.18.36 global_dos_free

Synopsis: Free DOS memory block

Declaration: `function global_dos_free(selector: Word) : Boolean`

Visibility: default

Description: Frees a previously allocated dos memory block.

Parameters:

selectorselector to the dos memory block.

Return value: `True` if successful, `False` otherwise.

Remark: The descriptor allocated for the memory block is automatically freed and hence invalid for further use. This function should only be used for memory allocated by global_dos_alloc (292).

For an example, see global_dos_alloc (292).

Errors: Check the int31error (275) variable.

See also: global_dos_alloc (292)

54.18.37 inportb

Synopsis: Read byte from I/O port

Declaration: `function inportb(port: Word) : Byte`

Visibility: default

Description: Reads 1 byte from the selected I/O port.

Parameters:

portthe I/O port number which is read.

Return values: Current I/O port value.

Errors: None.

See also: outportb (296), inportw (294), inportl (294)

54.18.38 inportl

Synopsis: Read longint from I/O port

Declaration: `function inportl(port: Word) : LongInt`

Visibility: default

Description: Reads 1 longint from the selected I/O port.

Parameters:

portthe I/O port number which is read.

Return values: Current I/O port value.

Errors: None.

See also: outportb (296), inportb (294), inportw (294)

54.18.39 inportw

Synopsis: Read word from I/O port

Declaration: `function inportw(port: Word) : Word`

Visibility: default

Description: Reads 1 word from the selected I/O port.

Parameters:

portthe I/O port number which is read.

Return values: Current I/O port value.

Errors: None.

See also: outportw (297), inportb (294), inportl (294)

54.18.40 lock_code

Synopsis: Lock code memory range

Declaration: `function lock_code(functionaddr: pointer;size: LongInt) : Boolean`

Visibility: default

Description: Locks a memory range which is in the code segment selector.

Parameters:

functionaddraddress of the function to be locked.

sizesize in bytes to be locked.

Return values: `True` if successful, `False` otherwise.

For an example, see get_rm_callback (287).

Errors: Check the int31error (275) variable.

See also: lock_linear_region (296), lock_data (295), unlock_linear_region (306), unlock_data (306), unlock_code (305)

54.18.41 lock_data

Synopsis: Lock data memory range

Declaration: `function lock_data(var data;size: LongInt) : Boolean`

Visibility: default

Description: Locks a memory range which resides in the data segment selector.

Parameters:

dataaddress of data to be locked.

sizelength of data to be locked.

Return values: `True` if successful, `False` otherwise.

For an example, see get_rm_callback (287).

Errors: Check the int31error (275) variable.

See also: lock_linear_region (296), lock_code (295), unlock_linear_region (306), unlock_data (306), unlock_code (305)

54.18.42 lock_linear_region

Synopsis: Lock linear memory region

Declaration:
```
function lock_linear_region(linearaddr: LongInt;size: LongInt) : Boolean
```

Visibility: default

Description: Locks a memory region to prevent swapping of it.

Parameters:

linearaddrthe linear address of the memory are to be locked.

sizesize in bytes to be locked.

Return value: `True` if successful, False otherwise.

Errors: Check the int31error (275) variable.

See also: lock_data (295), lock_code (295), unlock_linear_region (306), unlock_data (306), unlock_code (305)

54.18.43 map_device_in_memory_block

Synopsis: Map a device into program's memory space

Declaration:
```
function map_device_in_memory_block(handle: LongInt;offset: LongInt;
                                    pagecount: LongInt;device: LongInt)
                                     : Boolean
```

Visibility: default

Description: `map_device_in_memory_block` allows to map a device in memory. This function is a direct call of the extender. For more information about it's arguments, see the extender documentation.

54.18.44 outportb

Synopsis: Write byte to I/O port

Declaration:
```
procedure outportb(port: Word;data: Byte)
```

Visibility: default

Description: Sends 1 byte of data to the specified I/O port.

Parameters:

portthe I/O port number to send data to.

datavalue sent to I/O port.

Return values: None.

Errors: None.

See also: inportb (294), outportl (297), outportw (297)

Listing: ./go32ex/outport.pp

```
uses
        crt,
        go32;

begin
        outportb($61, $ff);
        delay(50);
        outportb($61, $0);
end.
```

54.18.45 outportl

Synopsis: Write longint to I/O port

Declaration: `procedure outportl(port: Word;data: LongInt)`

Visibility: default

Description: Sends 1 longint of data to the specified I/O port.

Parameters:

portthe I/O port number to send data to.

datavalue sent to I/O port.

Return values: None.

For an example, see outportb (296).

Errors: None.

See also: inportl (294), outportw (297), outportb (296)

54.18.46 outportw

Synopsis: Write word to I/O port

Declaration: `procedure outportw(port: Word;data: Word)`

Visibility: default

Description: Sends 1 word of data to the specified I/O port.

Parameters:

portthe I/O port number to send data to.

datavalue sent to I/O port.

Return values: None.

For an example, see outportb (296).

Errors: None.

See also: inportw (294), outportl (297), outportb (296)

54.18.47 realintr

Synopsis: Simulate interrupt

Declaration: `function realintr(intnr: Word;var regs: trealregs) : Boolean`

Visibility: default

Description: Simulates an interrupt in real mode.

Parameters:

intnrinterrupt number to issue in real mode.

regsregisters data structure.

Return values: The supplied registers data structure contains the values that were returned by the real mode interrupt. `True` if successful, `False` if not.

Remark: The function transfers control to the address specified by the real mode interrupt vector of intnr. The real mode handler must return by executing an IRET.

Errors: Check the int31error (275) variable.

Listing: ./go32ex/flags.pp

```
uses
        go32;

var
        r : trealregs;

begin
        r.ax := $5300;
        r.bx := 0;
        realintr($15, r);
        if ((r.flags and carryflag)=0) then begin
                Writeln('APM v', (r.ah and $f), '.',
                        (r.al shr 4), (r.al and $f), ' detected');
        end else
                Writeln('APM not present');
end.
```

54.18.48 request_linear_region

Synopsis: Request linear address region.

Declaration: `function request_linear_region(linearaddr: LongInt;size: LongInt;`
`var blockhandle: LongInt) : Boolean`

Visibility: default

Description: `request_linear_region` requests a linear range of addresses of size `Size`, starting at `linearaddr`. If successful, `True` is returned, and a handle to the address region is returned in `blockhandle`.

Errors: On error, `False` is returned.

54.18.49 segment_to_descriptor

Synopsis: Map segment address to descriptor

Declaration: `function segment_to_descriptor(seg: Word) : Word`

Visibility: default

Description: Maps a real mode segment (paragraph) address onto an descriptor that can be used by a protected mode program to access the same memory.

Parameters:

segthe real mode segment you want the descriptor to.

Return values: Descriptor to real mode segment address.

Remark: The returned descriptors limit will be set to 64 kB. Multiple calls to this function with the same segment address will return the same selector. Descriptors created by this function can never be modified or freed. Programs which need to examine various real mode addresses using the same selector should use the function allocate_ldt_descriptors (276) and change the base address as necessary.

For an example, see seg_fillchar (299).

Errors: Check the int31error (275) variable.

See also: allocate_ldt_descriptors (276), free_ldt_descriptor (281), set_segment_base_address (303)

54.18.50 seg_fillchar

Synopsis: Fill segment with byte value

Declaration: `procedure seg_fillchar(seg: Word;ofs: LongInt;count: LongInt;c: Char)`

Visibility: default

Description: Sets a memory area to a specific value.

Parameters:

segselector to memory area.

ofsoffset to memory.

countnumber of bytes to set.

cbyte data which is set.

Return values: None.

Notes: No range check is done in any way.

Errors: None.

See also: seg_move (300), seg_fillword (300), dosmemfillchar (270), dosmemfillword (271), dosmemget (271), dosmemput (271), dosmemmove (271)

Listing: ./go32ex/vgasel.pp

```
uses
        go32;

var
        vgasel : Word;
        r : trealregs;

begin
        r.eax := $13; realintr($10, r);
        vgasel := segment_to_descriptor($A000);
        seg_fillchar(vgasel, 0, 64000, #15);
        readln;
        r.eax := $3; realintr($10, r);
end.
```

54.18.51 seg_fillword

Synopsis: Fill segment with word value

Declaration: `procedure seg_fillword(seg: Word;ofs: LongInt;count: LongInt;w: Word)`

Visibility: default

Description: Sets a memory area to a specific value.

Parameters:

segselector to memory area.

ofsoffset to memory.

countnumber of words to set.

wword data which is set.

Return values: None.

Notes: No range check is done in any way.

For an example, see allocate_ldt_descriptors (276).

Errors: None.

See also: seg_move (300), seg_fillchar (299), dosmemfillchar (270), dosmemfillword (271), dosmemget (271), dosmemput (271), dosmemmove (271)

54.18.52 seg_move

Synopsis: Move data between 2 locations

Declaration: `procedure seg_move(sseg: Word;source: LongInt;dseg: Word;dest: LongInt;`
`count: LongInt)`

Visibility: default

Description: Copies data between two memory locations.

Parameters:

ssegsource selector.

sourcesource offset.

dsegdestination selector.

destdestination offset.

countsize in bytes to copy.

Return values: None.

Remark: Overlapping is only checked if the source selector is equal to the destination selector. No range check is done.

For an example, see allocate_ldt_descriptors (276).

Errors: None.

See also: seg_fillchar (299), seg_fillword (300), dosmemfillchar (270), dosmemfillword (271), dosmemget (271), dosmemput (271), dosmemmove (271)

54.18.53 set_descriptor_access_right

Synopsis: Set access rights to memory descriptor

Declaration:
```
function set_descriptor_access_right(d: Word;w: Word) : LongInt
```

Visibility: default

Description: `set_descriptor_access_right` sets the access rights for descriptor `d` to `w`

54.18.54 set_exception_handler

Synopsis: Set exception handler

Declaration:
```
function set_exception_handler(e: Byte;const intaddr: tseginfo)
                                    : Boolean
```

Visibility: default

Description: `set_exception_handler` sets the exception handler for exception `E` to `intaddr`. It returns `True` if the call was successful, `False` if not.

See also: get_exception_handler (283), set_pm_exception_handler (302)

54.18.55 set_page_attributes

Synopsis: ? No description available

Declaration:
```
function set_page_attributes(handle: dword;offset: dword;
                             pagecount: dword;buf: pointer) : Boolean
```

Visibility: default

54.18.56 set_pm_exception_handler

Synopsis: Set protected mode exception handler

Declaration:
```
function set_pm_exception_handler(e: Byte;const intaddr: tseginfo)
                                        : Boolean
```

Visibility: default

Description: `set_pm_exception_handler` sets the protected mode exception handler for exception `E` to `intaddr`. It returns `True` if the call was successful, `False` if not.

See also: set_exception_handler (301), get_pm_exception_handler (286)

54.18.57 set_pm_interrupt

Synopsis: Set protected mode interrupt handler

Declaration:
```
function set_pm_interrupt(vector: Byte;const intaddr: tseginfo)
                                : Boolean
```

Visibility: default

Description: Sets the address of the protected mode handler for an interrupt.

Parameters:

vectornumber of protected mode interrupt to set.

intaddrselector:offset address to the interrupt vector.

Return values: `True` if successful, `False` otherwise.

Remark: The address supplied must be a valid `selector:offset` protected mode address.

Errors: Check the int31error (275) variable.

See also: get_pm_interrupt (286), set_rm_interrupt (303), get_rm_interrupt (290)

Listing: ./go32ex/intpm.pp

```
uses
        crt,
        go32;

const
        int1c = $1c;

var
        oldint1c : tseginfo;
        newint1c : tseginfo;

        int1c_counter : Longint;

        int1c_ds : Word; external name '___v2prt0_ds_alias';

procedure int1c_handler; assembler;
asm
    cli
    pushw %ds
    pushw %ax
```

```
    movw %cs:int1c_ds, %ax
    movw %ax, %ds
    incl int1c_counter
    popw %ax
    popw %ds
    sti
    iret
end;

var i : Longint;

begin
     newint1c.offset := @int1c_handler;
     newint1c.segment := get_cs;
     get_pm_interrupt(int1c, oldint1c);
     Writeln('-- Press any key to exit --');
     set_pm_interrupt(int1c, newint1c);
     while (not keypressed) do begin
           gotoxy(1, wherey);
           write('Number of interrupts occured : ', int1c_counter);
     end;
     set_pm_interrupt(int1c, oldint1c);
end.
```

54.18.58 set_rm_interrupt

Synopsis: Set real mode interrupt handler

Declaration:
```
function set_rm_interrupt(vector: Byte;const intaddr: tseginfo)
                                 : Boolean
```

Visibility: default

Description: Sets a real mode interrupt handler.

Parameters:

vectorthe interrupt vector number to set.

intaddraddress of new interrupt vector.

Return values: `True` if successful, otherwise `False`.

Remark: The address supplied MUST be a real mode segment address, not a `selector:offset` address. So the interrupt handler must either reside in dos memory (below 1 Mb boundary) or the application must allocate a real mode callback address with get_rm_callback (287).

Errors: Check the int31error (275) variable.

See also: get_rm_interrupt (290), set_pm_interrupt (302), get_pm_interrupt (286), get_rm_callback (287)

54.18.59 set_segment_base_address

Synopsis: Set descriptor's base address

Declaration:
```
function set_segment_base_address(d: Word;s: LongInt) : Boolean
```

Visibility: default

Description: Sets the 32-bit linear base address of a descriptor.

Parameters:

dselector.

snew base address of the descriptor.

Errors: Check the int31error (275) variable.

See also: allocate_ldt_descriptors (276), get_segment_base_address (291), allocate_ldt_descriptors (276), set_segment_limit (304), get_segment_base_address (291), get_segment_limit (291)

54.18.60 set_segment_limit

Synopsis: Set descriptor limit

Declaration: `function set_segment_limit(d: Word;s: LongInt) : Boolean`

Visibility: default

Description: Sets the limit of a descriptor.

Parameters:

dselector.

snew limit of the descriptor.

Return values: Returns `True` if successful, else `False`.

Remark: The new limit specified must be the byte length of the segment - 1. Segment limits bigger than or equal to 1MB must be page aligned, they must have the lower 12 bits set.

For an example, see allocate_ldt_descriptors (276).

Errors: Check the int31error (275) variable.

See also: allocate_ldt_descriptors (276), set_segment_base_address (303), get_segment_limit (291), set_segment_limit (304)

54.18.61 tb_offset

Synopsis: Return DOS transfer buffer offset

Declaration: `function tb_offset : LongInt`

Visibility: default

Description: `tb_offset` returns the DOS transfer buffer segment.

See also: transfer_buffer (305), tb_segment (304), tb_size (305)

54.18.62 tb_segment

Synopsis: Return DOS transfer buffer segment

Declaration: `function tb_segment : LongInt`

Visibility: default

Description: `tb_segment` returns the DOS transfer buffer segment.

See also: transfer_buffer (305), tb_offset (304), tb_size (305)

54.18.63 tb_size

Synopsis: Return DOS transfer memory buffer size

Declaration: `function tb_size : LongInt`

Visibility: default

Description: Returns the size of the pre-allocated dos memory buffer.

Return values: The size of the pre-allocated dos memory buffer. This block always seems to be 16k in size, but don't rely on this.

Errors: None.

See also: transfer_buffer (305), copyfromdos (278), copytodos (279)

54.18.64 transfer_buffer

Synopsis: Return offset of DOS transfer buffer

Declaration: `function transfer_buffer : LongInt`

Visibility: default

Description: `transfer_buffer` returns the offset of the transfer buffer.

Errors: None.

See also: tb_size (305)

54.18.65 unlock_code

Synopsis: Unlock code segment

Declaration: `function unlock_code(functionaddr: pointer;size: LongInt) : Boolean`

Visibility: default

Description: Unlocks a memory range which resides in the code segment selector.

Parameters:

functionaddraddress of function to be unlocked.

sizesize bytes to be unlocked.

Return value: `True` if successful, `False` otherwise.

For an example, see get_rm_callback (287).

Errors: Check the int31error (275) variable.

See also: unlock_linear_region (306), unlock_data (306), lock_linear_region (296), lock_data (295), lock_code (295)

54.18.66 unlock_data

Synopsis: Unlock data segment

Declaration:
```
function unlock_data(var data;size: LongInt) : Boolean
```

Visibility: default

Description: Unlocks a memory range which resides in the data segment selector.

Parameters:

dataaddress of memory to be unlocked.

sizesize bytes to be unlocked.

Return values: `True` if successful, `False` otherwise.

For an example, see get_rm_callback (287).

Errors: Check the int31error (275) variable.

See also: unlock_linear_region (306), unlock_code (305), lock_linear_region (296), lock_data (295), lock_code (295)

54.18.67 unlock_linear_region

Synopsis: Unlock linear memory region

Declaration:
```
function unlock_linear_region(linearaddr: LongInt;size: LongInt)
                                    : Boolean
```

Visibility: default

Description: Unlocks a previously locked linear region range to allow it to be swapped out again if needed.

Parameters:

linearaddrlinear address of the memory to be unlocked.

sizesize bytes to be unlocked.

Return values: `True` if successful, `False` otherwise.

Errors: Check the int31error (275) variable.

See also: unlock_data (306), unlock_code (305), lock_linear_region (296), lock_data (295), lock_code (295)

Chapter 55

Reference for unit 'gpm'

55.1 Used units

Table 55.1: Used units by unit 'gpm'

Name	Page
BaseUnix	51
System	622

55.2 Overview

The GPM unit implements an interface to libgpm, the console program for mouse handling. This unit was created by Peter Vreman, and is only available on linux.

When this unit is used, your program is linked to the C libraries, so you must take care of the C library version. Also, it will only work with version 1.17 or higher of the libgpm library.

55.3 Constants, types and variables

55.3.1 Constants

```
GPM_BOT = 2
```

Bottom of area.

```
GPM_B_LEFT = 4
```

Left mouse button identifier.

```
GPM_B_MIDDLE = 2
```

Middle mouse button identifier.

```
GPM_B_RIGHT = 1
```

Right mouse button identifier.

```
GPM_DOUBLE = 32
```

Mouse double click event.

```
GPM_DOWN = 4
```

Mouse button down event.

```
GPM_DRAG = 2
```

Mouse drag event.

```
GPM_ENTER = 512
```

Enter area event.

```
GPM_HARD = 256
```

?

```
GPM_LEAVE = 1024
```

Leave area event.

```
GPM_LFT = 4
```

Left side of area.

```
GPM_MAGIC = $47706D4C
```

Constant identifying GPM in Gpm_Open (315).

```
GPM_MFLAG = 128
```

Motion flag.

```
GPM_MOVE = 1
```

Mouse move event.

```
GPM_NODE_CTL = GPM_NODE_DEV
```

Control socket

```
GPM_NODE_DEV = '/dev/gpmctl'
```

Device socket filename

```
GPM_NODE_DIR = _PATH_VARRUN
```

Where to write socket.

```
GPM_NODE_DIR_MODE = 0775
```

Mode of socket.

```
GPM_NODE_FIFO = '/dev/gpmdata'
```

FIFO name

```
GPM_NODE_PID = '/var/run/gpm.pid'
```

Name of PID file.

```
GPM_RGT = 8
```

Right side of area.

```
GPM_SINGLE = 16
```

Mouse single click event.

```
GPM_TOP = 1
```

Top of area.

```
GPM_TRIPLE = 64
```

Mouse triple click event.

```
GPM_UP = 8
```

Mouse button up event.

```
_PATH_DEV = '/dev/'
```

Location of /dev directory.

```
_PATH_VARRUN = '/var/run/'
```

Location of run PID files directory.

55.3.2 Types

```
Pgpmconnect = Pgpm_connect
```

Pointer to TGpmConnect (310) record.

```
Pgpmevent = Pgpm_event
```

Pointer to TGpmEvent (310) record

```
Pgpmroi = Pgpm_roi
```

Pointer to TGpmRoi (310) record.

```
Pgpm_connect = ^TGpm_connect
```

Pointer to TGpm_Connect (311) record.

```
Pgpm_event = ^Tgpm_event
```

Pointer to TGpm_Event (311) record

```
Pgpm_roi = ^Tgpm_roi
```

Pointer to Tgpm_roi (311) record.

```
Tgpmconnect = Tgpm_connect
```

Alias for TGpm_Connect (311) record.

```
TGpmEtype = LongInt
```

Type for event type.

```
Tgpmevent = Tgpm_event
```

Alias for TGPM_EVent (311) record

```
TGpmHandler = function(var event: Tgpmevent;clientdata: pointer)
                       : LongInt
```

Mouse event handler callback.

```
TGpmMargin = LongInt
```

Type to hold area margin.

```
Tgpmroi = Tgpm_roi
```

Alias for TGpm_roi (311)Record

```
Tgpm_connect = record
  eventMask : Word;
  defaultMask : Word;
  minMod : Word;
  maxMod : Word;
  pid : LongInt;
  vc : LongInt;
end
```

GPM server connection information.

```
Tgpm_event = record
  buttons : Byte;
  modifiers : Byte;
  vc : Word;
  dx : Word;
  dy : Word;
  x : Word;
  y : Word;
  EventType : TGpmEtype;
  clicks : LongInt;
  margin : TGpmMargin;
  wdx : Word;
  wdy : Word;
end
```

`Tgpm_event` describes the events that are reported by GPM.

```
Tgpm_roi = record
  xmin : Integer;
  xmax : Integer;
  ymin : Integer;
  ymax : Integer;
  minmod : Word;
  maxmod : Word;
  eventmask : Word;
  owned : Word;
  handler : TGpmHandler;
  clientdata : pointer;
  prev : Pgpm_roi;
  next : Pgpm_roi;
end
```

Record used to define regions of interest.

55.3.3 Variables

`gpm_current_roi : Pgpm_roi`

Internal gpm library variable. Do not use.

`gpm_handler : TGpmHandler`

Internal gpm library variable. Do not use.

`gpm_roi : Pgpm_roi`

Internal gpm library variable. Do not use.

`gpm_roi_data : pointer`

Internal gpm library variable. Do not use.

```
gpm_roi_handler : TGpmHandler
```

Internal gpm library variable. Do not use.

55.4 Procedures and functions

55.4.1 Gpm_AnyDouble

Synopsis: Check whether event has double click event.

Declaration: `function Gpm_AnyDouble(EventType: LongInt) : Boolean`

Visibility: default

Description: `Gpm_AnyDouble` returns `True` if `EventType` contains the `GPM_DOUBLE` flag, `False` otherwise.

Errors: None.

See also: Gpm_StrictSingle (317), Gpm_AnySingle (312), Gpm_StrictDouble (317), Gpm_StrictTriple (317), Gpm_AnyTriple (312)

55.4.2 Gpm_AnySingle

Synopsis: Check whether event has a single click event.

Declaration: `function Gpm_AnySingle(EventType: LongInt) : Boolean`

Visibility: default

Description: `Gpm_AnySingle` returns `True` if `EventType` contains the `GPM_SINGLE` flag, `False` otherwise.

Errors: None.

See also: Gpm_StrictSingle (317), Gpm_AnyDouble (312), Gpm_StrictDouble (317), Gpm_StrictTriple (317), Gpm_AnyTriple (312)

55.4.3 Gpm_AnyTriple

Synopsis: Check whether event has a triple click event.

Declaration: `function Gpm_AnyTriple(EventType: LongInt) : Boolean`

Visibility: default

Description: `Gpm_AnySingle` returns `True` if `EventType` contains the `GPM_TRIPLE` flag, `False` otherwise.

Errors: None.

See also: Gpm_StrictSingle (317), Gpm_AnyDouble (312), Gpm_StrictDouble (317), Gpm_StrictTriple (317), Gpm_AnySingle (312)

55.4.4 gpm_close

Synopsis: Close connection to GPM server.

Declaration:
```
function gpm_close : LongInt
```

Visibility: default

Description: `Gpm_Close` closes the current connection, and pops the connection stack; this means that the previous connection becomes active again.

The function returns -1 if the current connection is not the last one, and it returns 0 if the current connection is the last one.

for an example, see Gpm_GetEvent (313).

Errors: None.

See also: Gpm_Open (315)

55.4.5 gpm_fitvalues

Synopsis: Change coordinates to fit physical screen.

Declaration:
```
function gpm_fitvalues(var x: LongInt;var y: LongInt) : LongInt
```

Visibility: default

Description: `Gpm_fitValues` changes `x` and `y` so they fit in the visible screen. The actual mouse pointer is not affected by this function.

Errors: None.

See also: Gpm_FitValuesM (313)

55.4.6 gpm_fitvaluesM

Synopsis: Change coordinates to fit margin.

Declaration:
```
function gpm_fitvaluesM(var x: LongInt;var y: LongInt;margin: LongInt)
                                : LongInt
```

Visibility: default

Description: `Gpm_FitValuesM` changes `x` and `y` so they fit in the margin indicated by `margin`. If `margin` is -1, then the values are fitted to the screen. The actual mouse pointer is not affected by this function.

Errors: None.

See also: Gpm_FitValues (313)

55.4.7 gpm_getevent

Synopsis: Get event from event queue.

Declaration:
```
function gpm_getevent(var event: Tgpm_event) : LongInt
```

Visibility: default

Description: `Gpm_GetEvent` Reads an event from the file descriptor `gpm_fd`. This file is only for internal use and should never be called by a client application.

It returns 1 on succes, and -1 on failue.

Errors: On error, -1 is returned.

See also: Gpm_GetSnapshot (315)

Listing: ./gpmex/gpmex.pp

```
program gpmex;

{
  Example program to demonstrate the use of the gpm unit.
}

uses gpm;

var
  connect : TGPMConnect;
  event : tgpmevent;

begin
  connect.EventMask:=GPM_MOVE or GPM_DRAG or GPM_DOWN or GPM_UP;
  connect.DefaultMask:=0;
  connect.MinMod:=0;
  connect.MaxMod:=0;
  if Gpm_Open(connect,0)=-1 then
    begin
    Writeln('No mouse handler present.');
    Halt(1);
    end;
  Writeln('Click right button to end.');
  Repeat
    gpm_getevent(Event);
    With Event do
      begin
        Write('Pos = (',X,',',Y,') Buttons : (');
        if (buttons and Gpm_b_left)<>0 then
          write('left ');
        if (buttons and Gpm_b_right)<>0 then
          write('right ');
        if (buttons and Gpm_b_middle)<>0 then
          Write('middle ');
        Write(') Event : ');
        Case EventType and $F of
          GPM_MOVE: write('Move');
          GPM_DRAG: write('Drag');
          GPM_DOWN: write('Down');
          GPM_UP: write('Up');
        end;
        Writeln;
      end;
  Until (Event.Buttons and gpm_b_right)<>0;
  gpm_close;
end.
```

55.4.8 gpm_getsnapshot

Synopsis: Return servers' current image of mouse state.

Declaration:
```
function gpm_getsnapshot(eptr: Pgpmevent) : LongInt
function gpm_getsnapshot(var eptr: Tgpmevent) : LongInt
```

Visibility: default

Description: `Gpm_GetSnapshot` returns the picture that the server has of the current situation in `Event`. This call will not read the current situation from the mouse file descriptor, but returns a buffered version.

The function returns the number of mouse buttons, or -1 if this information is not available.

Errors: None.

See also: Gpm_GetEvent (313)

55.4.9 gpm_lowerroi

Synopsis: Lower a region of interest in the stack.

Declaration:
```
function gpm_lowerroi(which: Pgpm_roi;after: Pgpm_roi) : Pgpm_roi
```

Visibility: default

Description: `Gpm_LowerRoi` lowers the region of interest `which` after `after`. If `after` is `Nil`, the region of interest is moved to the bottom of the stack.

The return value is the new top of the region-of-interest stack.

Errors: None.

See also: Gpm_RaiseRoi (316), Gpm_PopRoi (316), Gpm_PushRoi (316)

55.4.10 gpm_open

Synopsis: Open connection to GPM server.

Declaration:
```
function gpm_open(var conn: Tgpm_connect;flag: LongInt) : LongInt
```

Visibility: default

Description: `Gpm_Open` opens a new connection to the mouse server. The connection is described by the fields of the `conn` record of type TGPMConnect (310).

if `Flag` is 0, then the application only receives events that come from its own terminal device. If it is negative it will receive all events. If the value is positive then it is considered a console number to which to connect.

The return value is -1 on error, or the file descriptor used to communicate with the client. Under an X-Term the return value is -2.

for an example, see Gpm_GetEvent (313).

Errors: On Error, the return value is -1.

See also: Gpm_Open (315)

55.4.11 gpm_poproi

Synopsis: Pop region of interest from the stack.

Declaration: `function gpm_poproi(which: Pgpm_roi) : Pgpm_roi`

Visibility: default

Description: `Gpm_PopRoi` pops the topmost region of interest from the stack. It returns the next element on the stack, or `Nil` if the current element was the last one.

Errors: None.

See also: Gpm_RaiseRoi (316), Gpm_LowerRoi (315), Gpm_PushRoi (316)

55.4.12 gpm_pushroi

Synopsis: Push region of interest on the stack.

Declaration:
```
function gpm_pushroi(x1: LongInt;y1: LongInt;x2: LongInt;y2: LongInt;
                     mask: LongInt;fun: TGpmHandler;xtradata: pointer)
                      : Pgpm_roi
```

Visibility: default

Description: `Gpm_PushRoi` puts a new *region of interest* on the stack. The region of interest is defined by a rectangle described by the corners `(X1,Y1)` and `(X2,Y2)`.

The `mask` describes which events the handler {fun} will handle; `ExtraData` will be put in the `xtradata` field of the {TGPM_Roi} record passed to the `fun` handler.

Errors: None.

See also: Gpm_RaiseRoi (316), Gpm_PopRoi (316), Gpm_LowerRoi (315)

55.4.13 gpm_raiseroi

Synopsis: Raise region of interest in the stack.

Declaration: `function gpm_raiseroi(which: Pgpm_roi;before: Pgpm_roi) : Pgpm_roi`

Visibility: default

Description: `Gpm_RaiseRoi` raises the *region of interest*`which` till it is on top of region `before`. If `before` is nil then the region is put on top of the stack. The returned value is the top of the stack.

Errors: None.

See also: Gpm_PushRoi (316), Gpm_PopRoi (316), Gpm_LowerRoi (315)

55.4.14 gpm_repeat

Synopsis: Check for presence of mouse event.

Declaration: `function gpm_repeat(millisec: LongInt) : LongInt`

Visibility: default

Description: `Gpm_Repeat` returns 1 of no mouse event arrives in the next `millisec` miiliseconds, it returns 0 otherwise.

Errors: None.

See also: Gpm_GetEvent (313)

55.4.15 Gpm_StrictDouble

Synopsis: Check whether event contains only a double-click event.

Declaration: `function Gpm_StrictDouble(EventType: LongInt) : Boolean`

Visibility: default

Description: `Gpm_StrictDouble` returns true if `EventType` contains only a doubleclick event, `False` otherwise.

Errors: None.

See also: Gpm_StrictSingle (317), Gpm_AnyTriple (312), Gpm_AnyDouble (312), Gpm_StrictTriple (317), Gpm_AnySingle (312)

55.4.16 Gpm_StrictSingle

Synopsis: Check whether event contains only a single-click event.

Declaration: `function Gpm_StrictSingle(EventType: LongInt) : Boolean`

Visibility: default

Description: `Gpm_StrictDouble` returns `True` if `EventType` contains only a singleclick event, `False` otherwise.

Errors: None.

See also: Gpm_AnyTriple (312), Gpm_StrictDouble (317), Gpm_AnyDouble (312), Gpm_StrictTriple (317), Gpm_AnySingle (312)

55.4.17 Gpm_StrictTriple

Synopsis: Check whether event contains only a triple-click event.

Declaration: `function Gpm_StrictTriple(EventType: LongInt) : Boolean`

Visibility: default

Description: `Gpm_StrictTriple` returns true if `EventType` contains only a triple click event, `False` otherwise.

Errors: None.

See also: Gpm_AnyTriple (312), Gpm_StrictDouble (317), Gpm_AnyDouble (312), Gpm_StrictSingle (317), Gpm_AnySingle (312)

Chapter 56

Reference for unit 'Graph'

56.1 Overview

This document describes the GRAPH unit for Free Pascal, for all platforms. The unit was first written for dos by Florian Klaempfl, but was later completely rewritten by Carl-Eric Codere to be completely portable. The unit is provided for compatibility only: It is recommended to use more modern graphical systems. The graph unit will allow to recompile old programs. They will work to some extent, but if the application has heavy graphical needs, it's recommended to use another set of graphical routines, suited to the platform the program should work on.

56.2 Categorized functions: Text and font handling

Functions to set texts on the screen.

Table 56.1:

Name	Description
GetTextSettings (318)	Get current text settings
InstallUserFont (318)	Install a new font
OutText (318)	Write text at current cursor position
OutTextXY (318)	Write text at coordinates X,Y
RegisterBGIFont (318)	Register a new font
SetTextJustify (318)	Set text justification
SetTextStyle (318)	Set text style
SetUserCharSize (318)	Set text size
TextHeight (318)	Calculate height of text
TextWidth (318)	Calculate width of text

56.3 Categorized functions: Filled drawings

Functions for drawing filled regions.

Table 56.2:

Name	Description
Bar3D (318)	Draw a filled 3D-style bar
Bar (318)	Draw a filled rectangle
FloodFill (318)	Fill starting from coordinate
FillEllipse (318)	Draw a filled ellipse
FillPoly (318)	Draw a filled polygone
GetFillPattern (318)	Get current fill pattern
GetFillSettings (318)	Get current fill settings
SetFillPattern (318)	Set current fill pattern
SetFillStyle (318)	Set current fill settings

56.4 Categorized functions: Drawing primitives

Functions for simple drawing.

Table 56.3:

Name	Description
Arc (318)	Draw an arc
Circle (318)	Draw a complete circle
DrawPoly (318)	Draw a polygone with N points
Ellipse (318)	Draw an ellipse
GetArcCoords (318)	Get arc coordinates
GetLineSettings (318)	Get current line drawing settings
Line (318)	Draw line between 2 points
LineRel (318)	Draw line relative to current position
LineTo (318)	Draw line from current position to absolute position
MoveRel (318)	Move cursor relative to current position
MoveTo (318)	Move cursor to absolute position
PieSlice (318)	Draw a pie slice
PutPixel (318)	Draw 1 pixel
Rectangle (318)	Draw a non-filled rectangle
Sector (318)	Draw a sector
SetLineStyle (318)	Set current line drawing style

56.5 Categorized functions: Color management

All functions related to color management.

Table 56.4:

Name	Description
GetBkColor (318)	Get current background color
GetColor (318)	Get current foreground color
GetDefaultPalette (318)	Get default palette entries
GetMaxColor (318)	Get maximum valid color
GetPaletteSize (318)	Get size of palette for current mode
GetPixel (318)	Get color of selected pixel
GetPalette (318)	Get palette entry
SetAllPalette (318)	Set all colors in palette
SetBkColor (318)	Set background color
SetColor (318)	Set foreground color
SetPalette (318)	Set palette entry
SetRGBPalette (318)	Set palette entry with RGB values

56.6 Categorized functions: Screen management

General drawing screen management functions.

Table 56.5:

Name	Description
ClearViewPort (318)	Clear the current viewport
GetImage (318)	Copy image from screen to memory
GetMaxX (318)	Get maximum X coordinate
GetMaxY (318)	Get maximum Y coordinate
GetX (318)	Get current X position
GetY (318)	Get current Y position
ImageSize (318)	Get size of selected image
GetViewSettings (318)	Get current viewport settings
PutImage (318)	Copy image from memory to screen
SetActivePage (318)	Set active video page
SetAspectRatio (318)	Set aspect ratio for drawing routines
SetViewPort (318)	Set current viewport
SetVisualPage (318)	Set visual page
SetWriteMode (318)	Set write mode for screen operations

56.7 Categorized functions: Initialization

Initialization of the graphics screen.

Table 56.6:

Name	Description
ClearDevice (318)	Empty the graphics screen
CloseGraph (318)	Finish drawing session, return to text mode
DetectGraph (318)	Detect graphical modes
GetAspectRatio (318)	Get aspect ratio of screen
GetModeRange (318)	Get range of valid modes for current driver
GraphDefaults (1)	Set defaults
GetDriverName (318)	Return name of graphical driver
GetGraphMode (318)	Return current or last used graphics mode
GetMaxMode (318)	Get maximum mode for current driver
GetModeName (318)	Get name of current mode
GraphErrorMsg (1)	String representation of graphical error
GraphResult (1)	Result of last drawing operation
InitGraph (318)	Initialize graphics drivers
InstallUserDriver (318)	Install a new driver
RegisterBGIDriver (318)	Register a new driver
RestoreCRTMode (318)	Go back to text mode
SetGraphMode (318)	Set graphical mode

56.8 Target specific issues: Linux

There are several issues on Linux that need to be taken care of:

The Linux version of the Graph unit uses the libvga library. This library works on the console, not under X.

If you get an error similar to

```
/usr/bin/ld: cannot find -lvga
```

This can mean one of two things: either libvga and it's development package is not installed properly, or the directory where it is installed is not in the linker path.

To remedy the former, you should install both the libvga package and libvga-devel package (or compile and install from scratch).

To remedy the latter, you should add the path to the compiler command-line using the `-Fl` option.

Programs using libvga need root privileges to run. You can make them setuid root with the following command:

```
chown root.root myprogram
chmod u+s myprogram
```

The libvga library will give up the root privileges after it is initialized.

there is an experimental version of the Graphics library available that uses GGI to do all the drawing, but it is not well tested. It's called ggigraph and is distributed in source form only.

Do not use the CRT unit together with the Graph unit: the console may end up in an unusable state. Instead, the ncurses unit may function fine.

56.9 Target specific issues: DOS

VESA modes (i.e., anything but 320x200x256 and 640x480x16) do not work under most installations of Windows NT, Windows 2000 and Windows XP. They also do not work for some people under Windows 98 and Windows ME, depending on their graphics drivers. However, the graph unit cannot detect this, because no errors are returned from the system. In such cases, the screen simply turns black, or will show garbage.

Nothing can be done about this, the reason is missing or buggy support in the graphics drivers of the operating system.

56.10 A word about mode selection

The graph unit was implemented for compatibility with the old Turbo Pascal graph unit. For this reason, the mode constants as they were defined in the Turbo Pascal graph unit are retained.

However, since

1. Video cards have evolved very much
2. Free Pascal runs on multiple platforms

it was decided to implement new mode and graphic driver constants, which are more independent of the specific platform the program runs on.

In this section we give a short explanation of the new mode system. the following drivers were defined:

```
D1bit = 11;
D2bit = 12;
D4bit = 13;
D6bit = 14;  { 64 colors Half-brite mode - Amiga }
D8bit = 15;
D12bit = 16; { 4096 color modes HAM mode - Amiga }
D15bit = 17;
D16bit = 18;
D24bit = 19; { not yet supported }
D32bit = 20; { not yet supported }
D64bit = 21; { not yet supported }

lowNewDriver = 11;
highNewDriver = 21;
```

Each of these drivers specifies a desired color-depth.

The following modes have been defined:

```
detectMode = 30000;
m320x200 = 30001;
m320x256 = 30002; { amiga resolution (PAL) }
m320x400 = 30003; { amiga/atari resolution }
m512x384 = 30004; { mac resolution }
m640x200 = 30005; { vga resolution }
m640x256 = 30006; { amiga resolution (PAL) }
m640x350 = 30007; { vga resolution }
```

```
m640x400 = 30008;
m640x480 = 30009;
m800x600 = 30010;
m832x624 = 30011; { mac resolution }
m1024x768 = 30012;
m1280x1024 = 30013;
m1600x1200 = 30014;
m2048x1536 = 30015;

lowNewMode = 30001;
highNewMode = 30015;
```

These modes start at 30000 because Borland specified that the mode number should be ascending with increasing X resolution, and the new constants shouldn't interfere with the old ones.

The above constants can be used to set a certain color depth and resultion, as demonstrated in the below example.

If other modes than the ones above are supported by the graphics card, you will not be able to select them with this mechanism.

For this reason, there is also a 'dynamic' mode number, which is assigned at run-time. This number increases with increasing X resolution. It can be queried with the `getmoderange` call. This call will return the range of modes which are valid for a certain graphics driver. The numbers are guaranteed to be consecutive, and can be used to search for a certain resolution, as in the second example below.

Thus, the `getmoderange` function can be used to detect all available modes and drivers, as in the third example below:

Listing: ./graphex/inigraph1.pp

```
Program inigraph1;

{ Program to demonstrate static graphics mode selection }

uses graph;


const
  TheLine = 'We are now in 640 x 480 x 256 colors!'+
            ' (press <Return> to continue)';

var
  gd, gm, lo, hi, error,tw,th: integer;
  found: boolean;

begin
  { We want an 8 bit mode }
  gd := D8bit;
  gm := m640x480;
  initgraph(gd,gm,'');
  { Make sure you always check graphresult! }
  error := graphResult;
  if (error <> grOk) Then
    begin
    writeln('640x480x256 is not supported!');
    halt(1)
    end;
  { We are now in 640x480x256 }
```

```
  setColor(cyan);
  rectangle(0,0,getmaxx,getmaxy);
  { Write a nice message in the center of the screen }
  setTextStyle(defaultFont,horizDir,1);
  tw:=TextWidth(TheLine);
  th:=TextHeight(TheLine);
  outTextXY((getMaxX - TW) div 2,
            (getMaxY - TH) div 2,TheLine);
  { Wait for return }
  readln;
  { Back to text mode }
  closegraph;
end.
```

Listing: ./graphex/inigraph2.pp

```
Program inigraph2;

{ Program to demonstrate dynamic graphics mode selection }

uses graph;

const
  TheLine = 'We are now in 640 x 480 x 256 colors!'+
            ' (press <Return> to continue)';

var
  th,tw,gd, gm, lo, hi, error: integer;
  found: boolean;

begin
  { We want an 8 bit mode }
  gd := D8bit;
  { Get all available resolutions for this bitdepth }
  getmoderange(gd,lo,hi);
  { If the highest available mode number is -1,
    no resolutions are supported for this bitdepth  }
  if hi = -1 then
    begin
    writeln('no 8 bit modes supported!');
    halt
    end;
  found := false;
  { Search all resolutions for 640x480 }
  for gm := lo to hi do
    begin
    initgraph(gd,gm,'');
    { Make sure you always check graphresult! }
    error := graphResult;
    if (error = grOk) and
       (getmaxx = 639) and (getmaxy = 479) then
      begin
      found := true;
      break;
      end;
    end;
  if not found then
      CloseGraph();
    begin
```

```
    writeln('640x480x256 is not supported!');
    halt(1)
    end;
  { We are now in 640x480x256 }
  setColor(cyan);
  rectangle(0,0,getmaxx,getmaxy);
  { Write a nice message in the center of the screen }
  setTextStyle(defaultFont,horizDir,1);
  TW:=TextWidth(TheLine);
  TH:=TextHeight(TheLine);
  outTextXY((getMaxX - TW) div 2,
            (getMaxY - TH) div 2,TheLine);
  { Wait for return }
  readln;
  { Back to text mode }
  closegraph;
end.
```

Listing: ./graphex/modrange.pp

```
Program GetModeRange_Example;

{ This program demonstrates how to find all available graph modes }

uses graph;


const
  { Currently, only 4, 8, 15 and 16 bit modes are supported
    but this may change in the future }
  gdnames: array[D4bit..D16bit] of string[6] =
    ('4 bit','6 bit','8 bit','12 bit','15 bit','16 bit');

procedure WriteRes(const depth : integer);
var
  tw, th : integer;
  v, text : String;
begin
  text := 'Current resolution is '; str(getmaxx+1, v);
  text := text + v + 'x'; str(getmaxy+1, v);
  text := text + v + 'x' + gdnames[depth];
  setTextStyle(defaultFont,horizDir,1);
  TW:=TextWidth(text);
  TH:=TextHeight(text);
  outTextXY((getMaxX - TW) div 2,
            (getMaxY - TH) div 2,text);
end;

var
  t: text;
  line : string;
  gd, c, low, high, res: integer;
begin
  assign(t,'modes.txt');
  rewrite(t);
  close(t);
  for gd := D4bit to D16bit do
    begin
    { Get the available mode numbers for this driver }
```

```
    getModeRange(gd,low,high);
    append(t);
    write(t,gdnames[gd]);
    Writeln(t,': low modenr = ',low,', high modenr = ',high);
    close(t);
    { If high is -1,
       no resolutions are supported for this bitdepth }
    if high = -1 then
      begin
      append(t);
      writeln(t,'  No modes supported!');
      writeln(t);
      close(t);
      end
    else
      { Enter all supported resolutions for this bitdepth
        and write their characteristics to the file }
      for c := low to high do
        begin
        append(t);
        writeln(t,'  testing mode nr ',c);
        close(t);
        initgraph(gd,c,'');
        res := graphresult;
        append(t);
        { An error occurred when entering the mode? }
        if res <> grok then
          writeln(t,grapherrormsg(res))
        else
          begin
          write(t,'maxx: ',getmaxx,', maxy: ',getmaxy);
          Writeln(t,', maxcolor: ',getmaxcolor);
          closegraph;
          end;
        writeln(t);
          WriteRes(gd);
        close(t);
        end;
    append(t);
    writeln(t);
    close(t);
    end;
  Writeln('All supported modes are listed in modes.txt files');
end.
```

56.11 Requirements

The unit Graph exports functions and procedures for graphical output. It requires at least a VGA-compatible Card or a VGA-Card with software-driver (min. **512Kb** video memory).

Chapter 57

Reference for unit 'heaptrc'

57.1 Overview

This document describes the HEAPTRC unit for Free Pascal. It was written by Pierre Muller. It is system independent, and works on all supported systems.

The HEAPTRC unit can be used to debug your memory allocation/deallocation. It keeps track of the calls to getmem/freemem, and, implicitly, of New/Dispose statements.

When the program exits, or when you request it explicitly. It displays the total memory used, and then dumps a list of blocks that were allocated but not freed. It also displays where the memory was allocated.

If there are any inconsistencies, such as memory blocks being allocated or freed twice, or a memory block that is released but with wrong size, this will be displayed also.

The information that is stored/displayed can be customized using some constants.

57.2 Controlling HeapTrc with environment variables

The HeapTrc unit can be controlled with the `HEAPTRC` environment variable. The contents of this variable controls the initial setting of some constants in the unit. `HEAPTRC` consists of one or more of the following strings, separated by spaces:

keepreleased If this string occurs, then the KeepReleased (329) variable is set to `True`

disabled If this string occurs, then the UseHeapTrace (330) variable is set to `False` and the heap trace is disabled. It does not make sense to combine this value with other values.

nohalt If this string occurs, then the HaltOnError (329) variable is set to `False`, so the program continues executing even in case of a heap error.

log=filename If this string occurs, then the output of heaptrc is sent to the specified `Filename`. (see also SetHeapTraceOutput (332))

The following are valid values for the HEAPTRC variable:

```
HEAPTRC=disabled
HEAPTRC="keepreleased log=heap.log"
HEAPTRC="log=myheap.log nohalt"
```

Note that these strings are case sensitive, and the name of the variable too.

57.3 HeapTrc Usage

All that you need to do is to include heaptrc in the uses clause of your program. Make sure that it is the first unit in the clause, otherwise memory allocated in initialization code of units that precede the heaptrc unit will not be accounted for, causing an incorrect memory usage report.

If you use the `-gh` switch, the compiler will insert the unit by itself, so you don't have to include it in your uses clause.

The below example shows how to use the heaptrc unit.

This is the memory dump shown when running this program in a standard way:

```
Marked memory at 0040FA50 invalid
Wrong size : 128 allocated 64 freed
  0x00408708
  0x0040CB49
  0x0040C481
Call trace for block 0x0040FA50 size 128
  0x0040CB3D
  0x0040C481
```

If you use the lineinfo unit (or use the `-gl` switch) as well, then heaptrc will also give you the filenames and line-numbers of the procedures in the backtrace:

```
Marked memory at 00410DA0 invalid
Wrong size : 128 allocated 64 freed
  0x004094B8
  0x0040D8F9  main,  line 25 of heapex.pp
  0x0040D231
Call trace for block 0x00410DA0 size 128
  0x0040D8ED  main,  line 23 of heapex.pp
  0x0040D231
```

If lines without filename/line-number occur, this means there is a unit which has no debug info included.

Listing: ./heapex/heapex.pp

```
Program heapex;

{ Program used to demonstrate the usage of heaptrc unit }

Uses heaptrc;

Var P1 : ^Longint;
    P2 : Pointer;
    I : longint;

begin
  New(P1);
  // causes previous allocation not to be de-allocated
  New(P1);
  Dispose(P1);
  For I:=1 to 10 do
    begin
    GetMem (P2,128);
    // When I is even, deallocate block. We loose 5 times 128
```

```
    // bytes this way.
    If (I mod 2) = 0 Then FreeMem(P2,128);
    end;
  GetMem(P2,128);
  // This will provoke an error and a memory dump
  Freemem (P2,64);
end.
```

57.4 Constants, types and variables

57.4.1 Constants

`add_tail : Boolean = True`

If `add\_tail` is `True` (the default) then a check is also performed on the memory location just behind the allocated memory.

`HaltOnError : Boolean = True`

If `HaltOnError` is set to `True` then an illegal call to `FreeMem` will cause the memory manager to execute a `halt(1)` instruction, causing a memory dump. By Default it is set to `True`.

`HaltOnNotReleased : Boolean = False`

`HaltOnNotReleased` can be set to `True` to make the DumpHeap (331) procedure halt (exit code 203) the program if any memory was not released when the dump is made. If it is `False` (the default) then `DumpHeap` just returns.

`keepreleased : Boolean = False`

If `keepreleased` is set to true, then a list of freed memory blocks is kept. This is useful if you suspect that the same memory block is released twice. However, this option is very memory intensive, so use it sparingly, and only when it's really necessary.

`maxprintedblocklength : Integer = 128`

`maxprintedblocklength` determines the maximum number of bytes written by a memory block dump, as produced when printleakedblock (330) or printfaultyblock (329) are true. If the size of the memory block is larger than this size, then only the first `maxprintedblocklength` will be included in the dump.

`printfaultyblock : Boolean = False`

`printleakedblock` can be set to `True` to print a memory dump of daulty memory blocks (in case a memory override occurs) The block is printed as a series of hexadecimal numbers, representing the bytes in the memory block. At most maxprintedblocklength (329) bytes of the memory block will be printed.

`printleakedblock : Boolean = False`

printleakedblock can be set to True to print a memory dump of unreleased blocks when the heaptrc unit produces a summary of memory leaks. The block is printed as a series of hexadecimal numbers, representing the bytes in the memory block. At most maxprintedblocklength (329) bytes of the memory block will be printed.

```
quicktrace : Boolean = True
```

Quicktrace determines whether the memory manager checks whether a block that is about to be released is allocated correctly. This is a rather time consuming search, and slows program execution significantly, so by default it is set to True.

```
tracesize = 8
```

Tracesize specifies how many levels of calls are displayed of the call stack during the memory dump. If you specify keepreleased:=True then half the TraceSize is reserved for the GetMem call stack, and the other half is reserved for the FreeMem call stack. For example, the default value of 8 will cause eight levels of call frames to be dumped for the getmem call if keepreleased is False. If KeepReleased is true, then 4 levels of call frames will be dumped for the GetMem call and 4 frames wil be dumped for the FreeMem call. If you want to change this value, you must recode the heaptrc unit.

```
usecrc : Boolean = True
```

If usecrc is True (the default) then a crc check is performed on locations before and after the allocated memory. This is useful to detect memory overwrites.

```
useheaptrace : Boolean = True
```

This variable must be set at program startup, through the help of an environment variable.

57.4.2 Types

```
tdisplayextrainfoProc = procedure(var ptext: text;p: pointer)
```

The TDisplayExtraInfoType is a procedural type used in the SetHeapExtraInfo (331) call to display a memory location which was previously filled with TFillExtraInfoProc (330)

```
tFillExtraInfoProc = procedure(p: pointer)
```

The TFillExtraInfoProc is a procedural type used in the SetHeapExtraInfo (331) call to fill a memory location with extra data for displaying.

57.5 Procedures and functions

57.5.1 CheckPointer

Synopsis: Check if a pointer is in the address range of the application

Declaration: `procedure CheckPointer(p: pointer)`

Visibility: default

Description: CheckPointer checks if the pointer is in the address range of the application, more specifically, if it is in the heap. if not, it prints an error and stops the program with aruntime error 204.

57.5.2 DumpHeap

Synopsis: Dump memory usage report to stderr.

Declaration: `procedure DumpHeap`

Visibility: default

Description: `DumpHeap` dumps to standard output a summary of memory usage. It is called automatically by the heaptrc unit when your program exits (by installing an exit procedure), but it can be called at any time.

Errors: None.

57.5.3 SetHeapExtraInfo

Synopsis: Store extra information in blocks.

Declaration:
```
procedure SetHeapExtraInfo(size: ptruint;fillproc: tFillExtraInfoProc;
                           displayproc: tdisplayextrainfoProc)
```

Visibility: default

Description: You can use `SetHeapExtraInfo` to store extra info in the blocks that the heaptrc unit reserves when tracing getmem calls. `Size` indicates the size (in bytes) that the trace mechanism should reserve for your extra information. For each call to `getmem`, `FillProc` will be called, and passed a pointer to the memory reserved.

When dumping the memory summary, the extra info is shown by calling `displayproc` and passing it the memory location which was filled by `fillproc`. It should write the information in readable form to the text file provided in the call to `displayproc`

Errors: You can only call `SetHeapExtraInfo` if no memroy has been allocated yet. If memory was already allocated prior to the call to `SetHeapExtraInfo`, then an error will be displayed on standard error output, and a DumpHeap (331) is executed.

See also: DumpHeap (331), SetHeapTraceOutput (332)

Listing: ./heapex/setinfo.pp

```
Program heapex;

{ Program used to demonstrate the usage of heaptrc unit }

Uses heaptrc;

Var P1 : ^Longint;
    P2 : Pointer;
    I : longint;
    Marker : Longint;

Procedure SetMarker (P : pointer);

Type PLongint = ^Longint;

begin
  PLongint(P)^:=Marker;
end;
```

```
Procedure  Part1;

begin
  // Blocks allocated here are marked with $FFAAFFAA = -5570646
  Marker := $FFAAFFAA;
  New(P1);
  New(P1);
  Dispose(P1);
  For I:=1 to 10 do
    begin
    GetMem (P2,128);
    If (I mod 2) = 0 Then FreeMem(P2,128);
    end;
  GetMem(P2,128);
end;

Procedure  Part2;

begin
  // Blocks allocated here are marked with $FAFAFAFA = -84215046
  Marker := $FAFAFAFA;
  New(P1);
  New(P1);
  Dispose(P1);
  For I:=1 to 10 do
    begin
    GetMem (P2,128);
    If (I mod 2) = 0 Then FreeMem(P2,128);
    end;
  GetMem(P2,128);
end;

begin
 SetExtraInfo(SizeOf(Marker),@SetMarker);
 Writeln ('Part 1');
 part1;
 Writeln('Part 2');
 part2;
end.
```

57.5.4 SetHeapTraceOutput

Synopsis: Specify filename for heap trace output.

Declaration:
```
procedure SetHeapTraceOutput(const name: string);  Overload
procedure SetHeapTraceOutput(var ATextOutput: Text);  Overload
```

Visibility: default

Description: `SetHeapTraceOutput` sets the filename into which heap trace info will be written. By default information is written to standard output, this function allows you to redirect the information to a file with full filename `name`.

Errors: If the file cannot be written to, errors will occur when writing the trace.

See also: SetHeapExtraInfo (331)

Chapter 58

Reference for unit 'ipc'

58.1 Used units

Table 58.1: Used units by unit 'ipc'

Name	Page
BaseUnix	51
System	622
unixtype	747

58.2 Overview

This document describes the IPC unit for Free Pascal. It was written for linux by Michael Van Canneyt. It gives all the functionality of System V Inter-Process Communication: shared memory, semaphores and messages. It works only on the linux operating system.

Many constants here are provided for completeness only, and should under normal circumstances not be used by the programmer.

Chapter 59

Reference for unit 'keyboard'

59.1 Overview

The Keyboard unit implements a keyboard access layer which is system independent. It can be used to poll the keyboard state and wait for certain events. Waiting for a keyboard event can be done with the GetKeyEvent (334) function, which will return a driver-dependent key event. This key event can be translated to a interpretable event by the TranslateKeyEvent (334) function. The result of this function can be used in the other event examining functions.

A custom keyboard driver can be installed using the SetKeyboardDriver (334) function. The current keyboard driver can be retrieved using the GetKeyboardDriver (334) function. The last section of this chapter demonstrates how to make a keyboard driver.

59.2 Unix specific notes

On Unix, applications run on a "terminal", and the application writes to the screen and reads from the keyboard by communicating with the terminal. Unix keyboard handling is mostly backward compatible with the DEC vt100 and vt220 terminals from tens of years ago. The vt100 and vt220 had very different keyboards than todays PC's and this is where the problems start. To make it worse the protocol of both terminals has not been very well designed.

Because of this, the keyboard unit on Unix operating systems does a best effort to provide keyboard functionality. An implementation with full keyboard facilities like on other operating systems is not possible.

The exception is the Linux kernel. The terminal emulation of the Linux kernel is from a PC keyboard viewpoint hopeless as well, but unlike other terminal emulators it is configurable. On the Linux console, the Free Pascal keyboard unit tries to implement full functionality.

Users of applications using the keyboard unit should expect the following:

- Full functionality on the Linux console. It must be the bare console, SSH into another machine will kill the full functionality.
- Limited functionality otherwise.

Notes about Linux full functionality:

- The keyboard is reprogrammed. If the keyboard is for whatever reason not restored in its original state, please load your keymap to reinitialize it.

- Alt+function keys generate keycodes for those keys. To switch virtual consoles, use ctrl+alt+function key.
- Unlike what you're used to with other Unix software, escape works as you intuitively expect, it generates the keycode for an escape key **without a delay.**

The limited functionality does include these quirks:

- Escape must be pressed two times before it has effect.
- On the Linux console, when the users runs the program by logging into another machine:
  - Shift+F1 and Shift+F12 will generate keycodes for F11 and F12.
  - Shift+arrow keys, shift+ins, shift+del, shift+home, shift+end do not work. The same is true about the control and alt combinations.
  - Alt+function keys will switch virtual consoles instead of generating the right key sequences.
  - Ctrl+function keys will generate the keycodes for the function keys without ctrl
- In Xterm:
  - Shift+insert pastes the x clipboard, no keycode will be generated.
- In Konsole:
  - Shift+insert pastes the x clipboard, no keycode will be generated.
  - Shift+arrow keys doesn't work, nor does ctrl+arrow keys

If you have a non-standard terminal, some keys may not work at all. When in limited functionality mode, the user can work around using an escape prefix:

- Esc+1 = F1, Esc+2 = F2.
- Esc before another key is equal to alt+key.

In such cases, if the terminal does output an escape sequence for those keys, please submit a bug report so we can add them.

59.3 Writing a keyboard driver

Writing a keyboard driver means that hooks must be created for most of the keyboard unit functions. The `TKeyBoardDriver` record contains a field for each of the possible hooks:

```
TKeyboardDriver = Record
  InitDriver : Procedure;
  DoneDriver : Procedure;
  GetKeyEvent : Function : TKeyEvent;
  PollKeyEvent : Function : TKeyEvent;
  GetShiftState : Function : Byte;
  TranslateKeyEvent : Function (KeyEvent: TKeyEvent): TKeyEvent;
  TranslateKeyEventUniCode: Function (KeyEvent: TKeyEvent): TKeyEvent;
end;
```

The meaning of these hooks is explained below:

InitDriver Called to initialize and enable the driver. Guaranteed to be called only once. This should initialize all needed things for the driver.

DoneDriver Called to disable and clean up the driver. Guaranteed to be called after a call to `initDriver`. This should clean up all things initialized by `InitDriver`.

GetKeyEvent Called by GetKeyEvent (334). Must wait for and return the next key event. It should NOT store keys.

PollKeyEvent Called by PollKeyEvent (334). It must return the next key event if there is one. Should not store keys.

GetShiftState Called by PollShiftStateEvent (334). Must return the current shift state.

TranslateKeyEvent Should translate a raw key event to a cOrrect key event, i.e. should fill in the shiftstate and convert function key scancodes to function key keycodes. If the `TranslateKeyEvent` is not filled in, a default translation function will be called which converts the known scancodes from the tables in the previous section to a correct keyevent.

TranslateKeyEventUniCode Should translate a key event to a unicode key representation.

Strictly speaking, only the `GetKeyEvent` and `PollKeyEvent` hooks must be implemented for the driver to function correctly.

The example unit demonstrates how a keyboard driver can be installed. It takes the installed driver, and hooks into the `GetKeyEvent` function to register and log the key events in a file. This driver can work on top of any other driver, as long as it is inserted in the `uses` clause *after* the real driver unit, and the real driver unit should set the driver record in its initialization section.

Note that with a simple extension of this unit could be used to make a driver that is capable of recording and storing a set of keyboard strokes, and replaying them at a later time, so a 'keyboard macro' capable driver. This driver could sit on top of any other driver.

Listing: ./kbdex/logkeys.pp

```
unit logkeys;

interface

Procedure StartKeyLogging;
Procedure StopKeyLogging;
Function  IsKeyLogging : Boolean;
Procedure  SetKeyLogFileName(FileName : String);


implementation

uses sysutils,keyboard;

var
  NewKeyBoardDriver,
  OldKeyBoardDriver : TKeyboardDriver;
  Active,Logging : Boolean;
  LogFileName : String;
  KeyLog : Text;

Function TimeStamp : String;
```

```
begin
  TimeStamp:=FormatDateTime('hh:nn:ss',Time());
end;

Procedure StartKeyLogging;

begin
  Logging:=True;
  Writeln(KeyLog,'Start logging keystrokes at: ',TimeStamp);
end;

Procedure StopKeyLogging;

begin
  Writeln(KeyLog,'Stop logging keystrokes at: ',TimeStamp);
  Logging:=False;
end;

Function IsKeyLogging : Boolean;

begin
  IsKeyLogging:=Logging;
end;

Function LogGetKeyEvent : TKeyEvent;

Var
  K : TKeyEvent;

begin
  K:=OldkeyboardDriver.GetKeyEvent();
  If Logging then
    begin
    Write(KeyLog,TimeStamp,': Key event: ');
    Writeln(KeyLog,KeyEventToString(TranslateKeyEvent(K)));
    end;
  LogGetKeyEvent:=K;
end;

Procedure LogInitKeyBoard;

begin
  OldKeyBoardDriver.InitDriver();
  Assign(KeyLog,logFileName);
  Rewrite(KeyLog);
  Active:=True;
  StartKeyLogging;
end;

Procedure LogDoneKeyBoard;

begin
  StopKeyLogging;
  Close(KeyLog);
  Active:=False;
  OldKeyBoardDriver.DoneDriver();
end;
```

```
Procedure SetKeyLogFileName(FileName : String);

begin
  If Not Active then
    LogFileName:=FileName;
end;

Initialization
  GetKeyBoardDriver(OldKeyBoardDriver);
  NewKeyBoardDriver:=OldKeyBoardDriver;
  NewKeyBoardDriver.GetKeyEvent:=@LogGetKeyEvent;
  NewKeyBoardDriver.InitDriver:=@LogInitKeyboard;
  NewKeyBoardDriver.DoneDriver:=@LogDoneKeyboard;
  LogFileName:='keyboard.log';
  Logging:=False;
  SetKeyboardDriver(NewKeyBoardDriver);
end.
```

Listing: ./kbdex/ex9.pp

```
program example9;

{ This program demonstrates the logkeys unit }

uses keyboard,logkeys;

Var
  K : TKeyEvent;

begin
  InitKeyBoard;
  Writeln('Press keys, press "q" to end, "s" toggles logging.');
  Repeat
    K:=GetKeyEvent;
    K:=TranslateKeyEvent(K);
    Writeln('Got key : ',KeyEventToString(K));
    if GetKeyEventChar(K)='s' then
      if IsKeyLogging then
        StopKeyLogging
      else
        StartKeyLogging;
  Until (GetKeyEventChar(K)='q');
  DoneKeyBoard;
end.
```

59.4 Keyboard scan codes

Special physical keys are encoded with the DOS scan codes for these keys in the second byte of the TKeyEvent (334) type. A complete list of scan codes can be found in the below table. This is the list of keys that is used by the default key event translation mechanism. When writing a keyboard driver, either these constants should be returned by the various key event functions, or the `TranslateKeyEvent` hook should be implemented by the driver.

Table 59.1: Key Scancodes

Code	Key	Code	Key	Code	Key
00	NoKey	3D	F3	70	ALT-F9
01	ALT-Esc	3E	F4	71	ALT-F10
02	ALT-Space	3F	F5	72	CTRL-PrtSc
04	CTRL-Ins	40	F6	73	CTRL-Left
05	SHIFT-Ins	41	F7	74	CTRL-Right
06	CTRL-Del	42	F8	75	CTRL-end
07	SHIFT-Del	43	F9	76	CTRL-PgDn
08	ALT-Back	44	F10	77	CTRL-Home
09	ALT-SHIFT-Back	47	Home	78	ALT-1
0F	SHIFT-Tab	48	Up	79	ALT-2
10	ALT-Q	49	PgUp	7A	ALT-3
11	ALT-W	4B	Left	7B	ALT-4
12	ALT-E	4C	Center	7C	ALT-5
13	ALT-R	4D	Right	7D	ALT-6
14	ALT-T	4E	ALT-GrayPlus	7E	ALT-7
15	ALT-Y	4F	end	7F	ALT-8
16	ALT-U	50	Down	80	ALT-9
17	ALT-I	51	PgDn	81	ALT-0
18	ALT-O	52	Ins	82	ALT-Minus
19	ALT-P	53	Del	83	ALT-Equal
1A	ALT-LftBrack	54	SHIFT-F1	84	CTRL-PgUp
1B	ALT-RgtBrack	55	SHIFT-F2	85	F11
1E	ALT-A	56	SHIFT-F3	86	F12
1F	ALT-S	57	SHIFT-F4	87	SHIFT-F11
20	ALT-D	58	SHIFT-F5	88	SHIFT-F12
21	ALT-F	59	SHIFT-F6	89	CTRL-F11
22	ALT-G	5A	SHIFT-F7	8A	CTRL-F12
23	ALT-H	5B	SHIFT-F8	8B	ALT-F11
24	ALT-J	5C	SHIFT-F9	8C	ALT-F12
25	ALT-K	5D	SHIFT-F10	8D	CTRL-Up
26	ALT-L	5E	CTRL-F1	8E	CTRL-Minus
27	ALT-SemiCol	5F	CTRL-F2	8F	CTRL-Center
28	ALT-Quote	60	CTRL-F3	90	CTRL-GreyPlus
29	ALT-OpQuote	61	CTRL-F4	91	CTRL-Down
2B	ALT-BkSlash	62	CTRL-F5	94	CTRL-Tab
2C	ALT-Z	63	CTRL-F6	97	ALT-Home
2D	ALT-X	64	CTRL-F7	98	ALT-Up
2E	ALT-C	65	CTRL-F8	99	ALT-PgUp
2F	ALT-V	66	CTRL-F9	9B	ALT-Left
30	ALT-B	67	CTRL-F10	9D	ALT-Right
31	ALT-N	68	ALT-F1	9F	ALT-end
32	ALT-M	69	ALT-F2	A0	ALT-Down
33	ALT-Comma	6A	ALT-F3	A1	ALT-PgDn
34	ALT-Period	6B	ALT-F4	A2	ALT-Ins
35	ALT-Slash	6C	ALT-F5	A3	ALT-Del
37	ALT-GreyAst	6D	ALT-F6	A5	ALT-Tab
3B	F1	6E	ALT-F7		
3C	F2	6F	ALT-F8		

A list of scan codes for special keys and combinations with the SHIFT, ALT and CTRL keys can be found in the following table: They are for quick reference only.

Table 59.2: Special keys scan codes

Key	Code	SHIFT-Key	CTRL-Key	Alt-Key
NoKey	00			
F1	3B	54	5E	68
F2	3C	55	5F	69
F3	3D	56	60	6A
F4	3E	57	61	6B
F5	3F	58	62	6C
F6	40	59	63	6D
F7	41	5A	64	6E
F8	42	5B	65	6F
F9	43	5C	66	70
F10	44	5D	67	71
F11	85	87	89	8B
F12	86	88	8A	8C
Home	47		77	97
Up	48		8D	98
PgUp	49		84	99
Left	4B		73	9B
Center	4C		8F	
Right	4D		74	9D
end	4F		75	9F
Down	50		91	A0
PgDn	51		76	A1
Ins	52	05	04	A2
Del	53	07	06	A3
Tab	8	0F	94	A5
GreyPlus			90	4E

Chapter 60

Reference for unit 'lineinfo'

60.1 Overview

The lineinfo provides a routine that reads the debug information of an executable (if any exists) and returns source code information about this address. It works with `Stabs` debug information. Note that this unit is not thread-safe, and that its behaviour is undefined if multiple threads try to write a backtrace at the same time.

For DWARF debug information, the lnfodwrf (370) unit must be used.

60.2 Procedures and functions

60.2.1 CloseStabs

Declaration: `procedure CloseStabs`

Visibility: default

60.2.2 GetLineInfo

Synopsis: Return source line information about an address.

Declaration:
```
function GetLineInfo(addr: ptruint;var func: string;var source: string;
                     var line: LongInt) : Boolean
```

Visibility: default

Description: `GetLineInfo` returns source line information about the address `addr`. It searches this information in the stabs debugging information found in the binary: If the file was compiled without debug information, nothing will be returned. Upon succesful retrieval of the debug information, `True` is returned, and the `func` parameter is filled with the name of the function in which the address is located. The `source` parameter contains the name of the file in which the function was implemented, and `line` contains the line number in the source file for `addr`.

Errors: If no debug information is found, `False` is returned.

60.2.3 StabBackTraceStr

Declaration: `function StabBackTraceStr(addr: CodePointer) : string`

Visibility: default

Chapter 61

Reference for unit 'Linux'

61.1 Used units

Table 61.1: Used units by unit 'Linux'

Name	Page
BaseUnix	51
System	622
unixtype	747

61.2 Overview

The linux unit contains linux specific operating system calls.

The platform independent functionality of the FPC 1.0.X version of the linux unit has been split out over the unix (734), baseunix (51) and unixutil (748) units.

The X86-specific parts have been moved to the X86 (765) unit.

61.3 Constants, types and variables

61.3.1 Constants

```
CAP_AUDIT_CONTROL = 30
```

Allow manipulation of kernel auditing features

```
CAP_AUDIT_WRITE = 29
```

Allow writing to kernel audit log

```
CAP_CHOWN = 0
```

Perform chown operation

```
CAP_DAC_OVERRIDE = 1
```

Bypass file operation (rwx) checks

```
CAP_DAC_READ_SEARCH = 2
```

Bypass file read-only operation checks

```
CAP_FOWNER = 3
```

Bypass owner ID checks

```
CAP_FSETID = 4
```

Do not clear SUID/GUID bits on modified files

```
CAP_FS_MASK = $1f
```

?

```
CAP_IPC_LOCK = 14
```

Allow memory locking calls

```
CAP_IPC_OWNER = 15
```

Bypass permission checks on IPC operations

```
CAP_KILL = 5
```

Bypass permission checks for sending signals

```
CAP_LEASE = 28
```

Allow file leases

```
CAP_LINUX_IMMUTABLE = 9
```

Allow setting ext2 file attributes

```
CAP_MKNOD = 27
```

Allow creation of special files through mknod calls

```
CAP_NET_ADMIN = 12
```

Allow network operations (e.g. setting socket options)

```
CAP_NET_BIND_SERVICE = 10
```

Allow binding to ports less than 1024

`CAP_NET_BROADCAST = 11`

Allow socket broadcast operations

`CAP_NET_RAW = 13`

Allow use of RAW and PACKET sockets

`CAP_SETGID = 6`

Allow GID manipulations

`CAP_SETPCAP = 8`

Allow to set other process' capabilities

`CAP_SETUID = 7`

Allow process ID manipulations

`CAP_SYS_ADMIN = 21`

Allow various system administration calls

`CAP_SYS_BOOT = 22`

Allow reboot calls

`CAP_SYS_CHROOT = 18`

Allow chroot calls.

`CAP_SYS_MODULE = 16`

Allow loading/unloading of kernel modules

`CAP_SYS_NICE = 23`

Allowing raising process and thread priorities

`CAP_SYS_PACCT = 20`

Allow acct calls

`CAP_SYS_PTRACE = 19`

Allow ptrace calls

`CAP_SYS_RAWIO = 17`

Allow raw I/O port operations

`CAP_SYS_RESOURCE = 24`

Allow use of special resources or raising of resource limits

`CAP_SYS_TIME = 25`

Allow system or real-time clock modification

`CAP_SYS_TTY_CONFIG = 26`

Allow vhangup calls

`CLOCKS_MASK = CLOCK_REALTIME or CLOCK_MONOTONIC`

Mask for supported clocks

`CLOCKS_MONO = CLOCK_MONOTONIC`

Monotonic clocks mask

`CLOCK_MONOTONIC = 1`

Monotonic system time since some undetermined start point. Can change if time is set.

`CLOCK_MONOTONIC_COARSE = 6`

Less precise (but faster) version of `CLOCK_MONOTONIC`

`CLOCK_MONOTONIC_RAW = 4`

Like `CLOCK_MONOTONIC`, not subject to NTP adjustments

`CLOCK_PROCESS_CPUTIME_ID = 2`

Processs-specific high-resolution timer from the CPU.

`CLOCK_REALTIME = 0`

System wide real-time clock. Can only be set by root.

`CLOCK_REALTIME_COARSE = 5`

Less precise (but faster) version of `CLOCK_REALTIME`

`CLOCK_SGI_CYCLE = 10`

High resolution timer

`CLOCK_THREAD_CPUTIME_ID = 3`

Thread-specific high-resolution timer from the CPU.

```
CLONE_CHILD_CLEARTID = $00200000
```

Clone option: Erase child thread ID in child memory space when child exits.

```
CLONE_CHILD_SETTID = $01000000
```

Clone option: Store child thread ID in child memory.

```
CLONE_DETACHED = $00400000
```

Clone option: Start clone detached.

```
CLONE_FILES = $00000400
```

Clone (361) option: open files shared between processes

```
CLONE_FS = $00000200
```

Clone (361) option: fs info shared between processes

```
CLONE_NEWNS = $00020000
```

Clone options: Start child in new (filesystem) namespace.

```
CLONE_PARENT = $00008000
```

Clone options: Set child parent to parent of calling process.

```
CLONE_PARENT_SETTID = $00100000
```

Clone option: Store child thread ID in memory in both parent and child.

```
CLONE_PID = $00001000
```

Clone (361) option: PID shared between processes

```
CLONE_PTRACE = $00002000
```

Clone options: if parent is traced, trace child also

```
CLONE_SETTLS = $00080000
```

Clone option: The newtls parameter is the TLS descriptor of the child

```
CLONE_SIGHAND = $00000800
```

Clone (361) option: signal handlers shared between processes

```
CLONE_STOPPED = $02000000
```

Clone option: Start child in stopped state.

CLONE_SYSVSEM = $00040000

Clone option: Caller and child share the same semaphore undo values

CLONE_THREAD = $00010000

Clone options: Set child in thread group of calling process.

CLONE_UNTRACED = $00800000

Clone option: Do not allow a ptrace call on this clone.

CLONE_VFORK = $00004000

Clone options: suspend parent till child execs

CLONE_VM = $00000100

Clone (361) option: VM shared between processes

CSIGNAL = $000000ff

Clone (361) option: Signal mask to be sent at exit

EPOLLERR = $08

event_wait error condition on file descriptor

EPOLLET = $80000000

Set event_wait edge trigger behaviour on file descriptor

EPOLLHUP = $10

event_wait hang up event

EPOLLIN = $01

event_wait input file descriptor ready event

EPOLLONESHOT = $40000000

Set single-shot behaviour on epoll_wait.

EPOLLOUT = $04

event_wait output file descriptor ready event

EPOLLPRI = $02

event_wait high priority data available on input file descriptor

```
EPOLL_CTL_ADD = 1
```

Add filedescriptor to list of events

```
EPOLL_CTL_DEL = 2
```

Delete event for filedescriptor

```
EPOLL_CTL_MOD = 3
```

Modify event for filedescriptor

```
FUTEX_CMP_REQUEUE = 4
```

Futex option: requeue waiting processes on other futex, but check it's value first

```
FUTEX_FD = 2
```

Futex option: Associate file descriptor with futex.

```
FUTEX_LOCK_PI = 6
```

Futex option: Undocumented

```
FUTEX_OP_ADD = 1
```

Futex operation: Undocumented

```
FUTEX_OP_ANDN = 3
```

Futex operation: Undocumented

```
FUTEX_OP_CMP_EQ = 0
```

Futex operation: Undocumented

```
FUTEX_OP_CMP_GE = 5
```

Futex operation: Undocumented

```
FUTEX_OP_CMP_GT = 4
```

Futex operation: Undocumented

```
FUTEX_OP_CMP_LE = 3
```

Futex operation: Undocumented

```
FUTEX_OP_CMP_LT = 2
```

Futex operation: Undocumented

```
FUTEX_OP_CMP_NE = 1
```

Futex operation: Undocumented

```
FUTEX_OP_OPARG_SHIFT = 8
```

Futex operation: Undocumented

```
FUTEX_OP_OR = 2
```

Futex operation: Undocumented

```
FUTEX_OP_SET = 0
```

Futex operation: Undocumented

```
FUTEX_OP_XOR = 4
```

Futex operation: Undocumented

```
FUTEX_REQUEUE = 3
```

Futex option: requeue waiting processes on other futex.

```
FUTEX_TRYLOCK_PI = 8
```

Futex option: Undocumented

```
FUTEX_UNLOCK_PI = 7
```

Futex option: Undocumented

```
FUTEX_WAIT = 0
```

Futex option: Wait on futex till wake call arrives.

```
FUTEX_WAKE = 1
```

Futex option: wakes any waiting processes on this futex

```
FUTEX_WAKE_OP = 5
```

Futex option: Undocumented

```
GIO_CMAP = $4B70
```

IOCTL: Get colour palette on VGA+

```
GIO_FONT = $4B60
```

IOCTL: Get font in expanded form.

```
GIO_FONTX = $4B6B
```

IOCTL: Get font in `consolefontdesc` record.

```
GIO_SCRNMAP = $4B40
```

IOCTL: get screen mapping from kernel

```
GIO_UNIMAP = $4B66
```

IOCTL: get unicode-to-font mapping from kernel

```
GIO_UNISCRNMAP = $4B69
```

IOCTL: get full Unicode screen mapping

```
IN_ACCESS = $00000001
```

Data was read from file.

```
IN_ALL_EVENTS = IN_ACCESS or IN_MODIFY or IN_ATTRIB or IN_CLOSE or IN_OPEN or IN_MOVI
```

All possible events OR-ed together.

```
IN_ATTRIB = $00000004
```

File attributes changed.

```
IN_CLOEXEC = &02000000
```

`IN_CLOEXEC` can be set to indicate that the inotify file handle must be closed on exec.

```
IN_CLOSE = IN_CLOSE_WRITE or IN_CLOSE_NOWRITE
```

File was closed (read or write)

```
IN_CLOSE_NOWRITE = $00000010
```

File opened for read was closed

```
IN_CLOSE_WRITE = $00000008
```

File opened for write was closed

```
IN_CREATE = $00000100
```

A file was created in the directory.

```
IN_DELETE = $00000200
```

A file was deleted from the directory.

```
IN_DELETE_SELF = $00000400
```

Directory or file under observation was deleted.

```
IN_DONT_FOLLOW = $02000000
```

Do not follow symlinks

```
IN_IGNORED = $00008000
```

Watch was ignored (removed). Only reported.

```
IN_ISDIR = $40000000
```

Event subject is a directory (reported only)

```
IN_MASK_ADD = $20000000
```

Add events to existing watch (OR-ing the sets) if one exists.

```
IN_MODIFY = $00000002
```

Data was written to file.

```
IN_MOVE = IN_MOVED_FROM or IN_MOVED_TO
```

File was moved (in or out of directory)

```
IN_MOVED_FROM = $00000040
```

File was moved away from watched directory

```
IN_MOVED_TO = $00000080
```

File was moved into watched directory

```
IN_MOVE_SELF = $00000800
```

Directory or file under observation was moved.

```
IN_NONBLOCK = &00004000
```

`IN_NONBLOCK` can be set to indicate that the inotify file handle should not block read operations.

```
IN_ONESHOT = $80000000
```

Only report one event, then remove the watch.

```
IN_ONLYDIR = $01000000
```

Only watch filename if it is a directory.

```
IN_OPEN = $00000020
```

File was opened

```
IN_Q_OVERFLOW = $00004000
```

Queue overflowed. Only reported.

```
IN_UNMOUNT = $00002000
```

File system on which file resides was unmounted. Only reported.

```
KB_101 = 2
```

IOCTL: Keyboard types: 101 keys

```
KB_84 = 1
```

IOCTL: Keyboard types: 84 keys

```
KB_OTHER = 3
```

IOCTL: Keyboard types: other type

```
KDADDIO = $4B34
```

IOCTL: add i/o port as valid

```
KDDELIO = $4B35
```

IOCTL: delete i/o port as valid

```
KDDISABIO = $4B37
```

IOCTL: disable i/o to video board

```
KDENABIO = $4B36
```

IOCTL: enable i/o to video board

```
KDFONTOP = $4B72
```

IOCTL: font operations

```
KDGETKEYCODE = $4B4C
```

IOCTL: read kernel keycode table entry

```
KDGETLED = $4B31
```

IOCTL: return current led state

```
KDGETMODE = $4B3B
```

IOCTL: get current mode

```
KDGKBDIACR = $4B4A
```

IOCTL: read kernel accent table

```
KDGKBTYPE = $4B33
```

IOCTL: get keyboard type

```
KDMAPDISP = $4B3C
```

IOCTL: map display into address space

```
KDMKTONE = $4B30
```

IOCTL: generate tone

```
KDSETKEYCODE = $4B4D
```

IOCTL: write kernel keycode table entry

```
KDSETLED = $4B32
```

IOCTL: set led state

```
KDSETMODE = $4B3A
```

IOCTL: set text/graphics mode

```
KDSIGACCEPT = $4B4E
```

IOCTL: accept kbd generated signals

```
KDSKBDIACR = $4B4B
```

IOCTL: write kernel accent table

```
KDUNMAPDISP = $4B3D
```

IOCTL: unmap display from address space

```
KD_GRAPHICS = 1
```

IOCTL: Tty modes: graphics mode

```
KD_TEXT = 0
```

IOCTL: Tty modes: Text mode

```
KD_TEXT0 = 2
```

IOCTL: Tty modes: Text mode (obsolete)

```
KD_TEXT1 = 3
```

IOCTL: Tty modes: Text mode (obsolete)

```
KIOCSOUND = $4B2F
```

IOCTL: start/stop sound generation (0 for off)

```
LED_CAP = 4
```

IOCTL: LED_CAP : caps lock led

```
LED_NUM = 2
```

IOCTL: LED_SCR : Num lock led

```
LED_SCR = 1
```

IOCTL: LED_SCR : scroll lock led

```
LINUX_CAPABILITY_VERSION = $19980330
```

Current capability version in use by kernel

```
MAP_DENYWRITE = $800
```

Read-only

```
MAP_EXECUTABLE = $1000
```

Memory area is marked as executable

```
MAP_GROWSDOWN = $100
```

Memory map grows down, like stack

```
MAP_LOCKED = $2000
```

Memory pages are locked

```
MAP_NORESERVE = $4000
```

Do not check for reservations

```
MAX_CLOCKS = 16
```

Maximum number of clocks in the system

```
PIO_CMAP = $4B71
```

IOCTL: Set colour palette on VGA+

```
PIO_FONT = $4B61
```

IOCTL: Use font in expanded form.

```
PIO_FONTRESET = $4B6D
```

IOCTL: Reset to default font

```
PIO_FONTX = $4B6C
```

IOCTL: Set font in `consolefontdesc` record.

```
PIO_SCRNMAP = $4B41
```

IOCTL: put screen mapping table in kernel

```
PIO_UNIMAP = $4B67
```

IOCTL: put unicode-to-font mapping in kernel

```
PIO_UNIMAPCLR = $4B68
```

IOCTL: clear table, possibly advise hash algorithm

```
PIO_UNISCRNMAP = $4B6A
```

IOCTL: set full Unicode screen mapping

```
POLLMSG = $0400
```

Unused in linux

```
POLLRDHUP = $2000
```

Peer Shutdown/closed writing half of connection

```
POLLREMOVE = $1000
```

Undocumented linux extension of Poll

```
SPLICE_F_GIFT = 8
```

Pages spliced in are a gift

```
SPLICE_F_MORE = 4
```

Expect more data

```
SPLICE_F_MOVE = 1
```

Move pages instead of copying

```
SPLICE_F_NONBLOCK = 2
```

Don't block on pipe splicing operations

```
SYNC_FILE_RANGE_WAIT_AFTER = 4
```

Wait upon write-out of specified pages in the range after performing any write.

```
SYNC_FILE_RANGE_WAIT_BEFORE = 1
```

Wait for write-out of previously-submitted specified pages before writing more data.

```
SYNC_FILE_RANGE_WRITE = 2
```

Initiate write of all dirty pages in the specified range.

61.3.2 Types

```
clockid_t = cint
```

Clock id type

```
EPoll_Data = record
case Integer of
0: (
  ptr : pointer;
);
1: (
  fd : cint;
);
2: (
  u32 : cuint;
);
3: (
  u64 : cuint64;
);
end
```

Data structure used in EPOLL IOCTL call.

```
EPoll_Event = record
  Events : cuint32;
  Data : TEPoll_Data;
end
```

Structure used in epoll_ctl (363) call.

```
inotify_event = record
  wd : cint;
  mask : cuint32;
  cookie : cuint32;
  len : cuint32;
  name : Char;
end
```

`inotify_event` is the structure used to report changes in a directory. When reading a inotify file descriptor, one or more `inotify_event` records can be read from the file descriptor.

`PEPoll_Data = ^EPoll_Data`

Pointer to EPoll_Data (357) record

`PEpoll_Event = ^EPoll_Event`

Pointer to EPoll_Event (357) type

`Pinotify_event = ^inotify_event`

Pointer to inotify_event (358) structure.

`PSysInfo = ^TSysInfo`

Pointer to TSysInfo (359) record.

`Puser_cap_data = ^user_cap_data`

Pointer to user_cap_data (359) record

`Puser_cap_header = ^user_cap_header`

Pointer to user_cap_header (359) record

`TCloneFunc = function(args: pointer) : LongInt`

Clone function prototype.

`TEPoll_Data = EPoll_Data`

Alias for EPoll_Data (357) type

`TEPoll_Event = EPoll_Event`

Alias for EPoll_Event (357) type

```
TSysInfo = record
  uptime : clong;
  loads : Array[0..2] of culong;
  totalram : culong;
  freeram : culong;
  sharedram : culong;
  bufferram : culong;
  totalswap : culong;
  freeswap : culong;
  procs : cushort;
  pad : cushort;
  totalhigh : culong;
  freehigh : culong;
  mem_unit : cuint;
  _f : Array[0..19-2*sizeof(clong)-sizeof(cint)] of cChar;
end
```

Record with system information, used by the SysInfo (368) call.

```
user_cap_data = record
  effective : cuint32;
  permitted : cuint32;
  inheritable : cuint32;
end
```

`user_cap_data` describes the set of capabilities for the indicated thread.

```
user_cap_header = record
  version : cuint32;
  pid : cint;
end
```

`user_cap_header` describes the root user capabilities for the current thread, as set by capget (359) and capset (360)

61.4 Procedures and functions

61.4.1 capget

Synopsis: Return the capabilities for the indicated thread

Declaration: `function capget(header: Puser_cap_header;data: Puser_cap_data) : cint`

Visibility: default

Description: `capget` returns the capabilities of the indicated thread in `header`. The thread is identified by the process ID, or -1 for all caller (and child) process ID's.

Refer to the linux man pages (7 capabilities) for more info.

Errors: On success, zero is returned, on error -1 is returned, and fperrno is set to the error.

See also: capset (360)

61.4.2 capset

Synopsis: Set the capabilities for the indicated thread

Declaration: `function capset(header: Puser_cap_header;data: Puser_cap_data) : cint`

Visibility: default

Description: `capget` sets the capabilities of the indicated thread in `header`. The thread is identified by the process ID, or -1 for all caller (and child) process ID's.

Refer to the linux man pages (7 capabilities) for more info.

Errors: On success, zero is returned, on error -1 is returned, and fperrno is set to the error.

See also: capget (359)

61.4.3 clock_getres

Synopsis: Get clock resolution

Declaration: `function clock_getres(clk_id: clockid_t;res: ptimespec) : cint`

Visibility: default

Description: `clock_getres` returns the resolution of the clock specified in `clk_id` in the `res` structure. It can be `Nil`. if the clock exists and the resolution can be retrieved, `0` is returned.

Errors: On Error, -1 is returned. `fpgeterrno` can be used to get more detailed error information.

See also: clock_gettime (360), clock_settime (360)

61.4.4 clock_gettime

Synopsis: Get the time of a clock

Declaration: `function clock_gettime(clk_id: clockid_t;tp: ptimespec) : cint`

Visibility: default

Description: `clock_gettime` returns the current time of the clock specified in `clk_id` in the `tp` structure. If the clock exists and the time can be retrieved, `0` is returned.

Errors: On Error, -1 is returned. `fpgeterrno` can be used to get more detailed error information.

See also: clock_getres (360), clock_settime (360)

61.4.5 clock_settime

Synopsis: Set the time of a clock

Declaration: `function clock_settime(clk_id: clockid_t;tp: ptimespec) : cint`

Visibility: default

Description: `clock_settime` sets the current time of the clock specified in `clk_id`. The time is specified in the `tp` structure. If the clock exists and the time can be retrieved, `0` is returned. The resolution is truncated to the resolution supported by the specified clock. Note that not all clocks can be set.

Errors: On Error, -1 is returned. `fpgeterrno` can be used to get more detailed error information.

See also: clock_getres (360), clock_gettime (360)

61.4.6 clone

Synopsis: Clone current process (create new thread)

Declaration:
```
function clone(func: TCloneFunc;sp: pointer;flags: LongInt;
                  args: pointer) : LongInt
```

Visibility: default

Description: `Clone` creates a child process which is a copy of the parent process, just like FpFork (51) does. In difference with `Fork`, however, the child process shares some parts of it's execution context with its parent, so it is suitable for the implementation of threads: many instances of a program that share the same memory.

When the child process is created, it starts executing the function `Func`, and passes it `Args`. The return value of `Func` is either the explicit return value of the function, or the exit code of the child process.

The `sp` pointer points to the memory reserved as stack space for the child process. This address should be the top of the memory block to be used as stack.

The `Flags` determine the behaviour of the `Clone` call. The low byte of the Flags contains the number of the signal that will be sent to the parent when the child dies. This may be bitwise OR'ed with the following constants:

CLONE_VM Parent and child share the same memory space, including memory (un)mapped with subsequent `mmap` calls.

CLONE_FS Parent and child have the same view of the filesystem; the `chroot`, `chdir` and `umask` calls affect both processes.

CLONE_FILESthe file descriptor table of parent and child is shared.

CLONE_SIGHANDthe parent and child share the same table of signal handlers. The signal masks are different, though.

CLONE_PIDPArent and child have the same process ID.

Clone returns the process ID in the parent process, and -1 if an error occurred.

Errors: On error, -1 is returned to the parent, and no child is created.

sys_eagain Too many processes are running.

sys_enomem Not enough memory to create child process.

See also: #rtl.baseunix.FpFork (51)

Listing: ./linuxex/ex71.pp

```
program TestC{lone};

{$ifdef Linux}
// close is very Linux specific. 1.9.x threading is done via pthreads.

uses
  Linux, Errors, crt;

const
  Ready : Boolean = false;
  aChar : Char    = 'a';

function CloneProc( Arg: Pointer ): LongInt; Cdecl;
```

```
begin
  WriteLn('Hello from the clone ',PChar(Arg));
  repeat
    Write(aChar);
    Select(0,0,0,0,600);
  until Ready;
  WriteLn( 'Clone finished.');
  CloneProc := 1;
end;

var
  PID : LongInt;

procedure MainProc;
begin
  WriteLn('cloned process PID: ', PID );
  WriteLn('Press <ESC> to kill ... ' );
  repeat
    Write('.');
    Select(0,0,0,0,300);
    if KeyPressed then
      case ReadKey of
        #27: Ready := true;
        'a': aChar := 'A';
        'A': aChar := 'a';
        'b': aChar := 'b';
        'B': aChar := 'B';
      end;
  until Ready;
  WriteLn('Ready.');
end;

const
  StackSze = 16384;
  theFlags = CLONE_VM+CLONE_FS+CLONE_FILES+CLONE_SIGHAND;
  aMsg     : PChar = 'Oops !';

var
  theStack : Pointer;
  ExitStat : LongInt;

begin
  GetMem(theStack,StackSze);
  PID := Clone(@CloneProc,
               Pointer( LongInt(theStack)+StackSze),
               theFlags,
               aMsg);
  if PID < 0 then
    WriteLn('Error : ', LinuxError, ' when cloning.')
  else
    begin
    MainProc;
    case WaitPID(0,@ExitStat,Wait_Untraced or wait_clone) of
      -1: WriteLn('error:',LinuxError,'; ',StrError(LinuxError));
       0: WriteLn('error:',LinuxError,'; ',StrError(LinuxError));
    else
      WriteLn('Clone exited with: ',ExitStat shr 8);
    end;
```

```
    end;
  FreeMem( theStack, StackSze );
{$else}
begin
{$endif}
end.
```

61.4.7 epoll_create

Synopsis: Create new epoll file descriptor

Declaration: `function epoll_create(size: cint) : cint`

Visibility: default

Description: `epoll_create` creates a new epoll file descriptor. The `size` argument indicates to the kernel approximately how many structures should be allocated, but is by no means an upper limit.

On success, a file descriptor is returned that can be used in subsequent epoll_ctl (363) or epoll_wait (364) calls, and should be closed using the fpClose (51) call.

Errors: On error, -1 is returned, and errno (51) is set.

See also: epoll_ctl (363), epoll_wait (364), fpClose (51)

61.4.8 epoll_ctl

Synopsis: Modify an epoll file descriptor

Declaration: `function epoll_ctl(epfd: cint;op: cint;fd: cint;event: PEpoll_Event) : cint`

Visibility: default

Description: `epoll_ctl` performs the `op` operation on epoll file descriptor `epfd`. The operation will be monitored on file descriptor `fd`, and is optionally controlled by `event`.

`op` can be one of the following values:

EPOLL_CTL_ADD Add filedescriptor to list of events

EPOLL_CTL_MOD Modify event for filedescriptor

EPOLL_CTL_DEL Delete event for filedescriptor

The `events` field in `event_data` is a bitmask of one or more of the following values:

EPOLLIN The file is ready for read operations

EPOLLOUT The file is ready for write operations.

EPOLLPRI Urgent data is available for read operations.

EPOLLERR An error condition is signaled on the file descriptor.

EPOLLHUP A Hang up happened on the file descriptor.

EPOLLET Set the Edge Triggered behaviour for the file descriptor.

EPOLLONESHOT Set One-Shot behaviour for the file descriptor. The event will be triggered only once.

Errors: On error -1 is returned, and errno is set accordingly.

See also: epoll_create (363), epoll_wait (364), fpClose (51)

61.4.9 epoll_wait

Synopsis: Wait for an event on an epoll file descriptor.

Declaration:
```
function epoll_wait(epfd: cint;events: PEpoll_Event;maxevents: cint;
                    timeout: cint) : cint
```

Visibility: default

Description: `epoll_wait` waits for `timeout` milliseconds for an event to occur on epoll file descriptor `epfd`. If `timeout` is -1, it waits indefinitely, if `timeour` is zero, it does not wait, but returns immediately, even if no events were detected.

On return, data for at most `maxevents` will be returned in the memory pointed to by `events`. The function returns the number of file descriptors for which events were reported. This can be zero if the timeout was reached.

Errors: On error -1 is returned, and errno is set accordingly.

See also: epoll_create (363), epoll_ctl (363), fpClose (51)

61.4.10 fdatasync

Synopsis: Synchronize the data in memory with the data on storage device

Declaration:
```
function fdatasync(fd: cint) : cint
```

Visibility: default

Description: `fdatasync` does the same as `fpfsync` but does not flush the metadata, unless it is vital to the correct reading/writing of the file. In practice, this means that unless the file size changed, the file metadata will not be synced.

See also: #rtl.unix.fsync (734)

61.4.11 futex

Synopsis: Perform a futex operation

Declaration:
```
function futex(uaddr: Pcint;op: cint;val: cint;timeout: Ptimespec;
               addr2: Pcint;val3: cint) : cint
function futex(var uaddr;op: cint;val: cint;timeout: Ptimespec;
               var addr2;val3: cint) : cint
function futex(var uaddr;op: cint;val: cint;var timeout: Ttimespec;
               var addr2;val3: cint) : cint
function futex(uaddr: Pcint;op: cint;val: cint;timeout: Ptimespec)
                : cint
function futex(var uaddr;op: cint;val: cint;timeout: Ptimespec) : cint
function futex(var uaddr;op: cint;val: cint;var timeout: Ttimespec)
                : cint
```

Visibility: default

Description: `futex` performs an operation on a memory futex as described in the kernel manual page for futex. The mutex is located at `uaddr`, the operation `op` is one of the following constants:

FUTEX_WAIT Futex option: Wait on futex till wake call arrives.

FUTEX_WAKE Futex option: Wait on futex till wake call arrives.

FUTEX_FD Futex option: Associate file descriptor with futex.

FUTEX_REQUEUE Futex option: requeue waiting processes on other futex.

FUTEX_CMP_REQUEUE Futex option: requeue waiting processes on other futex, but check it's value first

The value to check for is indicated in `val`, and a timeout can be specified in `timeout`. The optional arguments `addr2` and `val3` are used only with the `FUTEX_REQUEUE` and `FUTEX_CMP_REQUEUE` operations.

In case of an error, -1 is return. All other return values must be interpreted according to the operation performed.

This call directly interfaces with the Linux kernel, more information can be found in the kernel manual pages.

Errors: On error, -1 is returned. Use #rtl.baseunix.fpgeterrno (51) to get the error code.

61.4.12 futex_op

Synopsis: Futex operation:

Declaration: `function futex_op(op: cint;oparg: cint;cmp: cint;cmparg: cint) : cint`

Visibility: default

Description: `FUTEX_OP` Performs an operation on a futex:

```
FUTEX_OP := ((op and $F) shl 28) or
            ((cmp and $F) shl 24) or
            ((oparg and $FFF) shl 12)
            or (cmparg and $FFF);
```

61.4.13 inotify_add_watch

Synopsis: Add a watch to a notify file descriptor

Declaration: `function inotify_add_watch(fd: cint;name: Pchar;mask: cuint32) : cint`

Visibility: default

Description: `inotify_add_watch` can be used to add a watch to an initialized inotify file descriptor (`fd`). The file or directory to watch can be specified in the `name` parameter, and the events that must be reported can be specified in `mask`. The following flags can be specified:

IN_ACCESS Data was read from file.

IN_MODIFY Data was written to file.

IN_ATTRIB File attributes changed.

IN_CLOSE_WRITE File opened for write was closed

IN_CLOSE_NOWRITE File opened for read was closed

IN_CLOSE File was closed (read or write)

IN_OPEN File was opened

IN_MOVED_FROM File was moved away from watched directory

IN_MOVED_TO File was moved into watched directory

IN_MOVE File was moved (in or out of directory)

IN_CREATEA file was created in the directory.

IN_DELETEA file was deleted from the directory.

IN_DELETE_SELF Directory or file under observation was deleted.

IN_MOVE_SELF Directory or file under observation was moved.

IN_ALL_EVENTS All possible events OR-ed together.

These events can be OR-ed with some flags, controlling the behaviour of the watch:

IN_ONLYDIR Only watch filename if it is a directory.

IN_ISDIR Event occurred against directory.

IN_DONT_FOLLOW Do not follow symlinks

IN_MASK_ADD Add events to existing watch (OR-ing the sets) if one exists.

IN_ONESHOT Only report one event, then remove the watch.

On return, the function returns a watch descriptor, which will be reported in the inotify_event (358) structure's `wd`.

Errors: On Error, -1 is returned. `fpgeterrno` can be used to get more detailed error information.

See also: inotify_init (366), inotify_init1 (366), inotify_rm_watch (367), inotify_event (358)

61.4.14 inotify_init

Synopsis: Initialize a new inotify file descriptor

Declaration: `function inotify_init : cint`

Visibility: default

Description: `inotify_init` initializes a new `INotify` file descriptor. No options can be specified. On return, the file descriptor is returned.

Errors: On Error, -1 is returned. `fpgeterrno` can be used to get more detailed error information

See also: inotify_init1 (366), inotify_add_watch (365), inotify_rm_watch (367)

61.4.15 inotify_init1

Synopsis: Initialize a new inotify file descriptor with extra options.

Declaration: `function inotify_init1(flags: cint) : cint`

Visibility: default

Description: `inotify_init1` initializes a new `INotify` file descriptor. The following options can be OR-ed and passed in flags:

IN_NONBLOCK Do not block on read

IN_CLOEXEC Inotify close on exec flag.

Errors: On Error, -1 is returned. `fpgeterrno` can be used to get more detailed error information.

See also: inotify_init (366), inotify_add_watch (365), inotify_rm_watch (367)

61.4.16 inotify_rm_watch

Synopsis: Remove watch from Inotify file descriptor.

Declaration: `function inotify_rm_watch(fd: cint;wd: cint) : cint`

Visibility: default

Description: `inotify_rm_watch` removes watch descriptor `wd` from inotify descriptor `fd`. On success, 0 is returned.

Errors: On Error, -1 is returned. `fpgeterrno` can be used to get more detailed error information.

See also: inotify_init (366), inotify_init1 (366), inotify_add_watch (365), inotify_event (358)

61.4.17 sched_yield

Synopsis: Yield the processor to another thread.

Declaration: `procedure sched_yield`

Visibility: default

Description: `sched_yield` yields the processor to another thread. The current thread is put at the back of its queue. If there is only 1 thread in the application, the thread continues to run. The call always returns zero.

61.4.18 setregid

Declaration: `function setregid(rgid: uid_t;egid: uid_t) : cint`

Visibility: default

61.4.19 setreuid

Declaration: `function setreuid(ruid: uid_t;euid: uid_t) : cint`

Visibility: default

61.4.20 sync_file_range

Synopsis: Force committing of data to disk

Declaration:
```
function sync_file_range(fd: cInt;offset: off64_t;nbytes: off64_t;
                         flags: cuInt) : cInt
```

Visibility: default

Description: `sync_file_range` forces the linux kernel to write any data pages of a specified file (file descriptor `fd`) to disk. The range of the file is specified by the offset `offset` and the number of bytes `nbytes`. `Options` is an OR-ed combination of

SYNC_FILE_RANGE_WAIT_BEFORE Wait for write-out of previously-submitted specified pages before writing more data.

SYNC_FILE_RANGE_WRITE Initiate write of all dirty pages in the specified range.

SYNC_FILE_RANGE_WAIT_AFTER Wait upon write-out of specified pages in the range after performing any write.

If none is specified, the operation does nothing.

Errors: On return -1 is returned and fperrno is set to the actual error code. See the linux man page for more on the error codes.

See also: fdatasync (364)

61.4.21 Sysinfo

Synopsis: Return kernel system information

Declaration: `function Sysinfo(Info: PSysInfo) : cInt`

Visibility: default

Description: `SysInfo` returns system information in `Info`. Returned information in `Info` includes:

uptimeNumber of seconds since boot.

loads1, 5 and 15 minute load averages.

totalramtotal amount of main memory.

freeramamount of free memory.

sharedramamount of shared memory.

bufferramamount of memory used by buffers.

totalswaptotal amount of swapspace.

freeswapamount of free swapspace.

procsnumber of current processes.

Errors: None.

See also: #rtl.baseunix.fpUname (51)

Listing: ./linuxex/ex64.pp

```
program Example64;

{ Example to demonstrate the SysInfo function.
  Sysinfo is Linux-only. }

{$ifdef Linux}
Uses Linux;

Function Mb(L : Longint) : longint;

begin
  Mb:=L div (1024*1024);
end;

Var Info : TSysInfo;
    D,M,Secs,H : longint;
{$endif}

begin
```

```
  {$ifdef Linux}
  If Not SysInfo(Info) then
    Halt(1);
  With Info do
    begin
    D:=Uptime div (3600*24);
    UpTime:=UpTime mod (3600*24);
    h:=uptime div 3600;
    uptime:=uptime mod 3600;
    m:=uptime div 60;
    secs:=uptime mod 60;
    Writeln('Uptime : ',d,'days, ',h,' hours, ',m,' min, ',secs,' s.');
    Writeln('Loads  : ',Loads[1],'/',Loads[2],'/',Loads[3]);
    Writeln('Total Ram  : ',Mb(totalram),'Mb.');
    Writeln('Free Ram   : ',Mb(freeram),'Mb.');
    Writeln('Shared Ram : ',Mb(sharedram),'Mb.');
    Writeln('Buffer Ram : ',Mb(bufferram),'Mb.');
    Writeln('Total Swap : ',Mb(totalswap),'Mb.');
    Writeln('Free Swap  : ',Mb(freeswap),'Mb.');
    end;
  {$endif}
end.
```

Chapter 62

Reference for unit 'lnfodwrf'

62.1 Overview

The lnfodwrf provides a routine that reads the debug information of an executable (if any exists) and returns source code information about this address. It works with `DWARF` debug information. Note that this unit is not thread-safe, and that its behaviour is undefined if multiple threads try to write a backtrace at the same time.

For stabs debug information, the lineinfo (341) unit must be used.

62.2 Procedures and functions

62.2.1 GetLineInfo

Synopsis: Return source line information about an address.

Declaration:
```
function GetLineInfo(addr: ptruint;var func: string;var source: string;
                     var line: LongInt) : Boolean
```

Visibility: default

Description: `GetLineInfo` returns source line information about the address `addr`. It searches this information in the DWARF debugging information found in the binary: If the file was compiled without debug information, nothing will be returned. Upon succesful retrieval of the debug information, `True` is returned, and the `func` parameter is filled with the name of the function in which the address is located. The `source` parameter contains the name of the file in which the function was implemented, and `line` contains the line number in the source file for `addr`.

Errors: If no debug information is found, `False` is returned.

Chapter 63

Reference for unit 'math'

63.1 Used units

Table 63.1: Used units by unit 'math'

Name	Page
System	622
sysutils	628

63.2 Overview

This document describes the math unit. The `math` unit was initially written by Florian Klaempfl. It provides mathematical functions which aren't covered by the system unit.

This chapter starts out with a definition of all types and constants that are defined, after which an overview is presented of the available functions, grouped by category, and the last part contains a complete explanation of each function.

The following things must be taken into account when using this unit:

1. This unit is compiled in Object Pascal mode so all `integers` are 32 bit.
2. Some overloaded functions exist for data arrays of integers and floats. When using the address operator (`@`) to pass an array of data to such a function, make sure the address is typecasted to the right type, or turn on the 'typed address operator' feature. failing to do so, will cause the compiler not be able to decide which function you want to call.

63.3 Cash flow functions

The cash flow functions in the math unit resolve the following equation:

```
FV + PV * q^n + PMT (q^n - 1) / (q - 1) = 0
```

In this formula, the following variables are present:

FV Future value

PV Present value

PMT Payment per period

n Number of payments (number of periods)

q> Interest Rate (return rate)

The financial functions FutureValue (388), NumberOfPeriods (404), Payment (404), PresentValue (407) and InterestRate (391) solve this equation for one of the variables, when the other variables are known.

See also: FutureValue (388), NumberOfPeriods (404), Payment (404), PresentValue (407), TPaymentTime (377)

63.4 Geometrical functions

Table 63.2:

Name	Description
hypot (390)	Hypotenuse of triangle
norm (403)	Euclidian norm

63.5 Statistical functions

Table 63.3:

Name	Description
mean (398)	Mean of values
meanandstddev (399)	Mean and standard deviation of values
momentskewkurtosis (402)	Moments, skew and kurtosis
popnstddev (404)	Population standarddeviation
popnvariance (405)	Population variance
randg (409)	Gaussian distributed randum value
stddev (413)	Standard deviation
sum (414)	Sum of values
sumofsquares (415)	Sum of squared values
sumsandsquares (416)	Sum of values and squared values
totalvariance (418)	Total variance of values
variance (419)	variance of values

63.6 Number converting

Table 63.4:

Name	Description
ceil (381)	Round to infinity
floor (387)	Round to minus infinity
frexp (387)	Return mantissa and exponent

63.7 Exponential and logarithmic functions

Table 63.5:

Name	Description
intpower (391)	Raise float to integer power
ldexp (393)	Calculate $2p̂ x$
lnxp1 (393)	calculate `log(x+1)`
log10 (394)	calculate 10-base log
log2 (394)	calculate 2-base log
logn (395)	calculate N-base log
power (406)	raise float to arbitrary power

63.8 Hyperbolic functions

Table 63.6:

Name	Description
arcosh (378)	caculate reverse hyperbolic cosine
arsinh (380)	caculate reverse hyperbolic sine
artanh (381)	caculate reverse hyperbolic tangent
cosh (383)	calculate hyperbolic cosine
sinh (413)	calculate hyperbolic sine
tanh (418)	calculate hyperbolic tangent

63.9 Trigoniometric functions

Table 63.7:

Name	Description
arccos (377)	calculate reverse cosine
arcsin (379)	calculate reverse sine
arctan2 (380)	calculate reverse tangent
cotan (384)	calculate cotangent
sincos (412)	calculate sine and cosine
tan (417)	calculate tangent

63.10 Angle unit conversion

Routines to convert angles between different angle units.

Table 63.8:

Name	Description
cycletorad (384)	convert cycles to radians
degtograd (385)	convert degrees to grads
degtorad (386)	convert degrees to radians
gradtodeg (389)	convert grads to degrees
gradtorad (389)	convert grads to radians
radtocycle (407)	convert radians to cycles
radtodeg (408)	convert radians to degrees
radtograd (408)	convert radians to grads

63.11 Min/max determination

Functions to determine the minimum or maximum of numbers:

Table 63.9:

Name	Description
max (396)	Maximum of 2 values
maxIntValue (396)	Maximum of an array of integer values
maxvalue (397)	Maximum of an array of values
min (400)	Minimum of 2 values
minIntValue (400)	Minimum of an array of integer values
minvalue (401)	Minimum of an array of values

63.12 Constants, types and variables

63.12.1 Constants

```
EqualsValue = 0
```

Values are the same

```
GreaterThanValue = (TValueRelationship)
```

First values is greater than second value

```
Infinity = 1.0 / 0.0
```

Value is infinity

```
LessThanValue = (TValueRelationship)
```

First value is less than second value

```
MaxComp = 9.223372036854775807e + 18
```

```
MaxDouble = 1.7e + 308
```

```
MaxExtended = 1.1e + 4932
```

Maximum value of extended type

```
MaxFloat = 0
```

Maximum value of float type

```
MaxSingle = 3.4e + 38
```

```
MinComp = -9.223372036854775807e + 18
```

```
MinDouble = 5.0e - 324
```

```
MinExtended = 3.4e - 4932
```

Minimum value (closest to zero) of extended type

```
MinFloat = 0
```

Minimum value (closest to zero) of float type

```
MinSingle = 1.5e - 45
```

```
NaN = 0.0 / 0.0
```

Value is Not a Number

```
NegativeValue = (TValueSign)
```

Value is negative

```
NegInfinity = (-1.0) / 0.0
```

Value is negative (minus) infinity

```
PositiveValue = (TValueSign)
```

Value is positive

```
ZeroValue = 0
```

Value is zero

63.12.2 Types

```
Float = MaxFloatType
```

All calculations are done with the Float type which is the largest float type available for the current CPU. This allows to recompile the unit with a different float type to obtain a desired precision. The pointer type PFloat (376) is used in functions that accept an array of values of arbitrary length.

```
Float = MaxFloatType
```

All calculations are done with the Float type which is the largest float type available for the current CPU. This allows to recompile the unit with a different float type to obtain a desired precision. The pointer type PFloat (376) is used in functions that accept an array of values of arbitrary length.

```
PFloat = ^Float
```

Pointer to Float (376) type.

```
PInteger = ObjPas.PInteger
```

Pointer to integer type

```
TFPUException = system.TFPUException
```

Type describing Floating Point processor exceptions.

```
TFPUExceptionMask = system.TFPUExceptionMask
```

Type to set the Floating Point Unit exception mask.

```
TFPUPrecisionMode = system.TFPUPrecisionMode
```

Type describing the default precision for the Floating Point processor.

```
TFPURoundingMode = system.TFPURoundingMode
```

Type describing the rounding mode for the Floating Point processor.

```
tpaymenttime = (ptendofperiod,ptstartofperiod)
```

Table 63.10: Enumeration values for type tpaymenttime

Value	Explanation
ptendofperiod	End of period.
ptstartofperiod	Start of period.

Type used in financial (interest) calculations.

```
TRoundToRange = -37..37
```

`TRoundToRange` is the range of valid digits to be used in the RoundTo (410) function.

```
TValueRelationship = -1..1
```

Type to describe relational order between values

```
TValueSign = -1..1
```

Type indicating sign of a valuea

63.13 Procedures and functions

63.13.1 arccos

Synopsis: Return inverse cosine

Declaration: `function arccos(x: Float) : Float`

Visibility: default

Description: `Arccos` returns the inverse cosine of its argument `x`. The argument `x` should lie between -1 and 1 (borders included).

Errors: If the argument `x` is not in the allowed range, an `EInvalidArgument` exception is raised.

See also: arcsin (379), arccosh (378), arsinh (380), artanh (381)

Listing: ./mathex/ex1.pp

```
Program Example1;

{ Program to demonstrate the arccos function. }

Uses math;

  Procedure WriteRadDeg(X : float);

  begin
    WriteIn(X:8:5,' rad = ',radtodeg(x):8:5,' degrees.')
  end;

begin
  WriteRadDeg (arccos(1));
  WriteRadDeg (arccos(sqrt(3)/2));
  WriteRadDeg (arccos(sqrt(2)/2));
  WriteRadDeg (arccos(1/2));
  WriteRadDeg (arccos(0));
  WriteRadDeg (arccos(-1));
end.
```

63.13.2 arccosh

Synopsis: Return inverse hyperbolic cosine

Declaration: `function arccosh(x: Float) : Float`

Visibility: default

Description: `arccosh` returns the inverse hyperbolic cosine of it's argument `x`.

This function is an alias for arcosh (378), provided for Delphi compatibility.

See also: arcosh (378)

63.13.3 arcosh

Synopsis: Return inverse hyperbolic cosine

Declaration: `function arcosh(x: Float) : Float`

Visibility: default

Description: `Arcosh` returns the inverse hyperbolic cosine of its argument `x`. The argument `x` should be larger than 1. The `arccosh` variant of this function is supplied for Delphi compatibility.

Errors: If the argument `x` is not in the allowed range, an `EInvalidArgument` exception is raised.

See also: cosh (383), sinh (413), arcsin (379), arsinh (380), artanh (381), tanh (418)

Listing: ./mathex/ex3.pp

```
Program Example3;

{ Program to demonstrate the arcosh function. }

Uses math;
```

```
begin
  WriteLn(arcosh(1));
  WriteLn(arcosh(2));
end.
```

63.13.4 arcsin

Synopsis: Return inverse sine

Declaration: `function arcsin(x: Float) : Float`

Visibility: default

Description: `Arcsin` returns the inverse sine of its argument `x`. The argument `x` should lie between -1 and 1.

Errors: If the argument `x` is not in the allowed range, an `EInvalidArgument` exception is raised.

See also: arccos (377), arcosh (378), arsinh (380), artanh (381)

Listing: ./mathex/ex2.pp

```
Program Example1;

{ Program to demonstrate the arcsin function. }

Uses math;

  Procedure WriteRadDeg(X : float);

  begin
    WriteLn(X:8:5,' rad = ',radtodeg(x):8:5,' degrees.')
  end;

begin
  WriteRadDeg (arcsin(1));
  WriteRadDeg (arcsin(sqrt(3)/2));
  WriteRadDeg (arcsin(sqrt(2)/2));
  WriteRadDeg (arcsin(1/2));
  WriteRadDeg (arcsin(0));
  WriteRadDeg (arcsin(-1));
end.
```

63.13.5 arcsinh

Synopsis: Return inverse hyperbolic sine

Declaration: `function arcsinh(x: Float) : Float`

Visibility: default

Description: `arcsinh` returns the inverse hyperbolic sine of it's argument `x`.

This function is an alias for arsinh (380), provided for Delphi compatibility.

See also: arsinh (380)

63.13.6 arctan2

Synopsis: Return arctangent of (y/x)

Declaration: `function arctan2(y: Float;x: Float) : Float`

Visibility: default

Description: `arctan2` calculates `arctan(y/x)`, and returns an angle in the correct quadrant. The returned angle will be in the range $-\pi$ to π radians. The values of `x` and `y` must be between -2\^{}64 and 2\^{}64, moreover `x` should be different from zero. On Intel systems this function is implemented with the native intel `fpatan` instruction.

See also: arccos (377), arcosh (378), arsinh (380), artanh (381)

Listing: ./mathex/ex6.pp

```
Program Example6;

{ Program to demonstrate the arctan2 function. }

Uses math;

  Procedure WriteRadDeg(X : float);

  begin
    WriteIn(X:8:5,' rad = ',radtodeg(x):8:5,' degrees.')
  end;

begin
  WriteRadDeg (arctan2(2,1));
end.
```

63.13.7 arctanh

Synopsis: Return inverse hyperbolic tangent

Declaration: `function arctanh(x: Float) : Float`

Visibility: default

Description: `arcsinh` returns the inverse hyperbolic tangent of it's argument `x`.

This function is an alias for artanh (381), provided for Delphi compatibility.

See also: artanh (381)

63.13.8 arsinh

Synopsis: Return inverse hyperbolic sine

Declaration: `function arsinh(x: Float) : Float`

Visibility: default

Description: `arsinh` returns the inverse hyperbolic sine of its argument `x`. The `arscsinh` variant of this function is supplied for Delphi compatibility.

Errors: None.

See also: arcosh (378), arccos (377), arcsin (379), artanh (381)

Listing: ./mathex/ex4.pp

```
Program Example4;

{ Program to demonstrate the arsinh function. }

Uses math;

begin
  Writeln(arsinh(0));
  Writeln(arsinh(1));
end.
```

63.13.9 artanh

Synopsis: Return inverse hyperbolic tangent

Declaration: `function artanh(x: Float) : Float`

Visibility: default

Description: `artanh` returns the inverse hyperbolic tangent of its argument `x`, where `x` should lie in the interval [-1,1], borders included. The `arctanh` variant of this function is supplied for Delphi compatibility.

Errors: In case `x` is not in the interval [-1,1], an `EInvalidArgument` exception is raised.

See also: arcosh (378), arccos (377), arcsin (379), artanh (381)

Listing: ./mathex/ex5.pp

```
Program Example5;

{ Program to demonstrate the artanh function. }

Uses math;

begin
  Writeln(artanh(0));
  Writeln(artanh(0.5));
end.
```

63.13.10 ceil

Synopsis: Return the lowest integer number greater than or equal to argument

Declaration: `function ceil(x: Float) : Integer`

Visibility: default

Description: `Ceil` returns the lowest integer number greater than or equal to `x`. The absolute value of `x` should be less than `maxint`.

Errors: If the asolute value of `x` is larger than maxint, an overflow error will occur.

See also: floor (387)

Listing: ./mathex/ex7.pp

```
Program Example7;

{ Program to demonstrate the Ceil function. }

Uses math;

begin
  WriteLn(Ceil(-3.7)); // should be -3
  WriteLn(Ceil(3.7));  // should be 4
  WriteLn(Ceil(-4.0)); // should be -4
end.
```

63.13.11 ClearExceptions

Synopsis: Clear Floating Point Unit exceptions

Declaration:
```
procedure ClearExceptions(RaisePending: Boolean)
```

Visibility: default

Description: Clear Floating Point Unit exceptions

63.13.12 CompareValue

Synopsis: Compare 2 values

Declaration:
```
function CompareValue(const A: Integer;const B: Integer)
                        : TValueRelationship
function CompareValue(const A: Int64;const B: Int64)
                        : TValueRelationship
function CompareValue(const A: QWord;const B: QWord)
                        : TValueRelationship
function CompareValue(const A: Single;const B: Single;delta: Single)
                        : TValueRelationship
function CompareValue(const A: Double;const B: Double;delta: Double)
                        : TValueRelationship
function CompareValue(const A: Extended;const B: Extended;
                      delta: Extended) : TValueRelationship
```

Visibility: default

Description: `CompareValue` compares 2 integer or floating point values `A` and `B` and returns one of the following values:

-1if `A<B`

0if `A=B`

1if `A>B`

See also: TValueRelationship (377)

63.13.13 cosecant

Synopsis: Calculate cosecant

Declaration: `function cosecant(x: Float) : Float`

Visibility: default

Description: `cosecant` calculates the cosecant (1/`sin(x)`) of its argument `x`.

Errors: If 0 or 180 degrees is specified, an exception will be raised.

See also: secant (411)

63.13.14 cosh

Synopsis: Return hyperbolic cosine

Declaration: `function cosh(x: Float) : Float`

Visibility: default

Description: `Cosh` returns the hyperbolic cosine of it's argument {x}.

Errors: None.

See also: arcosh (378), sinh (413), arsinh (380)

Listing: ./mathex/ex8.pp

```
Program Example8;

{ Program to demonstrate the cosh function. }

Uses math;

begin
  Writeln(Cosh(0));
  Writeln(Cosh(1));
end.
```

63.13.15 cot

Synopsis: Alias for `Cotan`

Declaration: `function cot(x: Float) : Float`

Visibility: default

Description: `cot` is an alias for the cotan (384) function.

See also: cotan (384)

63.13.16 cotan

Synopsis: Return cotangent

Declaration: `function cotan(x: Float) : Float`

Visibility: default

Description: `Cotan` returns the cotangent of it's argument `x`. The argument `x` must be in radians. `x` should be different from zero.

Errors: If `x` is zero then a overflow error will occur.

See also: tanh (418)

Listing: ./mathex/ex9.pp

```
Program Example9;

{ Program to demonstrate the cotan function. }

Uses math;

begin
  writeln(cotan(pi/2));
  Writeln(cotan(pi/3));
  Writeln(cotan(pi/4));
end.
```

63.13.17 csc

Synopsis: Alias for `cosecant`

Declaration: `function csc(x: Float) : Float`

Visibility: default

Description: `csc` is an alias for the cosecant (383) function.

See also: cosecant (383)

63.13.18 cycletorad

Synopsis: Convert cycle angle to radians angle

Declaration: `function cycletorad(cycle: Float) : Float`

Visibility: default

Description: `Cycletorad` transforms it's argument `cycle` (an angle expressed in cycles) to radians. (1 cycle is 2π radians).

Errors: None.

See also: degtograd (385), degtorad (386), radtodeg (408), radtograd (408), radtocycle (407)

Listing: ./mathex/ex10.pp

```
Program Example10;

{ Program to demonstrate the cycletorad function. }

Uses math;

begin
  writeln(cos(cycletorad(1/6))); // Should print 1/2
  writeln(cos(cycletorad(1/8))); // should be sqrt(2)/2
end.
```

63.13.19 DegNormalize

Synopsis: Normalize an angle measured in degrees

Declaration:
```
function DegNormalize(deg: single) : single
function DegNormalize(deg: Double) : Double
function DegNormalize(deg: extended) : extended
```

Visibility: default

Description: `DegNormalize` returns the angle `deg` as a positive angle between `0` and `360` degrees.

63.13.20 degtograd

Synopsis: Convert degree angle to grads angle

Declaration:
```
function degtograd(deg: Float) : Float
```

Visibility: default

Description: `Degtograd` transforms it's argument `deg` (an angle in degrees) to grads. (90 degrees is 100 grad.)

Errors: None.

See also: cycletorad (384), degtorad (386), radtodeg (408), radtograd (408), radtocycle (407)

Listing: ./mathex/ex11.pp

```
Program Example11;

{ Program to demonstrate the degtograd function. }

Uses math;

begin
  writeln(degtograd(90));
  writeln(degtograd(180));
  writeln(degtograd(270))
end.
```

63.13.21 degtorad

Synopsis: Convert degree angle to radians angle.

Declaration:
```
function degtorad(deg: Float) : Float
```

Visibility: default

Description: `Degtorad` converts it's argument `deg` (an angle in degrees) to radians. (pi radians is 180 degrees)

Errors: None.

See also: cycletorad (384), degtograd (385), radtodeg (408), radtograd (408), radtocycle (407)

Listing: ./mathex/ex12.pp

```
Program Example12;

{ Program to demonstrate the degtorad function. }

Uses math;

begin
  writeln(degtorad(45));
  writeln(degtorad(90));
  writeln(degtorad(180));
  writeln(degtorad(270));
  writeln(degtorad(360));
end.
```

63.13.22 DivMod

Synopsis: Return DIV and MOD of arguments

Declaration:
```
procedure DivMod(Dividend: LongInt;Divisor: Word;var Result: Word;
                 var Remainder: Word)
procedure DivMod(Dividend: LongInt;Divisor: Word;var Result: SmallInt;
                 var Remainder: SmallInt)
procedure DivMod(Dividend: DWord;Divisor: DWord;var Result: DWord;
                 var Remainder: DWord)
procedure DivMod(Dividend: LongInt;Divisor: LongInt;var Result: LongInt;
                 var Remainder: LongInt)
```

Visibility: default

Description: `DivMod` returns `Dividend` DIV `Divisor` in `Result`, and `Dividend` MOD `Divisor` in `Remainder`

63.13.23 EnsureRange

Synopsis: Change value to it falls in specified range.

Declaration:
```
function EnsureRange(const AValue: Integer;const AMin: Integer;
                     const AMax: Integer) : Integer;  Overload
function EnsureRange(const AValue: Int64;const AMin: Int64;
                     const AMax: Int64) : Int64;  Overload
function EnsureRange(const AValue: Double;const AMin: Double;
                     const AMax: Double) : Double;  Overload
```

Visibility: default

Description: `EnsureRange` returns `Value` if `AValue` is in the range `AMin..AMax`. It returns `AMin` if the value is less than `AMin`, or `AMax` if the value is larger than `AMax`.

See also: InRange (391)

63.13.24 floor

Synopsis: Return the largest integer smaller than or equal to argument

Declaration: `function floor(x: Float) : Integer`

Visibility: default

Description: `Floor` returns the largest integer smaller than or equal to `x`. The absolute value of `x` should be less than `maxint`.

Errors: If `x` is larger than `maxint`, an overflow will occur.

See also: ceil (381)

Listing: ./mathex/ex13.pp

```
Program Example13;

{ Program to demonstrate the floor function. }

Uses math;

begin
  Writeln(Ceil(-3.7)); // should be -4
  Writeln(Ceil(3.7));  // should be 3
  Writeln(Ceil(-4.0)); // should be -4
end.
```

63.13.25 Frexp

Synopsis: Return mantissa and exponent.

Declaration: `procedure Frexp(X: Float;var Mantissa: Float;var Exponent: Integer)`

Visibility: default

Description: `Frexp` returns the mantissa and exponent of it's argument `x` in `mantissa` and `exponent`.

Errors: None

Listing: ./mathex/ex14.pp

```
Program Example14;

{ Program to demonstrate the frexp function. }

Uses math;

Procedure dofrexp(Const X : extended);
```

```
var man : extended;
    exp: longint;

begin
  man:=0;
  exp:=0;
  frexp(x,man,exp);
  write(x,' has ');
  Writeln('mantissa ',man,' and exponent ',exp);
end;


begin
//   dofrexp(1.00);
   dofrexp(1.02e-1);
   dofrexp(1.03e-2);
   dofrexp(1.02e1);
   dofrexp(1.03e2);
end.
```

63.13.26 FutureValue

Synopsis: Calculate the future value of an investment.

Declaration:
```
function FutureValue(ARate: Float;NPeriods: Integer;APayment: Float;
                     APresentValue: Float;APaymentTime: tpaymenttime)
                      : Float
```

Visibility: default

Description: `FutureValue` calculates the future value of an investment (`FV`) in the cash flow formula (see CashFlowFunctions (371)) The function result is the future value of an investment of `APresentValue` (`PV`), where `APayment` (`PMT`) is invested for `NPeriods` (`n`) at the rate of `ARate` (`q`) per period.

The `APaymentTime` parameter determines whether the investment (payment) is an ordinary annuity or an annuity due: `ptEndOfPeriod` must be used if payments are at the end of each period. If the payments are at the beginning of the periode, `ptStartOfPeriod` must be used.

See also: InterestRate (391), NumberOfPeriods (404), Payment (404), PresentValue (407), TPaymentTime (377), CashFlowFunctions (371)

63.13.27 GetExceptionMask

Synopsis: Get the Floating Point Unit exception mask.

Declaration:
```
function GetExceptionMask : TFPUExceptionMask
```

Visibility: default

Description: Get the Floating Point Unit exception mask.

63.13.28 GetPrecisionMode

Synopsis: Return the Floating Point Unit precision mode.

Declaration:
```
function GetPrecisionMode : TFPUPrecisionMode
```

Visibility: default

Description: Return the Floating Point Unit precision mode.

63.13.29 GetRoundMode

Synopsis: Return the Floating Point Unit rounding mode.

Declaration: `function GetRoundMode : TFPURoundingMode`

Visibility: default

Description: Return the Floating Point Unit rounding mode.

63.13.30 gradtodeg

Synopsis: Convert grads angle to degrees angle

Declaration: `function gradtodeg(grad: Float) : Float`

Visibility: default

Description: `Gradtodeg` converts its argument `grad` (an angle in grads) to degrees. (100 grad is 90 degrees)

Errors: None.

See also: cycletorad (384), degtograd (385), radtodeg (408), radtograd (408), radtocycle (407), gradtorad (389)

Listing: ./mathex/ex15.pp

```
Program Example15;

{ Program to demonstrate the gradtodeg function. }

Uses math;

begin
  writeln(gradtodeg(100));
  writeln(gradtodeg(200));
  writeln(gradtodeg(300));
end.
```

63.13.31 gradtorad

Synopsis: Convert grads angle to radians angle

Declaration: `function gradtorad(grad: Float) : Float`

Visibility: default

Description: `Gradtorad` converts its argument `grad` (an angle in grads) to radians. (200 grad is pi degrees).

Errors: None.

See also: cycletorad (384), degtograd (385), radtodeg (408), radtograd (408), radtocycle (407), gradtodeg (389)

Listing: ./mathex/ex16.pp

```
Program Example16;

{ Program to demonstrate the gradtorad function. }

Uses math;

begin
  writeln(gradtorad(100));
  writeln(gradtorad(200));
  writeln(gradtorad(300));
end.
```

63.13.32 hypot

Synopsis: Return hypotenuse of triangle

Declaration: `function hypot(x: Float;y: Float) : Float`

Visibility: default

Description: `Hypot` returns the hypotenuse of the triangle where the sides adjacent to the square angle have lengths `x` and `y`. The function uses Pythagoras' rule for this.

Errors: None.

Listing: ./mathex/ex17.pp

```
Program Example17;

{ Program to demonstrate the hypot function. }

Uses math;

begin
  WriteLn(hypot(3,4)); // should be 5
end.
```

63.13.33 ifthen

Synopsis: Return one of two values, depending on a boolean condition

Declaration:
```
function ifthen(val: Boolean;const iftrue: Integer;
                const iffalse: Integer) : Integer;  Overload
function ifthen(val: Boolean;const iftrue: Int64;const iffalse: Int64)
                : Int64;  Overload
function ifthen(val: Boolean;const iftrue: Double;const iffalse: Double)
                : Double;  Overload
```

Visibility: default

Description: `ifthen` returns `iftrue` if `val` is `True`, and `iffalse` if `val` is `False`.

This function can be used in expressions.

63.13.34 InRange

Synopsis: Check whether value is in range.

Declaration:
```
function InRange(const AValue: Integer;const AMin: Integer;
                 const AMax: Integer) : Boolean;  Overload
function InRange(const AValue: Int64;const AMin: Int64;
                 const AMax: Int64) : Boolean;  Overload
function InRange(const AValue: Double;const AMin: Double;
                 const AMax: Double) : Boolean;  Overload
```

Visibility: default

Description: `InRange` returns `True` if `AValue` is in the range `AMin..AMax`. It returns `False` if `Value` lies outside the specified range.

See also: EnsureRange (386)

63.13.35 InterestRate

Synopsis: Calculate the interest rate value of an investment

Declaration:
```
function InterestRate(NPeriods: Integer;APayment: Float;
                      APresentValue: Float;AFutureValue: Float;
                      APaymentTime: tpaymenttime) : Float
```

Visibility: default

Description: `InterestRate` calculates the future value of an investment (`q`) in the cash flow formula (see CashFlowFunctions (371)). The function result is the interest rate value in case of a future value `AFutureValue` for an investment of a start value `APresentValue` (`PV`), where `APayment` (`PMT`) is invested for `NPeriods` (`n`).

The `APaymentTime` parameter determines whether the investment (payment) is an ordinary annuity or an annuity due: `ptEndOfPeriod` must be used if payments are at the end of each period. If the payments are at the beginning of the periode, `ptStartOfPeriod` must be used.

See also: FutureValue (388), NumberOfPeriods (404), Payment (404), PresentValue (407), TPaymentTime (377), CashFlowFunctions (371)

63.13.36 intpower

Synopsis: Return integer power.

Declaration:
```
function intpower(base: Float;const exponent: Integer) : Float
```

Visibility: default

Description: `Intpower` returns `base` to the power `exponent`, where exponent is an integer value.

Errors: If `base` is zero and the exponent is negative, then an overflow error will occur.

See also: power (406)

Listing: ./mathex/ex18.pp

```
Program Example18;

{ Program to demonstrate the intpower function. }

Uses math;

Procedure DoIntpower (X : extended; Pow : Integer);

begin
  writeln(X:8:4,'^',Pow:2,' = ',intpower(X,pow):8:4);
end;

begin
  dointpower(0.0,0);
  dointpower(1.0,0);
  dointpower(2.0,5);
  dointpower(4.0,3);
  dointpower(2.0,-1);
  dointpower(2.0,-2);
  dointpower(-2.0,4);
  dointpower(-4.0,3);
end.
```

63.13.37 IsInfinite

Synopsis: Check whether value is infinite

Declaration: `function IsInfinite(const d: Double) : Boolean`

Visibility: default

Description: `IsInfinite` returns `True` if the double `d` contains the infinite value.

See also: IsZero (392), IsInfinite (392)

63.13.38 IsNan

Synopsis: Check whether value is Not a Number

Declaration:
```
function IsNan(const d: Single) : Boolean;  Overload
function IsNan(const d: Double) : Boolean;  Overload
function IsNan(const d: Extended) : Boolean;  Overload
```

Visibility: default

Description: `IsNan` returns `True` if the double `d` contains Not A Number (a value which cannot be represented correctly in double format).

See also: IsZero (392), IsInfinite (392)

63.13.39 IsZero

Synopsis: Check whether value is zero

Declaration:
```
function IsZero(const A: Single;Epsilon: Single) : Boolean;  Overload
function IsZero(const A: Single) : Boolean;  Overload
function IsZero(const A: Double;Epsilon: Double) : Boolean;  Overload
function IsZero(const A: Double) : Boolean;  Overload
function IsZero(const A: Extended;Epsilon: Extended) : Boolean
                ;  Overload
function IsZero(const A: Extended) : Boolean;  Overload
```

Visibility: default

Description: `IsZero` checks whether the float value `A` is zero, up to a precision of `Epsilon`. It returns `True` if Abs(`A`) is less than `Epsilon`.

The default value for `Epsilon` depends on the type of the argument: it is 1E-4 for Single, 1E-12 for Double and 1E-16 for extended.

See also: IsNan (392), IsInfinite (392), SameValue (410)

63.13.40 ldexp

Synopsis: Return (2 to the power p) times x

Declaration:
```
function ldexp(x: Float;const p: Integer) : Float
```

Visibility: default

Description: `Ldexp` returns (2 to the power `p`) times `x`.

Errors: None.

See also: lnxp1 (393), log10 (394), log2 (394), logn (395)

Listing: ./mathex/ex19.pp

```
Program Example19;

{ Program to demonstrate the ldexp function. }

Uses math;

begin
  writeln(ldexp(2,4):8:4);
  writeln(ldexp(0.5,3):8:4);
end.
```

63.13.41 lnxp1

Synopsis: Return natural logarithm of 1+X

Declaration:
```
function lnxp1(x: Float) : Float
```

Visibility: default

Description: `Lnxp1` returns the natural logarithm of `1+X`. The result is more precise for small values of `x`. `x` should be larger than -1.

Errors: If $x\leq -1$ then an `EInvalidArgument` exception will be raised.

See also: ldexp (393), log10 (394), log2 (394), logn (395)

Listing: ./mathex/ex20.pp

```
Program Example20;

{ Program to demonstrate the lnxp1 function. }

Uses math;

begin
  writeln(lnxp1(0));
  writeln(lnxp1(0.5));
  writeln(lnxp1(1));
end.
```

63.13.42 log10

Synopsis: Return 10-Based logarithm.

Declaration: `function log10(x: Float) : Float`

Visibility: default

Description: `Log10` returns the 10-base logarithm of `X`.

Errors: If `x` is less than or equal to 0 an 'invalid fpu operation' error will occur.

See also: ldexp (393), lnxp1 (393), log2 (394), logn (395)

Listing: ./mathex/ex21.pp

```
Program Example21;

{ Program to demonstrate the log10 function. }

Uses math;

begin
  Writeln(Log10(10):8:4);
  Writeln(Log10(100):8:4);
  Writeln(Log10(1000):8:4);
  Writeln(Log10(1):8:4);
  Writeln(Log10(0.1):8:4);
  Writeln(Log10(0.01):8:4);
  Writeln(Log10(0.001):8:4);
end.
```

63.13.43 log2

Synopsis: Return 2-based logarithm

Declaration: `function log2(x: Float) : Float`

Visibility: default

Description: `Log2` returns the 2-base logarithm of `X`.

Errors: If `x` is less than or equal to 0 an 'invalid fpu operation' error will occur.

See also: ldexp (393), lnxp1 (393), log10 (394), logn (395)

Listing: ./mathex/ex22.pp

```
Program Example22;

{ Program to demonstrate the log2 function. }

Uses math;

begin
  Writeln(Log2(2):8:4);
  Writeln(Log2(4):8:4);
  Writeln(Log2(8):8:4);
  Writeln(Log2(1):8:4);
  Writeln(Log2(0.5):8:4);
  Writeln(Log2(0.25):8:4);
  Writeln(Log2(0.125):8:4);
end.
```

63.13.44 logn

Synopsis: Return N-based logarithm.

Declaration: `function logn(n: Float;x: Float) : Float`

Visibility: default

Description: `Logn` returns the n-base logarithm of `X`.

Errors: If `x` is less than or equal to 0 an 'invalid fpu operation' error will occur.

See also: ldexp (393), lnxp1 (393), log10 (394), log2 (394)

Listing: ./mathex/ex23.pp

```
Program Example23;

{ Program to demonstrate the logn function. }

Uses math;

begin
  Writeln(Logn(3,4):8:4);
  Writeln(Logn(2,4):8:4);
  Writeln(Logn(6,9):8:4);
  Writeln(Logn(exp(1),exp(1)):8:4);
  Writeln(Logn(0.5,1):8:4);
  Writeln(Logn(0.25,3):8:4);
  Writeln(Logn(0.125,5):8:4);
end.
```

63.13.45 Max

Synopsis: Return largest of 2 values

Declaration:
```
function Max(a: Integer;b: Integer) : Integer;  Overload
function Max(a: Int64;b: Int64) : Int64;  Overload
function Max(a: Single;b: Single) : Single;  Overload
function Max(a: Double;b: Double) : Double;  Overload
function Max(a: Extended;b: Extended) : Extended;  Overload
```

Visibility: default

Description: `Max` returns the maximum of `Int1` and `Int2`.

Errors: None.

See also: min (400), maxIntValue (396), maxvalue (397)

Listing: ./mathex/ex24.pp

```
Program Example24;

{ Program to demonstrate the max function. }

Uses math;

Var
  A,B : Cardinal;

begin
  A:=1;b:=2;
  writeln(max(a,b));
end.
```

63.13.46 MaxIntValue

Synopsis: Return largest element in integer array

Declaration:
```
function MaxIntValue(const Data: Array of Integer) : Integer
```

Visibility: default

Description: `MaxIntValue` returns the largest integer out of the `Data` array.

This function is provided for Delphi compatibility, use the maxvalue (397) function instead.

Errors: None.

See also: maxvalue (397), minvalue (401), minIntValue (400)

Listing: ./mathex/ex25.pp

```
Program Example25;

{ Program to demonstrate the MaxIntValue function. }

{ Make sore integer is 32 bit}
{$mode objfpc}
```

```
Uses math;

Type
  TExArray = Array[1..100] of Integer;

Var
  I : Integer;
  ExArray : TExArray;

begin
  Randomize;
  for I:=low(exarray) to high(exarray) do
    ExArray[i]:=Random(I)-Random(100);
  WriteIn(MaxIntValue(ExArray));
end.
```

63.13.47 maxvalue

Synopsis: Return largest value in array

Declaration:
```
function maxvalue(const data: Array of Single) : Single
function maxvalue(const data: PSingle;const N: Integer) : Single
function maxvalue(const data: Array of Double) : Double
function maxvalue(const data: PDouble;const N: Integer) : Double
function maxvalue(const data: Array of Extended) : Extended
function maxvalue(const data: PExtended;const N: Integer) : Extended
function maxvalue(const data: Array of Integer) : Integer
function maxvalue(const data: PInteger;const N: Integer) : Integer
```

Visibility: default

Description: `Maxvalue` returns the largest value in the `data` array with integer or float values. The return value has the same type as the elements of the array.

The third and fourth forms accept a pointer to an array of `N` integer or float values.

Errors: None.

See also: maxIntValue (396), minvalue (401), minIntValue (400)

Listing: ./mathex/ex26.pp

```
program Example26;

{ Program to demonstrate the MaxValue function. }

{ Make sore integer is 32 bit}
{$mode objfpc}

uses math;

var i:1..100;
    f_array:array[1..100] of Float;
    i_array:array[1..100] of Integer;
    Pf_array:Pfloat;
    PI_array:Pinteger;

begin
```

```
  randomize;

  Pf_array:=@f_array[1];
  Pi_array:=@i_array[1];

  for i:=low(f_array) to high(f_array) do
    f_array[i]:=(random-random)*100;
  for i:=low(i_array) to high(i_array) do
    i_array[i]:=random(l)-random(100);

  Writeln('Max Float       : ',MaxValue(f_array):8:4);
  Writeln('Max Float   (b) : ',MaxValue(Pf_array,100):8:4);
  Writeln('Max Integer     : ',MaxValue(i_array):8);
  Writeln('Max Integer (b) : ',MaxValue(Pi_array,100):8);
end.
```

63.13.48 mean

Synopsis: Return mean value of array

Declaration:
```
function mean(const data: Array of Single) : Float
function mean(const data: PSingle;const N: LongInt) : Float
function mean(const data: Array of Double) : Float
function mean(const data: PDouble;const N: LongInt) : Float
function mean(const data: Array of Extended) : Float
function mean(const data: PExtended;const N: LongInt) : Float
```

Visibility: default

Description: `Mean` returns the average value of `data`. The second form accepts a pointer to an array of `N` values.

Errors: None.

See also: meanandstddev (399), momentskewkurtosis (402), sum (414)

Listing: ./mathex/ex27.pp

```
Program Example27;

{ Program to demonstrate the Mean function. }

Uses math;

Type
  TExArray = Array[1..100] of Float;

Var
  I : Integer;
  ExArray : TExArray;

begin
  Randomize;
  for I:=low(ExArray) to high(ExArray) do
    ExArray[i]:=(Random-Random)*100;
  Writeln('Max       : ',MaxValue(ExArray):8:4);
  Writeln('Min       : ',MinValue(ExArray):8:4);
  Writeln('Mean      : ',Mean(ExArray):8:4);
```

```
  Writeln('Mean (b) : ',Mean(@ExArray[1],100):8:4);
end.
```

63.13.49 meanandstddev

Synopsis: Return mean and standard deviation of array

Declaration:
```
procedure meanandstddev(const data: Array of Single;var mean: Float;
                        var stddev: Float)
procedure meanandstddev(const data: PSingle;const N: LongInt;
                        var mean: Float;var stddev: Float)
procedure meanandstddev(const data: Array of Double;var mean: Float;
                        var stddev: Float)
procedure meanandstddev(const data: PDouble;const N: LongInt;
                        var mean: Float;var stddev: Float)
procedure meanandstddev(const data: Array of Extended;var mean: Float;
                        var stddev: Float)
procedure meanandstddev(const data: PExtended;const N: LongInt;
                        var mean: Float;var stddev: Float)
```

Visibility: default

Description: `meanandstddev` calculates the mean and standard deviation of `data` and returns the result in `mean` and `stddev`, respectively. Stddev is zero if there is only one value. The second form accepts a pointer to an array of `N` values.

Errors: None.

See also: mean (398), sum (414), sumofsquares (415), momentskewkurtosis (402)

Listing: ./mathex/ex28.pp

```
Program Example28;

{ Program to demonstrate the Meanandstddev function. }

Uses math;

Type
  TExArray = Array[1..100] of Extended;

Var
  I : Integer;
  ExArray : TExArray;
  Mean,stddev : Extended;

begin
  Randomize;
  for I:=low(ExArray) to high(ExArray) do
    ExArray[i]:=(Random-Random)*100;
  MeanAndStdDev(ExArray,Mean,StdDev);
  Writeln('Mean      : ',Mean:8:4);
  Writeln('StdDev    : ',StdDev:8:4);
  MeanAndStdDev(@ExArray[1],100,Mean,StdDev);
  Writeln('Mean (b)  : ',Mean:8:4);
  Writeln('StdDev (b) : ',StdDev:8:4);
end.
```

63.13.50 Min

Synopsis: Return smallest of two values.

Declaration:
```
function Min(a: Integer;b: Integer) : Integer;  Overload
function Min(a: Int64;b: Int64) : Int64;  Overload
function Min(a: Single;b: Single) : Single;  Overload
function Min(a: Double;b: Double) : Double;  Overload
function Min(a: Extended;b: Extended) : Extended;  Overload
```

Visibility: default

Description: `min` returns the smallest value of `Int1` and `Int2`;

Errors: None.

See also: max (396)

Listing: ./mathex/ex29.pp

```
Program Example29;

{ Program to demonstrate the min function. }

Uses math;

Var
  A,B : Cardinal;

begin
  A:=1;b:=2;
  writeln(min(a,b));
end.
```

63.13.51 MinIntValue

Synopsis: Return smallest value in integer array

Declaration:
```
function MinIntValue(const Data: Array of Integer) : Integer
```

Visibility: default

Description: `MinIntvalue` returns the smallest value in the `Data` array.

This function is provided for Delphi compatibility, use `minvalue` instead.

Errors: None

See also: minvalue (401), maxIntValue (396), maxvalue (397)

Listing: ./mathex/ex30.pp

```
Program Example30;

{ Program to demonstrate the MinIntValue function. }

{ Make sore integer is 32 bit}
{$mode objfpc}
```

```
Uses math;

Type
  TExArray = Array[1..100] of Integer;

Var
  I : Integer;
  ExArray : TExArray;

begin
  Randomize;
  for I:=low(ExArray) to high(ExArray) do
    ExArray[i]:=Random(I)-Random(100);
  WriteIn(MinIntValue(ExArray));
end.
```

63.13.52 minvalue

Synopsis: Return smallest value in array

Declaration:
```
function minvalue(const data: Array of Single) : Single
function minvalue(const data: PSingle;const N: Integer) : Single
function minvalue(const data: Array of Double) : Double
function minvalue(const data: PDouble;const N: Integer) : Double
function minvalue(const data: Array of Extended) : Extended
function minvalue(const data: PExtended;const N: Integer) : Extended
function minvalue(const data: Array of Integer) : Integer
function MinValue(const Data: PInteger;const N: Integer) : Integer
```

Visibility: default

Description: `Minvalue` returns the smallest value in the `data` array with integer or float values. The return value has the same type as the elements of the array.

The third and fourth forms accept a pointer to an array of `N` integer or float values.

Errors: None.

See also: maxIntValue (396), maxvalue (397), minIntValue (400)

Listing: ./mathex/ex31.pp

```
program Example31;

{ Program to demonstrate the MinValue function. }

{ Make sure integer is 32 bit}
{$mode objfpc}

uses math;

var i:1..100;
    f_array:array[1..100] of Float;
    i_array:array[1..100] of Integer;
    Pf_array:Pfloat;
    PI_array:Pinteger;

begin
```

```
  randomize;

  Pf_array:=@f_array[1];
  Pi_array:=@i_array[1];

  for i:=low(f_array) to high(f_array) do
    f_array[i]:=(random-random)*100;
  for i:=low(i_array) to high(i_array) do
    i_array[i]:=random(l)-random(100);

  Writeln('Min Float       : ',MinValue(f_array):8:4);
  Writeln('Min Float   (b) : ',MinValue(Pf_array,100):8:4);
  Writeln('Min Integer     : ',MinValue(i_array):8);
  Writeln('Min Integer (b) : ',MinValue(Pi_array,100):8);
end.
```

63.13.53 momentskewkurtosis

Synopsis: Return 4 first moments of distribution

Declaration:
```
procedure momentskewkurtosis(const data: Array of Single;out m1: Float;
                             out m2: Float;out m3: Float;out m4: Float;
                             out skew: Float;out kurtosis: Float)
procedure momentskewkurtosis(const data: PSingle;const N: Integer;
                             out m1: Float;out m2: Float;out m3: Float;
                             out m4: Float;out skew: Float;
                             out kurtosis: Float)
procedure momentskewkurtosis(const data: Array of Double;out m1: Float;
                             out m2: Float;out m3: Float;out m4: Float;
                             out skew: Float;out kurtosis: Float)
procedure momentskewkurtosis(const data: PDouble;const N: Integer;
                             out m1: Float;out m2: Float;out m3: Float;
                             out m4: Float;out skew: Float;
                             out kurtosis: Float)
procedure momentskewkurtosis(const data: Array of Extended;
                             out m1: Float;out m2: Float;out m3: Float;
                             out m4: Float;out skew: Float;
                             out kurtosis: Float)
procedure momentskewkurtosis(const data: PExtended;const N: Integer;
                             out m1: Float;out m2: Float;out m3: Float;
                             out m4: Float;out skew: Float;
                             out kurtosis: Float)
```

Visibility: default

Description: `momentskewkurtosis` calculates the 4 first moments of the distribution of valuesin `data` and returns them in `m1`,`m2`,`m3` and `m4`, as well as the `skew` and `kurtosis`.

Errors: None.

See also: mean (398), meanandstddev (399)

Listing: ./mathex/ex32.pp

```
program Example32;
```

```
{ Program to demonstrate the momentskewkurtosis function. }

uses math;

var distarray:array[1..1000] of float;
    l:longint;
    m1,m2,m3,m4,skew,kurtosis:float;

begin
  randomize;
  for l:=low(distarray) to high(distarray) do
    distarray[i]:=random;
  momentskewkurtosis(DistArray,m1,m2,m3,m4,skew,kurtosis);

  Writeln('1st moment : ',m1:8:6);
  Writeln('2nd moment : ',m2:8:6);
  Writeln('3rd moment : ',m3:8:6);
  Writeln('4th moment : ',m4:8:6);
  Writeln('Skew       : ',skew:8:6);
  Writeln('kurtosis   : ',kurtosis:8:6);
end.
```

63.13.54 norm

Synopsis: Return Euclidian norm

Declaration:
```
function norm(const data: Array of Single) : Float
function norm(const data: PSingle;const N: Integer) : Float
function norm(const data: Array of Double) : Float
function norm(const data: PDouble;const N: Integer) : Float
function norm(const data: Array of Extended) : Float
function norm(const data: PExtended;const N: Integer) : Float
```

Visibility: default

Description: `Norm` calculates the Euclidian norm of the array of data. This equals `sqrt(sumofsquares(data))`. The second form accepts a pointer to an array of `N` values.

Errors: None.

See also: sumofsquares (415)

Listing: ./mathex/ex33.pp

```
program Example33;

{ Program to demonstrate the norm function. }

uses math;

var v:array[1..10] of Float;
    l:1..10;

begin
  for l:=low(v) to high(v) do
    v[i]:=random;
  writeln(norm(v));
end.
```

63.13.55 NumberOfPeriods

Synopsis: Calculate the number of periods for an investment

Declaration:
```
function NumberOfPeriods(ARate: Float;APayment: Float;
                         APresentValue: Float;AFutureValue: Float;
                         APaymentTime: tpaymenttime) : Float
```

Visibility: default

Description: `NumberOfPeriods` calculates the number of periods (`n`) needed to obtain future value of an investment in the cash flow formula (see CashFlowFunctions (371)). The function result is the number of periods a payment `APayment` (`PMT`) must be paid in order to obtain a future value `AFutureValue` for an investment of a start value `APresentValue` (`PV`), where `APayment` (`PMT`) is invested at a rate `ARate` (`q`).

The `APaymentTime` parameter determines whether the investment (payment) is an ordinary annuity or an annuity due: `ptEndOfPeriod` NumberOfPeriodsmust be used if payments are at the end of each period. If the payments are at the beginning of the periode, `ptStartOfPeriod` must be used.

See also: FutureValue (388), InterestRate (391), Payment (404), PresentValue (407), TPaymentTime (377), CashFlowFunctions (371)

63.13.56 Payment

Synopsis: Calculate the payment for an investment

Declaration:
```
function Payment(ARate: Float;NPeriods: Integer;APresentValue: Float;
                 AFutureValue: Float;APaymentTime: tpaymenttime) : Float
```

Visibility: default

Description: `Payment` calculates the amount that must be payed (`PMT`) during number of periods (`n`) needed to obtain future value of an investment in the cash flow formula (see CashFlowFunctions (371)). The function result is the amount (`PMT`) that must be paid in order to obtain a future value `AFutureValue` for an investment of a start value `APresentValue` (`PV`), where the amount must be payed `NPeriods` (`PMT`) and the interest rate is`ARate` (`q`).

The `APaymentTime` parameter determines whether the investment (payment) is an ordinary annuity or an annuity due: `ptEndOfPeriod` NumberOfPeriodsmust be used if payments are at the end of each period. If the payments are at the beginning of the periode, `ptStartOfPeriod` must be used.

See also: FutureValue (388), InterestRate (391), NumberOfPeriods (404), PresentValue (407), TPaymentTime (377), CashFlowFunctions (371)

63.13.57 popnstddev

Synopsis: Return population variance

Declaration:
```
function popnstddev(const data: Array of Single) : Float
function popnstddev(const data: PSingle;const N: Integer) : Float
function popnstddev(const data: Array of Double) : Float
function popnstddev(const data: PDouble;const N: Integer) : Float
function popnstddev(const data: Array of Extended) : Float
function popnstddev(const data: PExtended;const N: Integer) : Float
```

Visibility: default

Description: `Popnstddev` returns the square root of the population variance of the values in the `Data` array. It returns zero if there is only one value.

The second form of this function accepts a pointer to an array of `N` values.

Errors: None.

See also: popnvariance (405), mean (398), meanandstddev (399), stddev (413), momentskewkurtosis (402)

Listing: ./mathex/ex35.pp

```
Program Example35;

{ Program to demonstrate the PopnStdDev function. }

Uses Math;

Type
  TExArray = Array[1..100] of Float;

Var
  I : Integer;
  ExArray : TExArray;

begin
  Randomize;
  for I:=low(ExArray) to high(ExArray) do
    ExArray[i]:=(Random-Random)*100;
  Writeln('Max               : ',MaxValue(ExArray):8:4);
  Writeln('Min               : ',MinValue(ExArray):8:4);
  Writeln('Pop. stddev.      : ',PopnStdDev(ExArray):8:4);
  Writeln('Pop. stddev. (b) : ',PopnStdDev(@ExArray[1],100):8:4);
end.
```

63.13.58 popnvariance

Synopsis: Return population variance

Declaration:
```
function popnvariance(const data: PSingle;const N: Integer) : Float
function popnvariance(const data: Array of Single) : Float
function popnvariance(const data: PDouble;const N: Integer) : Float
function popnvariance(const data: Array of Double) : Float
function popnvariance(const data: PExtended;const N: Integer) : Float
function popnvariance(const data: Array of Extended) : Float
```

Visibility: default

Description: `Popnvariance` the population variance of the values in the `Data` array. It returns zero if there is only one value.

The second form of this function accepts a pointer to an array of `N` values.

Errors: None.

See also: popnstddev (404), mean (398), meanandstddev (399), stddev (413), momentskewkurtosis (402)

Listing: ./mathex/ex36.pp

```
Program Example36;

{ Program to demonstrate the PopnVariance function. }

Uses math;

Var
  I : Integer;
  ExArray : Array[1..100] of Float;

begin
  Randomize;
  for I:=low(ExArray) to high(ExArray) do
    ExArray[i]:=(Random-Random)*100;
  Writeln('Max            : ',MaxValue(ExArray):8:4);
  Writeln('Min            : ',MinValue(ExArray):8:4);
  Writeln('Pop. var.      : ',PopnVariance(ExArray):8:4);
  Writeln('Pop. var. (b) : ',PopnVariance(@ExArray[1],100):8:4);
end.
```

63.13.59 power

Synopsis: Return real power.

Declaration: `function power(base: Float;exponent: Float) : Float`

Visibility: default

Description: `power` raises `base` to the power `power`. This is equivalent to `exp(power*ln(base))`. Therefore `base` should be non-negative.

Errors: None.

See also: intpower (391)

Listing: ./mathex/ex34.pp

```
Program Example34;

{ Program to demonstrate the power function. }

Uses Math;

procedure dopower(x,y : float);

begin
  writeln(x:8:6,'^',y:8:6,' = ',power(x,y):8:6)
end;

begin
  dopower(2,2);
  dopower(2,-2);
  dopower(2,0.0);
end.
```

63.13.60 power(Float,Float):Float

Declaration: `operator **(bas: Float;expo: Float) : Float`

Visibility: default

63.13.61 power(Int64,Int64):Int64

Declaration: `operator **(bas: Int64;expo: Int64) : Int64`

Visibility: default

63.13.62 PresentValue

Synopsis: Calculate the present value given the future value of an investment.

Declaration:
```
function PresentValue(ARate: Float;NPeriods: Integer;APayment: Float;
                      AFutureValue: Float;APaymentTime: tpaymenttime)
                       : Float
```

Visibility: default

Description: `PresentValue` calculates the present (start) value of an investment (`PV`) in the cash flow formula (see CashFlowFunctions (371)) The function result is the start value of an investment, when the future value is `AFutureValue` (`FV`) and `APayment` (`PMT`) is invested for `NPeriods` (`n`) at the rate of `ARate` (`q`) per period.

The `APaymentTime` parameter determines whether the investment (payment) is an ordinary annuity or an annuity due: `ptEndOfPeriod` must be used if payments are at the end of each period. If the payments are at the beginning of the periode, `ptStartOfPeriod` must be used.

See also: FutureValue (388), InterestRate (391), NumberOfPeriods (404), Payment (404), TPaymentTime (377), CashFlowFunctions (371)

63.13.63 radtocycle

Synopsis: Convert radians angle to cycle angle

Declaration: `function radtocycle(rad: Float) : Float`

Visibility: default

Description: `Radtocycle` converts its argument `rad` (an angle expressed in radians) to an angle in cycles.

(1 cycle equals 2 `pi` radians)

Errors: None.

See also: degtograd (385), degtorad (386), radtodeg (408), radtograd (408), cycletorad (384)

Listing: ./mathex/ex37.pp

```
Program Example37;

{ Program to demonstrate the radtocycle function. }

Uses math;

begin
```

```
  writeln(radtocycle(2*pi):8:6);
  writeln(radtocycle(pi):8:6);
  writeln(radtocycle(pi/2):8:6);
end.
```

63.13.64 radtodeg

Synopsis: Convert radians angle to degrees angle

Declaration: `function radtodeg(rad: Float) : Float`

Visibility: default

Description: `Radtodeg` converts its argument `rad` (an angle expressed in radians) to an angle in degrees. (180 degrees equals pi radians)

Errors: None.

See also: degtograd (385), degtorad (386), radtocycle (407), radtograd (408), cycletorad (384)

Listing: ./mathex/ex38.pp

```
Program Example38;

{ Program to demonstrate the radtodeg function. }

Uses math;

begin
  writeln(radtodeg(2*pi):8:6);
  writeln(radtodeg(pi):8:6);
  writeln(radtodeg(pi/2):8:6);
end.
```

63.13.65 radtograd

Synopsis: Convert radians angle to grads angle

Declaration: `function radtograd(rad: Float) : Float`

Visibility: default

Description: `Radtodeg` converts its argument `rad` (an angle expressed in radians) to an angle in grads. (200 grads equals pi radians)

Errors: None.

See also: degtograd (385), degtorad (386), radtocycle (407), radtodeg (408), cycletorad (384)

Listing: ./mathex/ex39.pp

```
Program Example39;

{ Program to demonstrate the radtograd function. }

Uses math;
```

```
begin
  writeln(radtograd(2*pi):8:6);
  writeln(radtograd(pi):8:6);
  writeln(radtograd(pi/2):8:6);
end.
```

63.13.66 randg

Synopsis: Return gaussian distributed random number.

Declaration: `function randg(mean: Float;stddev: Float) : Float`

Visibility: default

Description: `randg` returns a random number which - when produced in large quantities - has a Gaussian distribution with mean `mean` and standarddeviation `stddev`.

Errors: None.

See also: mean (398), stddev (413), meanandstddev (399)

Listing: ./mathex/ex40.pp

```
Program Example40;

{ Program to demonstrate the randg function. }

Uses Math;

Var
  I : Integer;
  ExArray : Array[1..10000] of Float;;
  Mean,stddev : Float;

begin
  Randomize;
  for I:=low(ExArray) to high(ExArray) do
    ExArray[i]:=Randg(1,0.2);
  MeanAndStdDev(ExArray,Mean,StdDev);
  Writeln('Mean      : ',Mean:8:4);
  Writeln('StdDev    : ',StdDev:8:4);
end.
```

63.13.67 RandomFrom

Synopsis: Return a random element of an array of numbers

Declaration:
```
function RandomFrom(const AValues: Array of Double) : Double;  Overload
function RandomFrom(const AValues: Array of Integer) : Integer
                    ;  Overload
function RandomFrom(const AValues: Array of Int64) : Int64;  Overload
```

Visibility: default

Description: `RandomFrom` returns a random element from the array `AValues`. The return value has the same type as the type of the array elements.

See also: #rtl.system.Random (622), RandomRange (410)

63.13.68 RandomRange

Synopsis: Return a random number in a range

Declaration:
```
function RandomRange(const aFrom: Integer;const aTo: Integer) : Integer
function RandomRange(const aFrom: Int64;const aTo: Int64) : Int64
```

Visibility: default

Description: `RandomRange` returns a random value in the range `AFrom` to `ATo`. `AFrom` and `ATo` do not need to be in increasing order. The upper border is not included in the generated value, but the lower border is.

See also: #rtl.system.Random (622), RandomFrom (409)

63.13.69 RoundTo

Synopsis: Round to the specified number of digits

Declaration:
```
function RoundTo(const AValue: Double;const Digits: TRoundToRange)
                 : Double
function RoundTo(const AVAlue: Extended;const Digits: TRoundToRange)
                 : Extended
function RoundTo(const AValue: Single;const Digits: TRoundToRange)
                 : Single
```

Visibility: default

Description: `RoundTo` rounds the specified float `AVAlue` to the specified number of digits and returns the result. This result is accurate to "10 to the power Digits". It uses the standard `Round` function for this.

See also: TRoundToRange (377), SimpleRoundTo (412)

63.13.70 SameValue

Synopsis: Check whether 2 float values are the same

Declaration:
```
function SameValue(const A: Extended;const B: Extended) : Boolean
                   ;  Overload
function SameValue(const A: Double;const B: Double) : Boolean;  Overload
function SameValue(const A: Single;const B: Single) : Boolean;  Overload
function SameValue(const A: Extended;const B: Extended;
                   Epsilon: Extended) : Boolean;  Overload
function SameValue(const A: Double;const B: Double;Epsilon: Double)
                    : Boolean;  Overload
function SameValue(const A: Single;const B: Single;Epsilon: Single)
                    : Boolean;  Overload
```

Visibility: default

Description: `SameValue` returns `True` if the floating-point values `A` and `B` are the same, i.e. whether the absolute value of their their difference is smaller than `Epsilon`. If their difference is larger, then `False` is returned.

If unspecified, the default value for `Epsilon` is 0.0.

See also: MinFloat (375), IsZero (392)

63.13.71 sec

Synopsis: Alias for `secant`

Declaration:
```
function sec(x: Float) : Float
```

Visibility: default

Description: `sec` is an alias for the secant (411) function.

See also: secant (411)

63.13.72 secant

Synopsis: Calculate secant

Declaration:
```
function secant(x: Float) : Float
```

Visibility: default

Description: `Secant` calculates the secant (1/`cos(x)`) of its argument `x`.

Errors: If 90 or 270 degrees is specified, an exception will be raised.

See also: cosecant (383)

63.13.73 SetExceptionMask

Synopsis: Set the Floating Point Unit exception mask.

Declaration:
```
function SetExceptionMask(const Mask: TFPUExceptionMask)
                                  : TFPUExceptionMask
```

Visibility: default

Description: Set the Floating Point Unit exception mask.

63.13.74 SetPrecisionMode

Synopsis: Set the Floating Point Unit precision mode.

Declaration:
```
function SetPrecisionMode(const Precision: TFPUPrecisionMode)
                                  : TFPUPrecisionMode
```

Visibility: default

Description: Set the Floating Point Unit precision mode.

63.13.75 SetRoundMode

Synopsis: Set the Floating Point Unit rounding mode.

Declaration:
```
function SetRoundMode(const RoundMode: TFPURoundingMode)
                               : TFPURoundingMode
```

Visibility: default

Description: Set the Floating Point Unit rounding mode.

63.13.76 Sign

Synopsis: Return sign of argument

Declaration:
```
function Sign(const AValue: Integer) : TValueSign;  Overload
function Sign(const AValue: Int64) : TValueSign;  Overload
function Sign(const AValue: Single) : TValueSign;  Overload
function Sign(const AValue: Double) : TValueSign;  Overload
function Sign(const AValue: Extended) : TValueSign;  Overload
```

Visibility: default

Description: `Sign` returns the sign of it's argument, which can be an Integer, 64 bit integer, or a double. The returned value is an integer which is -1, 0 or 1, and can be used to do further calculations with.

63.13.77 SimpleRoundTo

Synopsis: Round to the specified number of digits (rounding up if needed)

Declaration:
```
function SimpleRoundTo(const AValue: Single;const Digits: TRoundToRange)
                        : Single
function SimpleRoundTo(const AValue: Double;const Digits: TRoundToRange)
                        : Double
function SimpleRoundTo(const AValue: Extended;
                       const Digits: TRoundToRange) : Extended
```

Visibility: default

Description: `SimpleRoundTo` rounds the specified float `AValue` to the specified number of digits, but rounds up, and returns the result. This result is accurate to "10 to the power Digits". It uses the standard `Round` function for this.

See also: TRoundToRange (377), RoundTo (410)

63.13.78 sincos

Synopsis: Return sine and cosine of argument

Declaration:
```
procedure sincos(theta: single;out sinus: single;out cosinus: single)
procedure sincos(theta: Double;out sinus: Double;out cosinus: Double)
procedure sincos(theta: extended;out sinus: extended;
                 out cosinus: extended)
```

Visibility: default

Description: `Sincos` calculates the sine and cosine of the angle `theta`, and returns the result in `sinus` and `cosinus`.

On Intel hardware, This calculation will be faster than making 2 calls to calculate the sine and cosine separately.

Errors: None.

See also: arcsin (379), arccos (377)

Listing: ./mathex/ex41.pp

```
Program Example41;

{ Program to demonstrate the sincos function. }

Uses math;

Procedure dosincos(Angle : Float);

Var
  Sine,Cosine : Float;

begin
  sincos(angle,sine,cosine);
  Write('Angle : ',Angle:8:6);
  Write(' Sine :',sine:8:6);
  Write(' Cosine :',cosine:8:6);
end;

begin
  dosincos(pi);
  dosincos(pi/2);
  dosincos(pi/3);
  dosincos(pi/4);
  dosincos(pi/6);
end.
```

63.13.79 sinh

Synopsis: Return hyperbolic sine

Declaration: `function sinh(x: Float) : Float`

Visibility: default

Description: `Sinh` returns the hyperbolic sine of its argument `x`.

See also: cosh (383), arsinh (380), tanh (418), artanh (381)

Listing: ./mathex/ex42.pp

```
Program Example42;

{ Program to demonstrate the sinh function. }

Uses math;

begin
  writeln(sinh(0));
  writeln(sinh(1));
  writeln(sinh(-1));
end.
```

63.13.80 stddev

Synopsis: Return standard deviation of data

Declaration:
```
function stddev(const data: Array of Single) : Float
function stddev(const data: PSingle;const N: Integer) : Float
function stddev(const data: Array of Double) : Float
function stddev(const data: PDouble;const N: Integer) : Float
function stddev(const data: Array of Extended) : Float
function stddev(const data: PExtended;const N: Integer) : Float
```

Visibility: default

Description: `Stddev` returns the standard deviation of the values in `Data`. It returns zero if there is only one value.

The second form of the function accepts a pointer to an array of `N` values.

Errors: None.

See also: mean (398), meanandstddev (399), variance (419), totalvariance (418)

Listing: ./mathex/ex43.pp

```
Program Example40;

{ Program to demonstrate the stddev function. }

Uses Math;

Var
  I : Integer;
  ExArray : Array[1..10000] of Float;

begin
  Randomize;
  for I:=low(ExArray) to high(ExArray) do
    ExArray[i]:=Randg(1,0.2);
  Writeln('StdDev      : ',StdDev(ExArray):8:4);
  Writeln('StdDev (b) : ',StdDev(@ExArray[0],10000):8:4);
end.
```

63.13.81 sum

Synopsis: Return sum of values

Declaration:
```
function sum(const data: Array of Single) : Float
function sum(const data: PSingle;const N: LongInt) : Float
function sum(const data: Array of Double) : Float
function sum(const data: PDouble;const N: LongInt) : Float
function sum(const data: Array of Extended) : Float
function sum(const data: PExtended;const N: LongInt) : Float
```

Visibility: default

Description: `Sum` returns the sum of the values in the `data` array.

The second form of the function accepts a pointer to an array of `N` values.

Errors: None.

See also: sumofsquares (415), sumsandsquares (416), totalvariance (418), variance (419)

Listing: ./mathex/ex44.pp

```
Program Example44;

{ Program to demonstrate the Sum function. }

Uses math;

Var
  I : 1..100;
  ExArray : Array[1..100] of Float;

begin
  Randomize;
  for I:=low(ExArray) to high(ExArray) do
    ExArray[i]:=(Random-Random)*100;
  Writeln('Max      : ',MaxValue(ExArray):8:4);
  Writeln('Min      : ',MinValue(ExArray):8:4);
  Writeln('Sum      : ',Sum(ExArray):8:4);
  Writeln('Sum (b) : ',Sum(@ExArray[1],100):8:4);
end.
```

63.13.82 sumInt

Synopsis: Return the sum of an array of integers

Declaration:
```
function sumInt(const data: PInt64;const N: LongInt) : Int64
function sumInt(const data: Array of Int64) : Int64
```

Visibility: default

Description: `SumInt` returns the sum of the `N` integers in the `Data` array, where this can be an open array or a pointer to an array.

Errors: An overflow may occur.

63.13.83 sumofsquares

Synopsis: Return sum of squares of values

Declaration:
```
function sumofsquares(const data: Array of Single) : Float
function sumofsquares(const data: PSingle;const N: Integer) : Float
function sumofsquares(const data: Array of Double) : Float
function sumofsquares(const data: PDouble;const N: Integer) : Float
function sumofsquares(const data: Array of Extended) : Float
function sumofsquares(const data: PExtended;const N: Integer) : Float
```

Visibility: default

Description: `Sumofsquares` returns the sum of the squares of the values in the `data` array.

The second form of the function accepts a pointer to an array of `N` values.

Errors: None.

See also: sum (414), sumsandsquares (416), totalvariance (418), variance (419)

Listing: ./mathex/ex45.pp

```
Program Example45;

{ Program to demonstrate the SumOfSquares function. }

Uses math;

Var
  I : 1..100;
  ExArray : Array[1..100] of Float;

begin
  Randomize;
  for I:=low(ExArray) to high(ExArray) do
    ExArray[i]:=(Random-Random)*100;
  Writeln('Max              : ',MaxValue(ExArray):8:4);
  Writeln('Min              : ',MinValue(ExArray):8:4);
  Writeln('Sum squares      : ',SumOfSquares(ExArray):8:4);
  Writeln('Sum squares (b) : ',SumOfSquares(@ExArray[1],100):8:4);
end.
```

63.13.84 sumsandsquares

Synopsis: Return sum and sum of squares of values.

Declaration:
```
procedure sumsandsquares(const data: Array of Single;var sum: Float;
                         var sumofsquares: Float)
procedure sumsandsquares(const data: PSingle;const N: Integer;
                         var sum: Float;var sumofsquares: Float)
procedure sumsandsquares(const data: Array of Double;var sum: Float;
                         var sumofsquares: Float)
procedure sumsandsquares(const data: PDouble;const N: Integer;
                         var sum: Float;var sumofsquares: Float)
procedure sumsandsquares(const data: Array of Extended;var sum: Float;
                         var sumofsquares: Float)
procedure sumsandsquares(const data: PExtended;const N: Integer;
                         var sum: Float;var sumofsquares: Float)
```

Visibility: default

Description: `sumsandsquares` calculates the sum of the values and the sum of the squares of the values in the `data` array and returns the results in `sum` and `sumofsquares`.

The second form of the function accepts a pointer to an array of `N` values.

Errors: None.

See also: sum (414), sumofsquares (415), totalvariance (418), variance (419)

Listing: ./mathex/ex46.pp

```
Program Example45;

{ Program to demonstrate the SumOfSquares function. }

Uses math;

Var
```

```
  I : 1..100;
  ExArray : Array[1..100] of Float
  s,ss : float;

begin
  Randomize;
  for I:=low(ExArray) to high(ExArray) do
    ExArray[i]:=(Random-Random)*100;
  Writeln('Max              : ',MaxValue(ExArray):8:4);
  Writeln('Min              : ',MinValue(ExArray):8:4);
  SumsAndSquares(ExArray,S,SS);
  Writeln('Sum              : ',S:8:4);
  Writeln('Sum squares      : ',SS:8:4);
  SumsAndSquares(@ExArray[1],100,S,SS);
  Writeln('Sum (b)          : ',S:8:4);
  Writeln('Sum squares (b) : ',SS:8:4);
end.
```

63.13.85 tan

Synopsis: Return tangent

Declaration: `function tan(x: Float) : Float`

Visibility: default

Description: `Tan` returns the tangent of `x`. The argument `x` must be in radians.

Errors: If `x` (normalized) is pi/2 or 3pi/2 then an overflow will occur.

See also: tanh (418), arcsin (379), sincos (412), arccos (377)

Listing: ./mathex/ex47.pp

```
Program Example47;

{ Program to demonstrate the Tan function. }

Uses math;

Procedure DoTan(Angle : Float);

begin
  Write('Angle : ',RadToDeg(Angle):8:6);
  Writeln(' Tangent : ',Tan(Angle):8:6);
end;

begin
  DoTan(0);
  DoTan(Pi);
  DoTan(Pi/3);
  DoTAn(Pi/4);
  DoTan(Pi/6);
end.
```

63.13.86 tanh

Synopsis: Return hyperbolic tangent

Declaration:
```
function tanh(x: Float) : Float
```

Visibility: default

Description: `Tanh` returns the hyperbolic tangent of `x`.

Errors: None.

See also: arcsin (379), sincos (412), arccos (377)

Listing: ./mathex/ex48.pp

```
Program Example48;

{ Program to demonstrate the Tanh function. }

Uses math;

begin
  writeln(tanh(0));
  writeln(tanh(1));
  writeln(tanh(-1));
end.
```

63.13.87 totalvariance

Synopsis: Return total varians of values

Declaration:
```
function totalvariance(const data: Array of Single) : Float
function totalvariance(const data: PSingle;const N: Integer) : Float
function totalvariance(const data: Array of Double) : Float
function totalvariance(const data: PDouble;const N: Integer) : Float
function totalvariance(const data: Array of Extended) : Float
function totalvariance(const data: PExtended;const N: Integer) : Float
```

Visibility: default

Description: `TotalVariance` returns the total variance of the values in the `data` array. It returns zero if there is only one value.

The second form of the function accepts a pointer to an array of `N` values.

Errors: None.

See also: variance (419), stddev (413), mean (398)

Listing: ./mathex/ex49.pp

```
Program Example49;

{ Program to demonstrate the TotalVariance function. }

Uses math;

Type
```

```
  TExArray = Array[1..100] of Float;

Var
  I : Integer;
  ExArray : TExArray;
  TV : float;

begin
  Randomize;
  for I:=1 to 100 do
    ExArray[i]:=(Random-Random)*100;
  TV:=TotalVariance(ExArray);
  Writeln('Total variance      : ',TV:8:4);
  TV:=TotalVariance(@ExArray[1],100);
  Writeln('Total Variance (b) : ',TV:8:4);
end.
```

63.13.88 variance

Synopsis: Return variance of values

Declaration:
```
function variance(const data: Array of Single) : Float
function variance(const data: PSingle;const N: Integer) : Float
function variance(const data: Array of Double) : Float
function variance(const data: PDouble;const N: Integer) : Float
function variance(const data: Array of Extended) : Float
function variance(const data: PExtended;const N: Integer) : Float
```

Visibility: default

Description: `Variance` returns the variance of the values in the `data` array. It returns zero if there is only one value.

The second form of the function accepts a pointer to an array of `N` values.

Errors: None.

See also: totalvariance (418), stddev (413), mean (398)

Listing: ./mathex/ex50.pp

```
Program Example50;

{ Program to demonstrate the Variance function. }

Uses math;

Var
  I : 1..100;
  ExArray : Array[1..100] of Float;
  V : float;

begin
  Randomize;
  for I:=low(ExArray) to high(ExArray) do
    ExArray[i]:=(Random-Random)*100;
  V:=Variance(ExArray);
  Writeln('Variance      : ',V:8:4);
```

```
  V:=Variance(@ExArray[1],100);
  Writeln('Variance (b) : ',V:8:4);
end.
```

63.14 EInvalidArgument

63.14.1 Description

Exception raised when invalid arguments are passed to a function.

Chapter 64

Reference for unit 'matrix'

64.1 Overview

The unit matrix is a unit that provides objects for the common two, three and four dimensional vectors matrixes. These vectors and matrixes are very common in computer graphics and are often implemented from scratch by programmers while every implementation provides exactly the same functionality.

It makes therefore sense to provide this functionality in the runtime library. This eliminates the need for programmers to reinvent the wheel and also allows libraries that use matrix operations to become more compatible.

The matrix unit does not provide n-dimensional matrixes. The functionality needs of a general matrix unit varies from application to application; one can think of reduced memory usage tricks for matrixes that only have data around the diagonal etc., desire for parallelization etc. etc. It is believed that programmers that do use n-dimensional matrices would not necessarily benfit from such a unit in the runtime library.

Design goals:

- Provide common dimensions, two three and four.
- Provide multiple floating point precisions, single, double, extended.
- Simple trivial binary representation; it is possible to typecast vectors into other implementations that use the same trivial representation.
- No dynamic memory management in the background. It must be possible to write expressions like matrix A * B * C without worrying about memory management.

Design decisions:

- Class object model is ruled out. The objects object model, without virtual methods, is suitable.
- Operator overloading is a good way to allow programmers to write matrix expressions.
- 3 dimensions * 3 precision means 9 vector and 9 matrix objects. Macro's have been used in the source to take care of this.

64.2 Constants, types and variables

64.2.1 Types

```
Tmatrix2_double_data = Array[0..1,0..1] of Double
```

This is the matrix internal data for a matrix. It uses a simple array structure so data from other libraries that define their own matrix type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tmatrix2_extended_data = Array[0..1,0..1] of extended
```

This is the matrix internal data for a matrix. It uses a simple array structure so data from other libraries that define their own matrix type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tmatrix2_single_data = Array[0..1,0..1] of single
```

This is the matrix internal data for a matrix. It uses a simple array structure so data from other libraries that define their own matrix type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tmatrix3_double_data = Array[0..2,0..2] of Double
```

This is the matrix internal data for a matrix. It uses a simple array structure so data from other libraries that define their own matrix type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tmatrix3_extended_data = Array[0..2,0..2] of extended
```

This is the matrix internal data for a matrix. It uses a simple array structure so data from other libraries that define their own matrix type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tmatrix3_single_data = Array[0..2,0..2] of single
```

This is the matrix internal data for a matrix. It uses a simple array structure so data from other libraries that define their own matrix type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tmatrix4_double_data = Array[0..3,0..3] of Double
```

This is the matrix internal data for a matrix. It uses a simple array structure so data from other libraries that define their own matrix type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tmatrix4_extended_data = Array[0..3,0..3] of extended
```

This is the matrix internal data for a matrix. It uses a simple array structure so data from other libraries that define their own matrix type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tmatrix4_single_data = Array[0..3,0..3] of single
```

This is the matrix internal data for a matrix. It uses a simple array structure so data from other libraries that define their own matrix type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tvector2_double_data = Array[0..1] of Double
```

This is the vector internal data for a vector. It uses a simple array structure so data from other libraries that define their own vector type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tvector2_extended_data = Array[0..1] of extended
```

This is the vector internal data for a vector. It uses a simple array structure so data from other libraries that define their own vector type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tvector2_single_data = Array[0..1] of single
```

This is the vector internal data for a vector. It uses a simple array structure so data from other libraries that define their own vector type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tvector3_double_data = Array[0..2] of Double
```

This is the vector internal data for a vector. It uses a simple array structure so data from other libraries that define their own vector type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tvector3_extended_data = Array[0..2] of extended
```

This is the vector internal data for a vector. It uses a simple array structure so data from other libraries that define their own vector type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tvector3_single_data = Array[0..2] of single
```

This is the vector internal data for a vector. It uses a simple array structure so data from other libraries that define their own vector type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tvector4_double_data = Array[0..3] of Double
```

This is the vector internal data for a vector. It uses a simple array structure so data from other libraries that define their own vector type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tvector4_extended_data = Array[0..3] of extended
```

This is the vector internal data for a vector. It uses a simple array structure so data from other libraries that define their own vector type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tvector4_single_data = Array[0..3] of single
```

This is the vector internal data for a vector. It uses a simple array structure so data from other libraries that define their own vector type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

64.3 Procedures and functions

64.3.1 add(Tmatrix2_double,Double):Tmatrix2_double

Synopsis: Add scalar to two-dimensional double precision matrix

Declaration: `operator +(const m: Tmatrix2_double;const x: Double) : Tmatrix2_double`

Visibility: default

Description: This operator allows you to add a scalar value to a matrix. The scalar is added to all elements of the matrix, the result is returned as a new vector.

64.3.2 add(Tmatrix2_double,Tmatrix2_double):Tmatrix2_double

Synopsis: Add two two-dimensional double precision matrices together.

Declaration:
```
operator +(const m1: Tmatrix2_double;const m2: Tmatrix2_double)
                : Tmatrix2_double
```

Visibility: default

Description: This operator allows you to add two two-dimensional double precision matrices together. A new matrix is returned with all elements of the two matrices added together.

64.3.3 add(Tmatrix2_extended,extended):Tmatrix2_extended

Synopsis: Add scalar to two-dimensional extended precision matrix

Declaration:
```
operator +(const m: Tmatrix2_extended;const x: extended)
                : Tmatrix2_extended
```

Visibility: default

Description: This operator allows you to add a scalar value to a matrix. The scalar is added to all elements of the matrix, the result is returned as a new vector.

64.3.4 add(Tmatrix2_extended,Tmatrix2_extended):Tmatrix2_extended

Synopsis: Add two two-dimensional extended precision matrices together.

Declaration:
```
operator +(const m1: Tmatrix2_extended;const m2: Tmatrix2_extended)
                : Tmatrix2_extended
```

Visibility: default

Description: This operator allows you to add two two-dimensional extended precision matrices together. A new matrix is returned with all elements of the two matrices added together.

64.3.5 add(Tmatrix2_single,single):Tmatrix2_single

Synopsis: Add scalar to two-dimensional single precision matrix

Declaration:
```
operator +(const m: Tmatrix2_single;const x: single) : Tmatrix2_single
```

Visibility: default

Description: This operator allows you to add a scalar value to a matrix. The scalar is added to all elements of the matrix, the result is returned as a new vector.

64.3.6 add(Tmatrix2_single,Tmatrix2_single):Tmatrix2_single

Synopsis: Add two two-dimensional single precision matrices together.

Declaration:
```
operator +(const m1: Tmatrix2_single;const m2: Tmatrix2_single)
                : Tmatrix2_single
```

Visibility: default

Description: This operator allows you to add two two-dimensional single precision matrices together. A new matrix is returned with all elements of the two matrices added together.

64.3.7 add(Tmatrix3_double,Double):Tmatrix3_double

Synopsis: Add scalar to three-dimensional double precision matrix

Declaration:
```
operator +(const m: Tmatrix3_double;const x: Double) : Tmatrix3_double
```

Visibility: default

Description: This operator allows you to add a scalar value to a matrix. The scalar is added to all elements of the matrix, the result is returned as a new vector.

64.3.8 add(Tmatrix3_double,Tmatrix3_double):Tmatrix3_double

Synopsis: Add two three-dimensional double precision matrices together.

Declaration:
```
operator +(const m1: Tmatrix3_double;const m2: Tmatrix3_double)
                : Tmatrix3_double
```

Visibility: default

Description: This operator allows you to add two three-dimensional double precision matrices together. A new matrix is returned with all elements of the two matrices added together.

64.3.9 add(Tmatrix3_extended,extended):Tmatrix3_extended

Synopsis: Add scalar to three-dimensional extended precision matrix

Declaration:
```
operator +(const m: Tmatrix3_extended;const x: extended)
                : Tmatrix3_extended
```

Visibility: default

Description: This operator allows you to add a scalar value to a matrix. The scalar is added to all elements of the matrix, the result is returned as a new vector.

64.3.10 add(Tmatrix3_extended,Tmatrix3_extended):Tmatrix3_extended

Synopsis: Add two three-dimensional extended precision matrices together.

Declaration:
```
operator +(const m1: Tmatrix3_extended;const m2: Tmatrix3_extended)
                : Tmatrix3_extended
```

Visibility: default

Description: This operator allows you to add two three-dimensional extended precision matrices together. A new matrix is returned with all elements of the two matrices added together.

64.3.11 add(Tmatrix3_single,single):Tmatrix3_single

Synopsis: Add scalar to three-dimensional single precision matrix

Declaration:
```
operator +(const m: Tmatrix3_single;const x: single) : Tmatrix3_single
```

Visibility: default

Description: This operator allows you to add a scalar value to a matrix. The scalar is added to all elements of the matrix, the result is returned as a new vector.

64.3.12 add(Tmatrix3_single,Tmatrix3_single):Tmatrix3_single

Synopsis: Add two three-dimensional single precision matrices together.

Declaration:
```
operator +(const m1: Tmatrix3_single;const m2: Tmatrix3_single)
                : Tmatrix3_single
```

Visibility: default

Description: This operator allows you to add two three-dimensional single precision matrices together. A new matrix is returned with all elements of the two matrices added together.

64.3.13 add(Tmatrix4_double,Double):Tmatrix4_double

Synopsis: Add scalar to four-dimensional double precision matrix

Declaration:
```
operator +(const m: Tmatrix4_double;const x: Double) : Tmatrix4_double
```

Visibility: default

Description: This operator allows you to add a scalar value to a matrix. The scalar is added to all elements of the matrix, the result is returned as a new vector.

64.3.14 add(Tmatrix4_double,Tmatrix4_double):Tmatrix4_double

Synopsis: Add two four-dimensional double precision matrices together.

Declaration:
```
operator +(const m1: Tmatrix4_double;const m2: Tmatrix4_double)
                : Tmatrix4_double
```

Visibility: default

Description: This operator allows you to add two four-dimensional double precision matrices together. A new matrix is returned with all elements of the two matrices added together.

64.3.15 add(Tmatrix4_extended,extended):Tmatrix4_extended

Synopsis: Add scalar to four-dimensional extended precision matrix

Declaration:
```
operator +(const m: Tmatrix4_extended;const x: extended)
                : Tmatrix4_extended
```

Visibility: default

Description: This operator allows you to add a scalar value to a matrix. The scalar is added to all elements of the matrix, the result is returned as a new vector.

64.3.16 add(Tmatrix4_extended,Tmatrix4_extended):Tmatrix4_extended

Synopsis: Add two four-dimensional extended precision matrices together.

Declaration:
```
operator +(const m1: Tmatrix4_extended;const m2: Tmatrix4_extended)
                : Tmatrix4_extended
```

Visibility: default

Description: This operator allows you to add two four-dimensional extended precision matrices together. A new matrix is returned with all elements of the two matrices added together.

64.3.17 add(Tmatrix4_single,single):Tmatrix4_single

Synopsis: Add scalar to four-dimensional single precision matrix

Declaration:
```
operator +(const m: Tmatrix4_single;const x: single) : Tmatrix4_single
```

Visibility: default

Description: This operator allows you to add a scalar value to a matrix. The scalar is added to all elements of the matrix, the result is returned as a new vector.

64.3.18 add(Tmatrix4_single,Tmatrix4_single):Tmatrix4_single

Synopsis: Add two four-dimensional single precision matrices together.

Declaration:
```
operator +(const m1: Tmatrix4_single;const m2: Tmatrix4_single)
              : Tmatrix4_single
```

Visibility: default

Description: This operator allows you to add two four-dimensional single precision matrices together. A new matrix is returned with all elements of the two matrices added together.

64.3.19 add(Tvector2_double,Double):Tvector2_double

Synopsis: Add scalar to two-dimensional double precision vector

Declaration:
```
operator +(const x: Tvector2_double;y: Double) : Tvector2_double
```

Visibility: default

Description: This operator allows you to add a scalar value to a vector. The scalar is added to all elements of the vector, the result is returned as a new vector.

64.3.20 add(Tvector2_double,Tvector2_double):Tvector2_double

Synopsis: Add two-dimensional double precision vectors together

Declaration:
```
operator +(const x: Tvector2_double;const y: Tvector2_double)
              : Tvector2_double
```

Visibility: default

Description: This operator allows you to add two two-dimensional vectors with double precision together. The result is a new vector which consists of the sums of the individual elements of the two vectors.

64.3.21 add(Tvector2_extended,extended):Tvector2_extended

Synopsis: Add scalar to two-dimensional extended precision vector

Declaration:
```
operator +(const x: Tvector2_extended;y: extended) : Tvector2_extended
```

Visibility: default

Description: This operator allows you to add a scalar value to a vector. The scalar is added to all elements of the vector, the result is returned as a new vector.

64.3.22 add(Tvector2_extended,Tvector2_extended):Tvector2_extended

Synopsis: Add two-dimensional extended precision vectors together

Declaration:
```
operator +(const x: Tvector2_extended;const y: Tvector2_extended)
              : Tvector2_extended
```

Visibility: default

Description: This operator allows you to add two two-dimensional vectors with extended precision together. The result is a new vector which consists of the sums of the individual elements of the two vectors.

64.3.23 add(Tvector2_single,single):Tvector2_single

Synopsis: Add scalar to two-dimensional single precision vector

Declaration:
```
operator +(const x: Tvector2_single;y: single) : Tvector2_single
```

Visibility: default

Description: This operator allows you to add a scalar value to a vector. The scalar is added to all elements of the vector, the result is returned as a new vector.

64.3.24 add(Tvector2_single,Tvector2_single):Tvector2_single

Synopsis: Add two-dimensional single precision vectors together

Declaration:
```
operator +(const x: Tvector2_single;const y: Tvector2_single)
                : Tvector2_single
```

Visibility: default

Description: This operator allows you to add two two-dimensional vectors with single precision together. The result is a new vector which consists of the sums of the individual elements of the two vectors.

64.3.25 add(Tvector3_double,Double):Tvector3_double

Synopsis: Add scalar to three-dimensional double precision vector

Declaration:
```
operator +(const x: Tvector3_double;y: Double) : Tvector3_double
```

Visibility: default

Description: This operator allows you to add a scalar value to a vector. The scalar is added to all elements of the vector, the result is returned as a new vector.

64.3.26 add(Tvector3_double,Tvector3_double):Tvector3_double

Synopsis: Add three-dimensional double precision vectors together

Declaration:
```
operator +(const x: Tvector3_double;const y: Tvector3_double)
                : Tvector3_double
```

Visibility: default

Description: This operator allows you to add two three-dimensional vectors with double precision together. The result is a new vector which consists of the sums of the individual elements of the two vectors.

64.3.27 add(Tvector3_extended,extended):Tvector3_extended

Synopsis: Add scalar to three-dimensional extended precision vector

Declaration:
```
operator +(const x: Tvector3_extended;y: extended) : Tvector3_extended
```

Visibility: default

Description: This operator allows you to add a scalar value to a vector. The scalar is added to all elements of the vector, the result is returned as a new vector.

64.3.28 add(Tvector3_extended,Tvector3_extended):Tvector3_extended

Synopsis: Add three-dimensional extended precision vectors together

Declaration:
```
operator +(const x: Tvector3_extended;const y: Tvector3_extended)
                : Tvector3_extended
```

Visibility: default

Description: This operator allows you to add two three-dimensional vectors with extended precision together. The result is a new vector which consists of the sums of the individual elements of the two vectors.

64.3.29 add(Tvector3_single,single):Tvector3_single

Synopsis: Add scalar to three-dimensional single precision vector

Declaration:
```
operator +(const x: Tvector3_single;y: single) : Tvector3_single
```

Visibility: default

Description: This operator allows you to add a scalar value to a vector. The scalar is added to all elements of the vector, the result is returned as a new vector.

64.3.30 add(Tvector3_single,Tvector3_single):Tvector3_single

Synopsis: Add three-dimensional extended precision vectors together

Declaration:
```
operator +(const x: Tvector3_single;const y: Tvector3_single)
                : Tvector3_single
```

Visibility: default

Description: This operator allows you to add two three-dimensional vectors with single precision together. The result is a new vector which consists of the sums of the individual elements of the two vectors.

64.3.31 add(Tvector4_double,Double):Tvector4_double

Synopsis: Add scalar to four-dimensional double precision vector

Declaration:
```
operator +(const x: Tvector4_double;y: Double) : Tvector4_double
```

Visibility: default

Description: This operator allows you to add a scalar value to a vector. The scalar is added to all elements of the vector, the result is returned as a new vector.

64.3.32 add(Tvector4_double,Tvector4_double):Tvector4_double

Synopsis: Add four-dimensional double precision vectors together

Declaration:
```
operator +(const x: Tvector4_double;const y: Tvector4_double)
                : Tvector4_double
```

Visibility: default

Description: This operator allows you to add two four-dimensional vectors with single precision together. The result is a new vector which consists of the sums of the individual elements of the two vectors.

64.3.33 add(Tvector4_extended,extended):Tvector4_extended

Synopsis: Add scalar to four-dimensional extended precision vector

Declaration:
```
operator +(const x: Tvector4_extended;y: extended) : Tvector4_extended
```

Visibility: default

Description: This operator allows you to add a scalar value to a vector. The scalar is added to all elements of the vector, the result is returned as a new vector.

64.3.34 add(Tvector4_extended,Tvector4_extended):Tvector4_extended

Synopsis: Add four-dimensional extended precision vectors together

Declaration:
```
operator +(const x: Tvector4_extended;const y: Tvector4_extended)
                : Tvector4_extended
```

Visibility: default

Description: This operator allows you to add two two-dimensional vectors with extended precision together. The result is a new vector which consists of the sums of the individual elements of the two vectors.

64.3.35 add(Tvector4_single,single):Tvector4_single

Synopsis: Add scalar to four-dimensional single precision vector

Declaration:
```
operator +(const x: Tvector4_single;y: single) : Tvector4_single
```

Visibility: default

Description: This operator allows you to add a scalar value to a vector. The scalar is added to all elements of the vector, the result is returned as a new vector.

64.3.36 add(Tvector4_single,Tvector4_single):Tvector4_single

Synopsis: Add four-dimensional single precision vectors together

Declaration:
```
operator +(const x: Tvector4_single;const y: Tvector4_single)
                : Tvector4_single
```

Visibility: default

Description: This operator allows you to add two four-dimensional vectors with single precision together. The result is a new vector which consists of the sums of the individual elements of the two vectors.

64.3.37 assign(Tmatrix2_double):Tmatrix2_extended

Synopsis: Allow assignment of two-dimensional double precision matrix to two-dimensional extended precision matrix

Declaration:
```
operator :=(const v: Tmatrix2_double) : Tmatrix2_extended
```

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with double precision values wherever a two-dimensional matrix with extended precision is expected.

64.3.38 assign(Tmatrix2_double):Tmatrix2_single

Synopsis: Allow assignment of two-dimensional double precision matrix to two-dimensional single precision matrix

Declaration: `operator :=(const v: Tmatrix2_double) : Tmatrix2_single`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with double precision values wherever a two-dimensional matrix with single precision is expected. Some accurace is lost because of the conversion.

64.3.39 assign(Tmatrix2_double):Tmatrix3_double

Synopsis: Allow assignment of two-dimensional double precision matrix to three-dimensional double precision matrix

Declaration: `operator :=(const v: Tmatrix2_double) : Tmatrix3_double`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with double precision values wherever a three-dimensional matrix with double precision is expected. The extra fields are set to 0.

64.3.40 assign(Tmatrix2_double):Tmatrix3_extended

Synopsis: Allow assignment of two-dimensional double precision matrix to three-dimensional extended precision matrix

Declaration: `operator :=(const v: Tmatrix2_double) : Tmatrix3_extended`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with double precision values wherever a three-dimensional matrix with extended precision is expected. The extra fields are set to 0.

64.3.41 assign(Tmatrix2_double):Tmatrix3_single

Synopsis: Allow assignment of two-dimensional single precision matrix to three-dimensional single precision matrix

Declaration: `operator :=(const v: Tmatrix2_double) : Tmatrix3_single`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with single precision values wherever a three-dimensional matrix with single precision is expected. The extra fields are set to 0 and some accuracy is lost because of the conversion.

64.3.42 assign(Tmatrix2_double):Tmatrix4_double

Synopsis: Allow assignment of two-dimensional double precision matrix to four-dimensional double precision matrix

Declaration: `operator :=(const v: Tmatrix2_double) : Tmatrix4_double`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with double precision values wherever a four-dimensional matrix with double precision is expected. The extra fields are set to 0.

64.3.43 assign(Tmatrix2_double):Tmatrix4_extended

Synopsis: Allow assignment of two-dimensional double precision matrix to four-dimensional extended precision matrix

Declaration: `operator :=(const v: Tmatrix2_double) : Tmatrix4_extended`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with double precision values wherever a four-dimensional matrix with extended precision is expected. The extra fields are set to 0.

64.3.44 assign(Tmatrix2_double):Tmatrix4_single

Synopsis: Allow assignment of two-dimensional double precision matrix to four-dimensional single precision matrix

Declaration: `operator :=(const v: Tmatrix2_double) : Tmatrix4_single`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with double precision values wherever a four-dimensional matrix with single precision is expected. The extra fields are set to 0 and some precision is lost because of the conversion.

64.3.45 assign(Tmatrix2_extended):Tmatrix2_double

Synopsis: Allow assignment of two-dimensional extended precision matrix to two-dimensional double precision matrix

Declaration: `operator :=(const v: Tmatrix2_extended) : Tmatrix2_double`

Visibility: default

Description: This operator allows you to use a two-dimensional two with extended precision values wherever a two-dimensional matrix with double precision is expected. Some accuracy is lost because of the conversion.

64.3.46 assign(Tmatrix2_extended):Tmatrix2_single

Synopsis: Allow assignment of two-dimensional extended precision matrix to two-dimensional single precision matrix

Declaration: `operator :=(const v: Tmatrix2_extended) : Tmatrix2_single`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with extended precision values wherever a two-dimensional matrix with single precision is expected. Some accuracy is lost because of the conversion.

64.3.47 assign(Tmatrix2_extended):Tmatrix3_double

Synopsis: Allow assignment of two-dimensional extended precision matrix to three-dimensional double precision matrix

Declaration: `operator :=(const v: Tmatrix2_extended) : Tmatrix3_double`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with extended precision values wherever a three-dimensional matrix with double precision is expected. The extra fields are set to 0 and some accuracy is lost because of the conversion.

64.3.48 assign(Tmatrix2_extended):Tmatrix3_extended

Synopsis: Allow assignment of two-dimensional extended precision matrix to three-dimensional extended precision matrix

Declaration: `operator :=(const v: Tmatrix2_extended) : Tmatrix3_extended`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with extended precision values wherever a three-dimensional matrix with extended precision is expected. The extra fields are set to 0.

64.3.49 assign(Tmatrix2_extended):Tmatrix3_single

Synopsis: Allow assignment of two-dimensional extended precision matrix to three-dimensional single precision matrix

Declaration: `operator :=(const v: Tmatrix2_extended) : Tmatrix3_single`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with extended precision values wherever a three-dimensional matrix with single precision is expected. The extra fields are set to 0 and some accuracy is lost because of the conversion.

64.3.50 assign(Tmatrix2_extended):Tmatrix4_double

Synopsis: Allow assignment of two-dimensional extended precision matrix to four-dimensional double precision matrix

Declaration: `operator :=(const v: Tmatrix2_extended) : Tmatrix4_double`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with extended precision values wherever a four-dimensional matrix with double precision is expected. The extra fields are set to 0 and some accuracy is lost because of the conversion.

64.3.51 assign(Tmatrix2_extended):Tmatrix4_extended

Synopsis: Allow assignment of two-dimensional extended precision matrix to four-dimensional extended precision matrix

Declaration: `operator :=(const v: Tmatrix2_extended) : Tmatrix4_extended`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with single precision values wherever a four-dimensional matrix with single precision is expected. The extra fields are set to 0.

64.3.52 assign(Tmatrix2_extended):Tmatrix4_single

Synopsis: Allow assignment of two-dimensional extended precision matrix to four-dimensional single precision matrix

Declaration: `operator :=(const v: Tmatrix2_extended) : Tmatrix4_single`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with extended precision values wherever a four-dimensional matrix with single precision is expected. The extra fields are set to 0 and some precision is lost because of the conversion.

64.3.53 assign(Tmatrix2_single):Tmatrix2_double

Synopsis: Allow assignment of two-dimensional single precision matrix to two-dimensional double precision matrix

Declaration: `operator :=(const v: Tmatrix2_single) : Tmatrix2_double`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with single precision values wherever a two-dimensional matrix with double precision is expected.

64.3.54 assign(Tmatrix2_single):Tmatrix2_extended

Synopsis: Allow assignment of two-dimensional single precision matrix to two-dimensional extended precision matrix

Declaration: `operator :=(const v: Tmatrix2_single) : Tmatrix2_extended`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with single precision values wherever a two-dimensional matrix with extended precision is expected.

64.3.55 assign(Tmatrix2_single):Tmatrix3_double

Synopsis: Allow assignment of two-dimensional single precision matrix to three-dimensional double precision matrix

Declaration: `operator :=(const v: Tmatrix2_single) : Tmatrix3_double`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with single precision values wherever a three-dimensional matrix with double precision is expected. The extra fields are set to 0.

64.3.56 assign(Tmatrix2_single):Tmatrix3_extended

Synopsis: Allow assignment of two-dimensional single precision matrix to three-dimensional extended precision matrix

Declaration: `operator :=(const v: Tmatrix2_single) : Tmatrix3_extended`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with single precision values wherever a three-dimensional matrix with extended precision is expected. The extra fields are set to 0.

64.3.57 assign(Tmatrix2_single):Tmatrix3_single

Synopsis: Allow assignment of two-dimensional single precision matrix to three-dimensional single precision matrix

Declaration: `operator :=(const v: Tmatrix2_single) : Tmatrix3_single`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with single precision values wherever a three-dimensional matrix with single precision is expected. The extra fields are set to 0.

64.3.58 assign(Tmatrix2_single):Tmatrix4_double

Synopsis: Allow assignment of two-dimensional single precision matrix to four-dimensional double precision matrix

Declaration: `operator :=(const v: Tmatrix2_single) : Tmatrix4_double`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with single precision values wherever a four-dimensional matrix with double precision is expected. The extra fields are set to 0.

64.3.59 assign(Tmatrix2_single):Tmatrix4_extended

Synopsis: Allow assignment of two-dimensional single precision matrix to four-dimensional extended precision matrix

Declaration: `operator :=(const v: Tmatrix2_single) : Tmatrix4_extended`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with single precision values wherever a four-dimensional matrix with extended precision is expected. The extra fields are set to 0.

64.3.60 assign(Tmatrix2_single):Tmatrix4_single

Synopsis: Allow assignment of two-dimensional single precision matrix to four-dimensional single precision matrix

Declaration: `operator :=(const v: Tmatrix2_single) : Tmatrix4_single`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with single precision values wherever a four-dimensional matrix with single precision is expected. The extra fields are set to 0.

64.3.61 assign(Tmatrix3_double):Tmatrix2_double

Synopsis: Allow assignment of three-dimensional double precision matrix to two-dimensional double precision matrix

Declaration: `operator :=(const v: Tmatrix3_double) : Tmatrix2_double`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with double precision values wherever a two-dimensional matrix with double precision is expected. The surplus fields are thrown away.

64.3.62 assign(Tmatrix3_double):Tmatrix2_extended

Synopsis: Allow assignment of three-dimensional double precision matrix to two-dimensional extended precision matrix

Declaration: `operator :=(const v: Tmatrix3_double) : Tmatrix2_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with double precision values wherever a two-dimensional matrix with extended precision is expected. The surplus fields are thrown away.

64.3.63 assign(Tmatrix3_double):Tmatrix2_single

Synopsis: Allow assignment of three-dimensional double precision matrix to two-dimensional single precision matrix

Declaration: `operator :=(const v: Tmatrix3_double) : Tmatrix2_single`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with double precision values wherever a two-dimensional matrix with single precision is expected. The surplus fields are thrown away and some accuracy is lost because of the conversion.

64.3.64 assign(Tmatrix3_double):Tmatrix3_extended

Synopsis: Allow assignment of three-dimensional double precision matrix to three-dimensional extended precision matrix

Declaration: `operator :=(const v: Tmatrix3_double) : Tmatrix3_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with double precision values wherever a three-dimensional matrix with extended precision is expected.

64.3.65 assign(Tmatrix3_double):Tmatrix3_single

Synopsis: Allow assignment of three-dimensional double precision matrix to three-dimensional single precision matrix

Declaration: `operator :=(const v: Tmatrix3_double) : Tmatrix3_single`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with double precision values wherever a three-dimensional matrix with single precision is expected. Some precision is lost because of the conversion.

64.3.66 assign(Tmatrix3_double):Tmatrix4_double

Synopsis: Allow assignment of three-dimensional double precision matrix to four-dimensional double precision matrix

Declaration: `operator :=(const v: Tmatrix3_double) : Tmatrix4_double`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with double precision values wherever a four-dimensional matrix with double precision is expected.

64.3.67 assign(Tmatrix3_double):Tmatrix4_extended

Synopsis: Allow assignment of three-dimensional double precision matrix to four-dimensional extended precision matrix

Declaration: `operator :=(const v: Tmatrix3_double) : Tmatrix4_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with double precision values wherever a four-dimensional matrix with extended precision is expected.

64.3.68 assign(Tmatrix3_double):Tmatrix4_single

Synopsis: Allow assignment of three-dimensional double precision matrix to four-dimensional single precision matrix

Declaration: `operator :=(const v: Tmatrix3_double) : Tmatrix4_single`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with double precision values wherever a four-dimensional matrix with single precision is expected. Some precision is lost because of the conversion.

64.3.69 assign(Tmatrix3_extended):Tmatrix2_double

Synopsis: Allow assignment of three-dimensional extended precision matrix to two-dimensional double precision matrix

Declaration: `operator :=(const v: Tmatrix3_extended) : Tmatrix2_double`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with extended precision values wherever a two-dimensional matrix with double precision is expected. The surplus fields are thrown away and some accuracy is lost becuase of the conversion.

64.3.70 assign(Tmatrix3_extended):Tmatrix2_extended

Synopsis: Allow assignment of three-dimensional extended precision matrix to two-dimensional extended precision matrix

Declaration: `operator :=(const v: Tmatrix3_extended) : Tmatrix2_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with extended precision values wherever a two-dimensional matrix with extended precision is expected. The surplus fields are thrown away.

64.3.71 assign(Tmatrix3_extended):Tmatrix2_single

Synopsis: Allow assignment of three-dimensional extended precision matrix to two-dimensional single precision matrix

Declaration: `operator :=(const v: Tmatrix3_extended) : Tmatrix2_single`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with extended precision values wherever a two-dimensional matrix with single precision is expected. The surplus fields are thrown away and some precision is lost because of the conversion.

64.3.72 assign(Tmatrix3_extended):Tmatrix3_double

Synopsis: Allow assignment of three-dimensional extended precision matrix to three-dimensional double precision matrix

Declaration: `operator :=(const v: Tmatrix3_extended) : Tmatrix3_double`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with extended precision values wherever a three-dimensional matrix with double precision is expected. Some precision is lost because of the conversion.

64.3.73 assign(Tmatrix3_extended):Tmatrix3_single

Synopsis: Allow assignment of three-dimensional extended precision matrix to three-dimensional single precision matrix

Declaration: `operator :=(const v: Tmatrix3_extended) : Tmatrix3_single`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with extended precision values wherever a three-dimensional matrix with single precision is expected. Some precision is lost because of the conversion.

64.3.74 assign(Tmatrix3_extended):Tmatrix4_double

Synopsis: Allow assignment of three-dimensional extended precision matrix to four-dimensional double precision matrix

Declaration: `operator :=(const v: Tmatrix3_extended) : Tmatrix4_double`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with extended precision values wherever a four-dimensional matrix with double precision is expected. Some precision is lost because of the conversion.

64.3.75 assign(Tmatrix3_extended):Tmatrix4_extended

Synopsis: Allow assignment of three-dimensional extended precision matrix to four-dimensional extended precision matrix

Declaration: `operator :=(const v: Tmatrix3_extended) : Tmatrix4_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with extended precision values wherever a four-dimensional matrix with extended precision is expected.

64.3.76 assign(Tmatrix3_extended):Tmatrix4_single

Synopsis: Allow assignment of three-dimensional extended precision matrix to four-dimensional single precision matrix

Declaration: `operator :=(const v: Tmatrix3_extended) : Tmatrix4_single`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with extended precision values wherever a four-dimensional matrix with single precision is expected. Some precision is lost because of the conversion.

64.3.77 assign(Tmatrix3_single):Tmatrix2_double

Synopsis: Allow assignment of three-dimensional single precision matrix to two-dimensional double precision matrix

Declaration: `operator :=(const v: Tmatrix3_single) : Tmatrix2_double`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with single precision values wherever a two-dimensional matrix with double precision is expected. The surplus fields are thrown away.

64.3.78 assign(Tmatrix3_single):Tmatrix2_extended

Synopsis: Allow assignment of three-dimensional single precision matrix to two-dimensional extended precision matrix

Declaration: `operator :=(const v: Tmatrix3_single) : Tmatrix2_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with single precision values wherever a two-dimensional matrix with extended precision is expected. The surplus fields are thrown away.

64.3.79 assign(Tmatrix3_single):Tmatrix2_single

Synopsis: Allow assignment of three-dimensional single precision matrix to two-dimensional single precision matrix

Declaration: `operator :=(const v: Tmatrix3_single) : Tmatrix2_single`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with single precision values wherever a two-dimensional matrix with single precision is expected. The surplus fields are thrown away.

64.3.80 assign(Tmatrix3_single):Tmatrix3_double

Synopsis: Allow assignment of three-dimensional single precision matrix to three-dimensional double precision matrix

Declaration: `operator :=(const v: Tmatrix3_single) : Tmatrix3_double`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with single precision values wherever a three-dimensional matrix with double precision is expected.

64.3.81 assign(Tmatrix3_single):Tmatrix3_extended

Synopsis: Allow assignment of three-dimensional single precision matrix to three-dimensional extended precision matrix

Declaration: `operator :=(const v: Tmatrix3_single) : Tmatrix3_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with single precision values wherever a three-dimensional matrix with extended precision is expected.

64.3.82 assign(Tmatrix3_single):Tmatrix4_double

Synopsis: Allow assignment of three-dimensional single precision matrix to four-dimensional double precision matrix

Declaration: `operator :=(const v: Tmatrix3_single) : Tmatrix4_double`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with single precision values wherever a four-dimensional matrix with double precision is expected.

64.3.83 assign(Tmatrix3_single):Tmatrix4_extended

Synopsis: Allow assignment of three-dimensional single precision matrix to four-dimensional extended precision matrix

Declaration: `operator :=(const v: Tmatrix3_single) : Tmatrix4_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with single precision values wherever a four-dimensional matrix with extended precision is expected.

64.3.84 assign(Tmatrix3_single):Tmatrix4_single

Synopsis: Allow assignment of three-dimensional single precision matrix to four-dimensional single precision matrix

Declaration: `operator :=(const v: Tmatrix3_single) : Tmatrix4_single`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with single precision values wherever a four-dimensional matrix with single precision is expected.

64.3.85 assign(Tmatrix4_double):Tmatrix2_double

Synopsis: Allow assignment of four-dimensional double precision matrix to two-dimensional double precision matrix

Declaration: `operator :=(const v: Tmatrix4_double) : Tmatrix2_double`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with double precision values wherever a two-dimensional matrix with double precision is expected. The surplus fields are thrown away.

64.3.86 assign(Tmatrix4_double):Tmatrix2_extended

Synopsis: Allow assignment of four-dimensional double precision matrix to two-dimensional extended precision matrix

Declaration: `operator :=(const v: Tmatrix4_double) : Tmatrix2_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with double precision values wherever a two-dimensional matrix with extended precision is expected. The surplus fields are thrown away.

64.3.87 assign(Tmatrix4_double):Tmatrix2_single

Synopsis: Allow assignment of four-dimensional double precision matrix to two-dimensional single precision matrix

Declaration: `operator :=(const v: Tmatrix4_double) : Tmatrix2_single`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with double precision values wherever a two-dimensional matrix with single precision is expected. The surplus fields are thrown away and some precision is lost in the conversion.

64.3.88 assign(Tmatrix4_double):Tmatrix3_double

Synopsis: Allow assignment of four-dimensional double precision matrix to three-dimensional double precision matrix

Declaration: `operator :=(const v: Tmatrix4_double) : Tmatrix3_double`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with double precision values wherever a three-dimensional matrix with double precision is expected. The surplus fields are thrown away.

64.3.89 assign(Tmatrix4_double):Tmatrix3_extended

Synopsis: Allow assignment of four-dimensional double precision matrix to three-dimensional extended precision matrix

Declaration: `operator :=(const v: Tmatrix4_double) : Tmatrix3_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with double precision values wherever a three-dimensional matrix with extended precision is expected. The surplus fields are thrown away.

64.3.90 assign(Tmatrix4_double):Tmatrix3_single

Synopsis: Allow assignment of four-dimensional double precision matrix to three-dimensional single precision matrix

Declaration: `operator :=(const v: Tmatrix4_double) : Tmatrix3_single`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with double precision values wherever a three-dimensional matrix with single precision is expected. The surplus fields are thrown away and some precision is lost because of the conversion.

64.3.91 assign(Tmatrix4_double):Tmatrix4_extended

Synopsis: Allow assignment of four-dimensional double precision matrix to four-dimensional extended precision matrix

Declaration: `operator :=(const v: Tmatrix4_double) : Tmatrix4_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with double precision values wherever a four-dimensional matrix with extended precision is expected.

64.3.92 assign(Tmatrix4_double):Tmatrix4_single

Synopsis: Allow assignment of four-dimensional single precision matrix to four-dimensional single precision matrix

Declaration: `operator :=(const v: Tmatrix4_double) : Tmatrix4_single`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with double precision values wherever a four-dimensional matrix with single precision is expected. Some precision is lost because of the conversion.

64.3.93 assign(Tmatrix4_extended):Tmatrix2_double

Synopsis: Allow assignment of four-dimensional extended precision matrix to two-dimensional double precision matrix

Declaration: `operator :=(const v: Tmatrix4_extended) : Tmatrix2_double`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with extended precision values wherever a two-dimensional matrix with double precision is expected. The surplus fields are thrown away and some precision is lost in the conversion.

64.3.94 assign(Tmatrix4_extended):Tmatrix2_extended

Synopsis: Allow assignment of four-dimensional extended precision matrix to two-dimensional extended precision matrix

Declaration: `operator :=(const v: Tmatrix4_extended) : Tmatrix2_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with single precision values wherever a two-dimensional matrix with single precision is expected. The surplus fields are thrown away.

64.3.95 assign(Tmatrix4_extended):Tmatrix2_single

Synopsis: Allow assignment of four-dimensional extended precision matrix to two-dimensional single precision matrix

Declaration: `operator :=(const v: Tmatrix4_extended) : Tmatrix2_single`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with extended precision values wherever a two-dimensional matrix with single precision is expected. The surplus fields are thrown away and some precision is lost in the conversion.

64.3.96 assign(Tmatrix4_extended):Tmatrix3_double

Synopsis: Allow assignment of four-dimensional extended precision matrix to three-dimensional double precision matrix

Declaration: `operator :=(const v: Tmatrix4_extended) : Tmatrix3_double`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with extended precision values wherever a three-dimensional matrix with double precision is expected. The surplus fields are thrown away.

64.3.97 assign(Tmatrix4_extended):Tmatrix3_extended

Synopsis: Allow assignment of four-dimensional extended precision matrix to three-dimensional extended precision matrix

Declaration: `operator :=(const v: Tmatrix4_extended) : Tmatrix3_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with extended precision values wherever a three-dimensional matrix with double precision is expected. The surplus fields are thrown away.

64.3.98 assign(Tmatrix4_extended):Tmatrix3_single

Synopsis: Allow assignment of four-dimensional extended precision matrix to three-dimensional single precision matrix

Declaration: `operator :=(const v: Tmatrix4_extended) : Tmatrix3_single`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with extended precision values wherever a three-dimensional matrix with single precision is expected. The surplus fields are thrown away and some precision is lost because of the conversion.

64.3.99 assign(Tmatrix4_extended):Tmatrix4_double

Synopsis: Allow assignment of four-dimensional extended precision matrix to four-dimensional double precision matrix

Declaration: `operator :=(const v: Tmatrix4_extended) : Tmatrix4_double`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with extended precision values wherever a four-dimensional matrix with double precision is expected.

64.3.100 assign(Tmatrix4_extended):Tmatrix4_single

Synopsis: Allow assignment of four-dimensional extended precision matrix to four-dimensional single precision matrix

Declaration: `operator :=(const v: Tmatrix4_extended) : Tmatrix4_single`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with extended precision values wherever a four-dimensional matrix with single precision is expected. Some precision is lost because of the conversion.

64.3.101 assign(Tmatrix4_single):Tmatrix2_double

Synopsis: Allow assignment of four-dimensional single precision matrix to two-dimensional double precision matrix

Declaration: `operator :=(const v: Tmatrix4_single) : Tmatrix2_double`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with single precision values wherever a two-dimensional matrix with double precision is expected. The surplus fields are thrown away.

64.3.102 assign(Tmatrix4_single):Tmatrix2_extended

Synopsis: Allow assignment of four-dimensional single precision matrix to two-dimensional extended precision matrix

Declaration: `operator :=(const v: Tmatrix4_single) : Tmatrix2_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with single precision values wherever a two-dimensional matrix with extended precision is expected. The surplus fields are thrown away.

64.3.103 assign(Tmatrix4_single):Tmatrix2_single

Synopsis: Allow assignment of four-dimensional single precision matrix to two-dimensional single precision matrix

Declaration: `operator :=(const v: Tmatrix4_single) : Tmatrix2_single`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with single precision values wherever a two-dimensional matrix with single precision is expected. The surplus fields are thrown away.

64.3.104 assign(Tmatrix4_single):Tmatrix3_double

Synopsis: Allow assignment of four-dimensional single precision matrix to three-dimensional double precision matrix

Declaration: `operator :=(const v: Tmatrix4_single) : Tmatrix3_double`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with single precision values wherever a three-dimensional matrix with double precision is expected. The surplus fields are thrown away.

64.3.105 assign(Tmatrix4_single):Tmatrix3_extended

Synopsis: Allow assignment of four-dimensional single precision matrix to three-dimensional extended precision matrix

Declaration: `operator :=(const v: Tmatrix4_single) : Tmatrix3_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with single precision values wherever a three-dimensional matrix with extended precision is expected. The surplus fields are thrown away.

64.3.106 assign(Tmatrix4_single):Tmatrix3_single

Synopsis: Allow assignment of four-dimensional single precision matrix to three-dimensional single precision matrix

Declaration: `operator :=(const v: Tmatrix4_single) : Tmatrix3_single`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with single precision values wherever a three-dimensional matrix with single precision is expected. The surplus fields are thrown away.

64.3.107 assign(Tmatrix4_single):Tmatrix4_double

Synopsis: Allow assignment of four-dimensional single precision matrix to four-dimensional double precision matrix

Declaration: `operator :=(const v: Tmatrix4_single) : Tmatrix4_double`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with single precision values wherever a four-dimensional matrix with double precision is expected.

64.3.108 assign(Tmatrix4_single):Tmatrix4_extended

Synopsis: Allow assignment of four-dimensional single precision matrix to four-dimensional extended precision matrix

Declaration: `operator :=(const v: Tmatrix4_single) : Tmatrix4_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with single precision values wherever a four-dimensional matrix with extended precision is expected.

64.3.109 assign(Tvector2_double):Tvector2_extended

Synopsis: Allow assignment of double precision vector to extended precision vector

Declaration: `operator :=(const v: Tvector2_double) : Tvector2_extended`

Visibility: default

Description: This operator allows you to use a vector with double precision values wherever an extended precision vector is expected.

64.3.110 assign(Tvector2_double):Tvector2_single

Synopsis: Allow assignment of double precision vector to single precision vector

Declaration: `operator :=(const v: Tvector2_double) : Tvector2_single`

Visibility: default

Description: This operator allows you to use a vector with double precision values wherever a single precision vector is expected, at the cost of loosing some precision.

64.3.111 assign(Tvector2_double):Tvector3_double

Synopsis: Allow assignment of two-dimensional double precision vector to three-dimensional double precision vector

Declaration: `operator :=(const v: Tvector2_double) : Tvector3_double`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with double precision values wherever a three-dimensional vector with double precision is expected. The third dimension is set to 0.0.

64.3.112 assign(Tvector2_double):Tvector3_extended

Synopsis: Allow assignment of two-dimensional double precision vector to three-dimensional extended precision vector

Declaration: `operator :=(const v: Tvector2_double) : Tvector3_extended`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with double precision values wherever a three-dimensional vector with extended precision is expected. The third dimension is set to 0.0.

64.3.113 assign(Tvector2_double):Tvector3_single

Synopsis: Allow assignment of two-dimensional double precision vector to three-dimensional single precision vector

Declaration: `operator :=(const v: Tvector2_double) : Tvector3_single`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with double precision values wherever a three-dimensional vector with single precision is expected. Some accuracy is lost because of the conversion and the third dimension is set to 0.0.

64.3.114 assign(Tvector2_double):Tvector4_double

Synopsis: Allow assignment of two-dimensional double precision vector to four-dimensional double precision vector

Declaration: `operator :=(const v: Tvector2_double) : Tvector4_double`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with double precision values wherever a four-dimensional vector with double precision is expected. The third and fourth dimensions are set to 0.0.

64.3.115 assign(Tvector2_double):Tvector4_extended

Synopsis: Allow assignment of two-dimensional double precision vector to four-dimensional extended precision vector

Declaration: `operator :=(const v: Tvector2_double) : Tvector4_extended`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with single precision values wherever a four-dimensional vector with extended precision is expected. The third and fourth dimensions are set to 0.0.

64.3.116 assign(Tvector2_double):Tvector4_single

Synopsis: Allow assignment of two-dimensional double precision vector to four-dimensional single precision vector

Declaration: `operator :=(const v: Tvector2_double) : Tvector4_single`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with double precision values wherever a four-dimensional vector with single precision is expected. Some accurace is lost because of the conversion and the third and fourth dimensions are set to 0.0.

64.3.117 assign(Tvector2_extended):Tvector2_double

Synopsis: Allow assignment of extended precision vector to double precision vector

Declaration: `operator :=(const v: Tvector2_extended) : Tvector2_double`

Visibility: default

Description: This operator allows you to use a vector with extended precision values wherever a double precision vector is expected, at the cost of loosing some precision.

64.3.118 assign(Tvector2_extended):Tvector2_single

Synopsis: Allow assignment of extended precision vector to single precision vector

Declaration: `operator :=(const v: Tvector2_extended) : Tvector2_single`

Visibility: default

Description: This operator allows you to use a vector with extended precision values wherever a single precision vector is expected, at the cost of loosing some precision.

64.3.119 assign(Tvector2_extended):Tvector3_double

Synopsis: Allow assignment of two-dimensional extended precision vector to three-dimensional double precision vector

Declaration: `operator :=(const v: Tvector2_extended) : Tvector3_double`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with extended precision values wherever a three-dimensional vector with double precision is expected. Some accuracy is lost because of the conversion and the third dimension is set to 0.0.

64.3.120 assign(Tvector2_extended):Tvector3_extended

Synopsis: Allow assignment of two-dimensional extened precision vector to three-dimensional extended precision vector

Declaration: `operator :=(const v: Tvector2_extended) : Tvector3_extended`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with extended precision values wherever a three-dimensional vector with extended precision is expected. The third dimension is set to 0.0.

64.3.121 assign(Tvector2_extended):Tvector3_single

Synopsis: Allow assignment of two-dimensional extended precision vector to three-dimensional single precision vector

Declaration: `operator :=(const v: Tvector2_extended) : Tvector3_single`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with extended precision values wherever a three-dimensional vector with single precision is expected. Some accuracy is lost because of the conversion and the third dimension is set to 0.0.

64.3.122 assign(Tvector2_extended):Tvector4_double

Synopsis: Allow assignment of two-dimensional extended precision vector to four-dimensional double precision vector

Declaration: `operator :=(const v: Tvector2_extended) : Tvector4_double`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with extended precision values wherever a four-dimensional vector with double precision is expected. Some accuracy is lost because of the conversion and the third and fourth dimensions are set to 0.0.

64.3.123 assign(Tvector2_extended):Tvector4_extended

Synopsis: Allow assignment of two-dimensiona extended precision vector to four-dimensional extended precision vector

Declaration: `operator :=(const v: Tvector2_extended) : Tvector4_extended`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with extended precision values wherever a four-dimensional vector with extended precision is expected. The third and fourth dimensions are set to 0.0.

64.3.124 assign(Tvector2_extended):Tvector4_single

Synopsis: Allow assignment of two-dimensional extended precision vector to four-dimensional single precision vector

Declaration: `operator :=(const v: Tvector2_extended) : Tvector4_single`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with extended precision values wherever a four-dimensional vector with single precision is expected. Some accuracy is lost because of the conversion and the third and fourth dimensions are set to 0.0.

64.3.125 assign(Tvector2_single):Tvector2_double

Synopsis: Allow assignment of single precision vector to double precision vector

Declaration: `operator :=(const v: Tvector2_single) : Tvector2_double`

Visibility: default

Description: This operator allows you to use a vector with single precision values wherever a double precision vector is expected.

64.3.126 assign(Tvector2_single):Tvector2_extended

Synopsis: Allow assignment of single precision vector to extended precision vector

Declaration: `operator :=(const v: Tvector2_single) : Tvector2_extended`

Visibility: default

Description: This operator allows you to use a vector with single precision values wherever an extended precision vector is expected.

64.3.127 assign(Tvector2_single):Tvector3_double

Synopsis: Allow assignment of two-dimensional single precision vector to three-dimensional double precision vector

Declaration: `operator :=(const v: Tvector2_single) : Tvector3_double`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with single precision values wherever a three-dimensional vector with double precision is expected. The third dimension is set to 0.0.

64.3.128 assign(Tvector2_single):Tvector3_extended

Synopsis: Allow assignment of two-dimensional single precision vector to three-dimensional extended precision vector

Declaration: `operator :=(const v: Tvector2_single) : Tvector3_extended`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with single precision values wherever a three-dimensional vector with extended precision is expected. The third dimension is set to 0.0.

64.3.129 assign(Tvector2_single):Tvector3_single

Synopsis: Allow assignment of two-dimensional single precision vector to three-dimensional single precision vector

Declaration: `operator :=(const v: Tvector2_single) : Tvector3_single`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with single precision values wherever a three-dimensional vector with single precision is expected. The third dimension is set to 0.0.

64.3.130 assign(Tvector2_single):Tvector4_double

Synopsis: Allow assignment of two-dimensional single precision vector to four-dimensional double precision vector

Declaration: `operator :=(const v: Tvector2_single) : Tvector4_double`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with single precision values wherever a four-dimensional vector with double precision is expected. The third and fourth dimensions are set to 0.0.

64.3.131 assign(Tvector2_single):Tvector4_extended

Synopsis: Allow assignment of two-dimensional single precision vector to four-dimensional extended precision vector

Declaration: `operator :=(const v: Tvector2_single) : Tvector4_extended`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with single precision values wherever a four-dimensional vector with extended precision is expected. The third and fourth dimensions are set to 0.0.

64.3.132 assign(Tvector2_single):Tvector4_single

Synopsis: Allow assignment of two-dimensional single precision vector to four-dimensional single precision vector

Declaration: `operator :=(const v: Tvector2_single) : Tvector4_single`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with single precision values wherever a four-dimensional vector with single precision is expected. The third and fourth dimensions are set to 0.0.

64.3.133 assign(Tvector3_double):Tvector2_double

Synopsis: Allow assignment of three-dimensional double precision vector to two-dimensional double precision vector

Declaration: `operator :=(const v: Tvector3_double) : Tvector2_double`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with double precision values wherever a two-dimensional vector with double precision is expected. The third dimension is thrown away.

64.3.134 assign(Tvector3_double):Tvector2_extended

Synopsis: Allow assignment of three-dimensional double precision vector to two-dimensional extended precision vector

Declaration: `operator :=(const v: Tvector3_double) : Tvector2_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with double precision values wherever a two-dimensional vector with extended precision is expected. The third dimension is thrown away.

64.3.135 assign(Tvector3_double):Tvector2_single

Synopsis: Allow assignment of three-dimensional double precision vector to two-dimensional single precision vector

Declaration: `operator :=(const v: Tvector3_double) : Tvector2_single`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with double precision values wherever a two-dimensional vector with single precision is expected. The third dimension is thrown away and some precision is lost because of the conversion.

64.3.136 assign(Tvector3_double):Tvector3_extended

Synopsis: Allow assignment of three-dimensional double precision vector to three-dimensional extended precision vector

Declaration: `operator :=(const v: Tvector3_double) : Tvector3_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with double precision values wherever a three-dimensional vector with extended precision is expected.

64.3.137 assign(Tvector3_double):Tvector3_single

Synopsis: Allow assignment of three-dimensional double precision vector to three-dimensional single precision vector

Declaration: `operator :=(const v: Tvector3_double) : Tvector3_single`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with double precision values wherever a three-dimensional vector with single precision is expected. Some precision is lost because of the conversion.

64.3.138 assign(Tvector3_double):Tvector4_double

Synopsis: Allow assignment of three-dimensional double precision vector to four-dimensional double precision vector

Declaration: `operator :=(const v: Tvector3_double) : Tvector4_double`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with double precision values wherever a four-dimensional vector with double precision is expected. The fourth dimension is set to 0.

64.3.139 assign(Tvector3_double):Tvector4_extended

Synopsis: Allow assignment of three-dimensional double precision vector to four-dimensional extended precision vector

Declaration: `operator :=(const v: Tvector3_double) : Tvector4_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with double precision values wherever a four-dimensional vector with extended precision is expected. The fourth dimension is set to 0.

64.3.140 assign(Tvector3_double):Tvector4_single

Synopsis: Allow assignment of three-dimensional double precision vector to four-dimensional single precision vector

Declaration: `operator :=(const v: Tvector3_double) : Tvector4_single`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with single precision values wherever a four-dimensional vector with double precision is expected. The fourth dimension is set to 0 and some precision is lost because of the conversion.

64.3.141 assign(Tvector3_extended):Tvector2_double

Synopsis: Allow assignment of three-dimensional extended precision vector to two-dimensional double precision vector

Declaration: `operator :=(const v: Tvector3_extended) : Tvector2_double`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with extended precision values wherever a two-dimensional vector with double precision is expected. The third dimension is thrown away and some precision is lost because of the conversion.

64.3.142 assign(Tvector3_extended):Tvector2_extended

Synopsis: Allow assignment of three-dimensional extended precision vector to two-dimensional extended precision vector

Declaration: `operator :=(const v: Tvector3_extended) : Tvector2_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with extended precision values wherever a two-dimensional vector with extended precision is expected. The third dimension is thrown away.

64.3.143 assign(Tvector3_extended):Tvector2_single

Synopsis: Allow assignment of three-dimensional extended precision vector to two-dimensional single precision vector

Declaration: `operator :=(const v: Tvector3_extended) : Tvector2_single`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with extended precision values wherever a two-dimensional vector with single precision is expected. The third dimension is thrown away and some precision is lost because of the conversion.

64.3.144 assign(Tvector3_extended):Tvector3_double

Synopsis: Allow assignment of three-dimensional extended precision vector to three-dimensional double precision vector

Declaration: `operator :=(const v: Tvector3_extended) : Tvector3_double`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with extended precision values wherever a three-dimensional vector with double precision is expected. Some precision is lost because of the conversion.

64.3.145 assign(Tvector3_extended):Tvector3_single

Synopsis: Allow assignment of three-dimensional single precision vector to three-dimensional double precision vector

Declaration: `operator :=(const v: Tvector3_extended) : Tvector3_single`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with single precision values wherever a three-dimensional vector with double precision is expected. Some precision is lost because of the conversion.

64.3.146 assign(Tvector3_extended):Tvector4_double

Synopsis: Allow assignment of three-dimensional extended precision vector to four-dimensional double precision vector

Declaration: `operator :=(const v: Tvector3_extended) : Tvector4_double`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with extended precision values wherever a four-dimensional vector with double precision is expected. The fourth dimension is set to 0 and some accuracy is lost because of the conversion.

64.3.147 assign(Tvector3_extended):Tvector4_extended

Synopsis: Allow assignment of three-dimensional extended precision vector to four-dimensional extended precision vector

Declaration: `operator :=(const v: Tvector3_extended) : Tvector4_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with extended precision values wherever a four-dimensional vector with extended precision is expected. The fourth dimension is set to 0.

64.3.148 assign(Tvector3_extended):Tvector4_single

Synopsis: Allow assignment of three-dimensional extended precision vector to four-dimensional single precision vector

Declaration: `operator :=(const v: Tvector3_extended) : Tvector4_single`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with extended precision values wherever a four-dimensional vector with single precision is expected. The fourth dimension is set to 0 and some accuracy is lost because of the conversion.

64.3.149 assign(Tvector3_single):Tvector2_double

Synopsis: Allow assignment of three-dimensional single precision vector to two-dimensional double precision vector

Declaration: `operator :=(const v: Tvector3_single) : Tvector2_double`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with single precision values wherever a two-dimensional vector with double precision is expected. The third dimension is thrown away.

64.3.150 assign(Tvector3_single):Tvector2_extended

Synopsis: Allow assignment of three-dimensional single precision vector to two-dimensional extended precision vector

Declaration: `operator :=(const v: Tvector3_single) : Tvector2_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with single precision values wherever a two-dimensional vector with extended precision is expected. The third dimension is thrown away.

64.3.151 assign(Tvector3_single):Tvector2_single

Synopsis: Allow assignment of three-dimensional single precision vector to two-dimensional single precision vector

Declaration: `operator :=(const v: Tvector3_single) : Tvector2_single`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with single precision values wherever a two-dimensional vector with single precision is expected. The third dimension is thrown away.

64.3.152 assign(Tvector3_single):Tvector3_double

Synopsis: Allow assignment of three-dimensional single precision vector to three-dimensional double precision vector

Declaration: `operator :=(const v: Tvector3_single) : Tvector3_double`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with single precision values wherever a three-dimensional vector with double precision is expected.

64.3.153 assign(Tvector3_single):Tvector3_extended

Synopsis: Allow assignment of three-dimensional single precision vector to three-dimensional extended precision vector

Declaration: `operator :=(const v: Tvector3_single) : Tvector3_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with single precision values wherever a three-dimensional vector with extended precision is expected.

64.3.154 assign(Tvector3_single):Tvector4_double

Synopsis: Allow assignment of three-dimensional single precision vector to four-dimensional double precision vector

Declaration: `operator :=(const v: Tvector3_single) : Tvector4_double`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with single precision values wherever a four-dimensional vector with double precision is expected. The fourth dimension is set to 0.

64.3.155 assign(Tvector3_single):Tvector4_extended

Synopsis: Allow assignment of three-dimensional single precision vector to four-dimensional extended precision vector

Declaration: `operator :=(const v: Tvector3_single) : Tvector4_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with single precision values wherever a four-dimensional vector with extended precision is expected. The fourth dimension is set to 0.

64.3.156 assign(Tvector3_single):Tvector4_single

Synopsis: Allow assignment of three-dimensional single precision vector to four-dimensional single precision vector

Declaration: `operator :=(const v: Tvector3_single) : Tvector4_single`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with single precision values wherever a four-dimensional vector with single precision is expected. The fourth dimension is set to 0.

64.3.157 assign(Tvector4_double):Tvector2_double

Synopsis: Allow assignment of four-dimensional double precision vector to two-dimensional double precision vector

Declaration: `operator :=(const v: Tvector4_double) : Tvector2_double`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with double precision values wherever a two-dimensional vector with double precision is expected. The third and fourth dimensions are thrown away.

64.3.158 assign(Tvector4_double):Tvector2_extended

Synopsis: Allow assignment of four-dimensional double precision vector to two-dimensional extended precision vector

Declaration: `operator :=(const v: Tvector4_double) : Tvector2_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with double precision values wherever a two-dimensional vector with extended precision is expected. The third and fourth dimensions are thrown away.

64.3.159 assign(Tvector4_double):Tvector2_single

Synopsis: Allow assignment of four-dimensional double precision vector to two-dimensional single precision vector

Declaration: `operator :=(const v: Tvector4_double) : Tvector2_single`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with double precision values wherever a two-dimensional vector with single precision is expected. The third and fourth dimensions are thrown away and some accuracy is lost because of the conversion.

64.3.160 assign(Tvector4_double):Tvector3_double

Synopsis: Allow assignment of four-dimensional double precision vector to three-dimensional double precision vector

Declaration: `operator :=(const v: Tvector4_double) : Tvector3_double`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with double precision values wherever a three-dimensional vector with double precision is expected. The fourth dimension is thrown away.

64.3.161 assign(Tvector4_double):Tvector3_extended

Synopsis: Allow assignment of four-dimensional double precision vector to three-dimensional extended precision vector

Declaration: `operator :=(const v: Tvector4_double) : Tvector3_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with double precision values wherever a three-dimensional vector with extended precision is expected. The fourth dimension is thrown away.

64.3.162 assign(Tvector4_double):Tvector3_single

Synopsis: Allow assignment of four-dimensional double precision vector to three-dimensional single precision vector

Declaration: `operator :=(const v: Tvector4_double) : Tvector3_single`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with double precision values wherever a three-dimensional vector with single precision is expected. The fourth dimension is thrown away and some accuracy is lost because of the conversion.

64.3.163 assign(Tvector4_double):Tvector4_extended

Synopsis: Allow assignment of four-dimensional single precision vector to four-dimensional extended precision vector

Declaration: `operator :=(const v: Tvector4_double) : Tvector4_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with double precision values wherever a four-dimensional vector with extended precision is expected.

64.3.164 assign(Tvector4_double):Tvector4_single

Synopsis: Allow assignment of four-dimensional double precision vector to four-dimensional single precision vector

Declaration: `operator :=(const v: Tvector4_double) : Tvector4_single`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with double precision values wherever a four-dimensional vector with single precision is expected. Some accuracy is lost because of the conversion.

64.3.165 assign(Tvector4_extended):Tvector2_double

Synopsis: Allow assignment of four-dimensional extended precision vector to two-dimensional double precision vector

Declaration: `operator :=(const v: Tvector4_extended) : Tvector2_double`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with extended precision values wherever a two-dimensional vector with double precision is expected. The third and fourth dimensions are thrown away and some accuracy is lost because of the conversion.

64.3.166 assign(Tvector4_extended):Tvector2_extended

Synopsis: Allow assignment of four-dimensional extended precision vector to two-dimensional extended precision vector

Declaration: `operator :=(const v: Tvector4_extended) : Tvector2_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with extended precision values wherever a two-dimensional vector with extended precision is expected. The third and fourth dimensions are thrown away.

64.3.167 assign(Tvector4_extended):Tvector2_single

Synopsis: Allow assignment of four-dimensional extended precision vector to two-dimensional single precision vector

Declaration: `operator :=(const v: Tvector4_extended) : Tvector2_single`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with extended precision values wherever a two-dimensional vector with single precision is expected. The third and fourth dimensions are thrown away and some accuracy is los because of the conversion.

64.3.168 assign(Tvector4_extended):Tvector3_double

Synopsis: Allow assignment of four-dimensional extended precision vector to three-dimensional double precision vector

Declaration: `operator :=(const v: Tvector4_extended) : Tvector3_double`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with extended precision values wherever a three-dimensional vector with double precision is expected. The fourth dimension is thrown away and some accuracy is lost because of the conversion.

64.3.169 assign(Tvector4_extended):Tvector3_extended

Synopsis: Allow assignment of four-dimensional extended precision vector to three-dimensional extended precision vector

Declaration: `operator :=(const v: Tvector4_extended) : Tvector3_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with extended precision values wherever a three-dimensional vector with extended precision is expected. The fourth dimensions are thrown away.

64.3.170 assign(Tvector4_extended):Tvector3_single

Synopsis: Allow assignment of four-dimensional extended precision vector to three-dimensional single precision vector

Declaration: `operator :=(const v: Tvector4_extended) : Tvector3_single`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with extended precision values wherever a three-dimensional vector with single precision is expected. The fourth dimension is thrown away and some accuracy is los because of the conversion.

64.3.171 assign(Tvector4_extended):Tvector4_double

Synopsis: Allow assignment of four-dimensional single precision vector to four-dimensional double precision vector

Declaration: `operator :=(const v: Tvector4_extended) : Tvector4_double`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with extended precision values wherever a four-dimensional vector with double precision is expected. Some accuracy is lost because of the conversion.

64.3.172 assign(Tvector4_extended):Tvector4_single

Synopsis: Allow assignment of four-dimensional extended precision vector to four-dimensional single precision vector

Declaration: `operator :=(const v: Tvector4_extended) : Tvector4_single`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with extended precision values wherever a four-dimensional vector with single precision is expected. Some accuracy is lost because of the conversion.

64.3.173 assign(Tvector4_single):Tvector2_double

Synopsis: Allow assignment of four-dimensional single precision vector to two-dimensional double precision vector

Declaration: `operator :=(const v: Tvector4_single) : Tvector2_double`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with single precision values wherever a two-dimensional vector with double precision is expected. The third and fourth dimensions are thrown away.

64.3.174 assign(Tvector4_single):Tvector2_extended

Synopsis: Allow assignment of four-dimensional single precision vector to two-dimensional extended precision vector

Declaration: `operator :=(const v: Tvector4_single) : Tvector2_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with single precision values wherever a two-dimensional vector with extended precision is expected. The third and fourth dimensions are thrown away.

64.3.175 assign(Tvector4_single):Tvector2_single

Synopsis: Allow assignment of four-dimensional single precision vector to two-dimensional single precision vector

Declaration: `operator :=(const v: Tvector4_single) : Tvector2_single`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with single precision values wherever a two-dimensional vector with single precision is expected. The third and fourth dimensions are thrown away.

64.3.176 assign(Tvector4_single):Tvector3_double

Synopsis: Allow assignment of four-dimensional single precision vector to three-dimensional double precision vector

Declaration: `operator :=(const v: Tvector4_single) : Tvector3_double`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with single precision values wherever a three-dimensional vector with double precision is expected. The fourth dimension is thrown away.

64.3.177 assign(Tvector4_single):Tvector3_extended

Synopsis: Allow assignment of four-dimensional single precision vector to three-dimensional extended precision vector

Declaration: `operator :=(const v: Tvector4_single) : Tvector3_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with single precision values wherever a three-dimensional vector with extended precision is expected. The fourth dimension is thrown away.

64.3.178 assign(Tvector4_single):Tvector3_single

Synopsis: Allow assignment of four-dimensional single precision vector to three-dimensional single precision vector

Declaration: `operator :=(const v: Tvector4_single) : Tvector3_single`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with single precision values wherever a three-dimensional vector with single precision is expected. The fourth dimension is thrown away.

64.3.179 assign(Tvector4_single):Tvector4_double

Synopsis: Allow assignment of four-dimensional single precision vector to four-dimensional double precision vector

Declaration: `operator :=(const v: Tvector4_single) : Tvector4_double`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with single precision values wherever a four-dimensional vector with double precision is expected.

64.3.180 assign(Tvector4_single):Tvector4_extended

Synopsis: Allow assignment of four-dimensional single precision vector to four-dimensional extended precision vector

Declaration: `operator :=(const v: Tvector4_single) : Tvector4_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with single precision values wherever a four-dimensional vector with extended precision is expected.

64.3.181 divide(Tmatrix2_double,Double):Tmatrix2_double

Synopsis: Divide a two-dimensional single precision matrix by a scalar

Declaration:
```
operator /(const m: Tmatrix2_double;const x: Double) : Tmatrix2_double
```

Visibility: default

Description: This operator allows you to divide a matrix by a scalar. All elements in the matrix are divided by the scalar, the result is returned as a new matrix.

64.3.182 divide(Tmatrix2_extended,extended):Tmatrix2_extended

Synopsis: Divide a two-dimensional single precision matrix by a scalar

Declaration:
```
operator /(const m: Tmatrix2_extended;const x: extended)
                  : Tmatrix2_extended
```

Visibility: default

Description: This operator allows you to divide a matrix by a scalar. All elements in the matrix are divided by the scalar, the result is returned as a new matrix.

64.3.183 divide(Tmatrix2_single,single):Tmatrix2_single

Synopsis: Divide a two-dimensional single precision matrix by a scalar

Declaration:
```
operator /(const m: Tmatrix2_single;const x: single) : Tmatrix2_single
```

Visibility: default

Description: This operator allows you to divide a matrix by a scalar. All elements in the matrix are divided by the scalar, the result is returned as a new matrix.

64.3.184 divide(Tmatrix3_double,Double):Tmatrix3_double

Synopsis: Divide a two-dimensional single precision matrix by a scalar

Declaration:
```
operator /(const m: Tmatrix3_double;const x: Double) : Tmatrix3_double
```

Visibility: default

Description: This operator allows you to divide a matrix by a scalar. All elements in the matrix are divided by the scalar, the result is returned as a new matrix.

64.3.185 divide(Tmatrix3_extended,extended):Tmatrix3_extended

Synopsis: Divide a two-dimensional single precision matrix by a scalar

Declaration:
```
operator /(const m: Tmatrix3_extended;const x: extended)
                  : Tmatrix3_extended
```

Visibility: default

Description: This operator allows you to divide a matrix by a scalar. All elements in the matrix are divided by the scalar, the result is returned as a new matrix.

64.3.186 divide(Tmatrix3_single,single):Tmatrix3_single

Synopsis: Divide a two-dimensional single precision matrix by a scalar

Declaration:
```
operator /(const m: Tmatrix3_single;const x: single) : Tmatrix3_single
```

Visibility: default

Description: This operator allows you to divide a matrix by a scalar. All elements in the matrix are divided by the scalar, the result is returned as a new matrix.

64.3.187 divide(Tmatrix4_double,Double):Tmatrix4_double

Synopsis: Divide a two-dimensional single precision matrix by a scalar

Declaration:
```
operator /(const m: Tmatrix4_double;const x: Double) : Tmatrix4_double
```

Visibility: default

Description: This operator allows you to divide a matrix by a scalar. All elements in the matrix are divided by the scalar, the result is returned as a new matrix.

64.3.188 divide(Tmatrix4_extended,extended):Tmatrix4_extended

Synopsis: Divide a two-dimensional single precision matrix by a scalar

Declaration:
```
operator /(const m: Tmatrix4_extended;const x: extended)
                : Tmatrix4_extended
```

Visibility: default

Description: This operator allows you to divide a matrix by a scalar. All elements in the matrix are divided by the scalar, the result is returned as a new matrix.

64.3.189 divide(Tmatrix4_single,single):Tmatrix4_single

Synopsis: Divide a two-dimensional single precision matrix by a scalar

Declaration:
```
operator /(const m: Tmatrix4_single;const x: single) : Tmatrix4_single
```

Visibility: default

Description: This operator allows you to divide a matrix by a scalar. All elements in the matrix are divided by the scalar, the result is returned as a new matrix.

64.3.190 divide(Tvector2_double,Double):Tvector2_double

Synopsis: Divide a two-dimensional double precision vector by a scalar

Declaration:
```
operator /(const x: Tvector2_double;y: Double) : Tvector2_double
```

Visibility: default

Description: This operator allows you to divide a vector by a scalar value. Each vector element is divided by the scalar value; the result is returned as a new vector.

64.3.191 divide(Tvector2_extended,extended):Tvector2_extended

Synopsis: Divide a two-dimensional extended precision vector by a scalar

Declaration: `operator /(const x: Tvector2_extended;y: extended) : Tvector2_extended`

Visibility: default

Description: This operator allows you to divide a vector by a scalar value. Each vector element is divided by the scalar value; the result is returned as a new vector.

64.3.192 divide(Tvector2_single,single):Tvector2_single

Synopsis: Divide a two-dimensional single precision vector by a scalar

Declaration: `operator /(const x: Tvector2_single;y: single) : Tvector2_single`

Visibility: default

Description: This operator allows you to divide a vector by a scalar value. Each vector element is divided by the scalar value; the result is returned as a new vector.

64.3.193 divide(Tvector3_double,Double):Tvector3_double

Synopsis: Divide a three-dimensional double precision vector by a scalar

Declaration: `operator /(const x: Tvector3_double;y: Double) : Tvector3_double`

Visibility: default

Description: This operator allows you to divide a vector by a scalar value. Each vector element is divided by the scalar value; the result is returned as a new vector.

64.3.194 divide(Tvector3_extended,extended):Tvector3_extended

Synopsis: Divide a three-dimensional extended precision vector by a scalar

Declaration: `operator /(const x: Tvector3_extended;y: extended) : Tvector3_extended`

Visibility: default

Description: This operator allows you to divide a vector by a scalar value. Each vector element is divided by the scalar value; the result is returned as a new vector.

64.3.195 divide(Tvector3_single,single):Tvector3_single

Synopsis: Divide a three-dimensional single precision vector by a scalar

Declaration: `operator /(const x: Tvector3_single;y: single) : Tvector3_single`

Visibility: default

Description: This operator allows you to divide a vector by a scalar value. Each vector element is divided by the scalar value; the result is returned as a new vector.

64.3.196 divide(Tvector4_double,Double):Tvector4_double

Synopsis: Divide a four-dimensional double precision vector by a scalar

Declaration: `operator /(const x: Tvector4_double;y: Double) : Tvector4_double`

Visibility: default

Description: This operator allows you to divide a vector by a scalar value. Each vector element is divided by the scalar value; the result is returned as a new vector.

64.3.197 divide(Tvector4_extended,extended):Tvector4_extended

Synopsis: Divide a four-dimensional extended precision vector by a scalar

Declaration: `operator /(const x: Tvector4_extended;y: extended) : Tvector4_extended`

Visibility: default

Description: This operator allows you to divide a vector by a scalar value. Each vector element is divided by the scalar value; the result is returned as a new vector.

64.3.198 divide(Tvector4_single,single):Tvector4_single

Synopsis: Divide a four-dimensional single precision vector by a scalar

Declaration: `operator /(const x: Tvector4_single;y: single) : Tvector4_single`

Visibility: default

Description: This operator allows you to divide a vector by a scalar value. Each vector element is divided by the scalar value; the result is returned as a new vector.

64.3.199 multiply(Tmatrix2_double,Double):Tmatrix2_double

Synopsis: Multiply a two-dimensional double precision matrix by a scalar

Declaration: `operator *(const m: Tmatrix2_double;const x: Double) : Tmatrix2_double`

Visibility: default

Description: This operator allows you to multiply a matrix with a scalar. All elements in the matrix are multiplied by the scalar, the result is returned as a new matrix.

64.3.200 multiply(Tmatrix2_double,Tmatrix2_double):Tmatrix2_double

Synopsis: Give product of two two-dimensional double precision matrices

Declaration:
```
operator *(const m1: Tmatrix2_double;const m2: Tmatrix2_double)
  : Tmatrix2_double
```

Visibility: default

Description: This operator allows you to multipy two two-dimensional single precision matrices. A new matrix is returned which is the product of both matrices. The product is calculated using the well known matrix multiplication algorithm.

64.3.201 multiply(Tmatrix2_double,Tvector2_double):Tvector2_double

Synopsis: Give product of a two-dimendional double precision matrix and vector

Declaration:
```
operator *(const m: Tmatrix2_double;const v: Tvector2_double)
  : Tvector2_double
```

Visibility: default

Description: This operator allows you to multipy a two-dimensional double precision matrices with a two dimensional double precision vector. A new vector is returned which is the product of the matrix and the vector. The product is calculated using the well known matrix-vector multiplication algorithm.

64.3.202 multiply(Tmatrix2_extended,extended):Tmatrix2_extended

Synopsis: Multiply a two-dimensional extended precision matrix by a scalar

Declaration:
```
operator *(const m: Tmatrix2_extended;const x: extended)
  : Tmatrix2_extended
```

Visibility: default

Description: This operator allows you to multiply a matrix with a scalar. All elements in the matrix are multiplied by the scalar, the result is returned as a new matrix.

64.3.203 multiply(Tmatrix2_extended,Tmatrix2_extended):Tmatrix2_extended

Synopsis: Give product of two two-dimensional extended precision matrices

Declaration:
```
operator *(const m1: Tmatrix2_extended;const m2: Tmatrix2_extended)
  : Tmatrix2_extended
```

Visibility: default

Description: This operator allows you to multipy two two-dimensional single precision matrices. A new matrix is returned which is the product of both matrices. The product is calculated using the well known matrix multiplication algorithm.

64.3.204 multiply(Tmatrix2_extended,Tvector2_extended):Tvector2_extended

Synopsis: Give product of a two-dimendional extended precision matrix and vector

Declaration:
```
operator *(const m: Tmatrix2_extended;const v: Tvector2_extended)
  : Tvector2_extended
```

Visibility: default

Description: This operator allows you to multipy a two-dimensional extended precision matrices with a two dimensional extended precision vector. A new vector is returned which is the product of the matrix and the vector. The product is calculated using the well known matrix-vector multiplication algorithm.

64.3.205 multiply(Tmatrix2_single,single):Tmatrix2_single

Synopsis: Multiply a two-dimensional single precision matrix by a scalar

Declaration:
```
operator *(const m: Tmatrix2_single;const x: single) : Tmatrix2_single
```

Visibility: default

Description: This operator allows you to multiply a matrix with a scalar. All elements in the matrix are multiplied by the scalar, the result is returned as a new matrix.

64.3.206 multiply(Tmatrix2_single,Tmatrix2_single):Tmatrix2_single

Synopsis: Give product of two two-dimensional single precision matrices

Declaration:
```
operator *(const m1: Tmatrix2_single;const m2: Tmatrix2_single)
  : Tmatrix2_single
```

Visibility: default

Description: This operator allows you to multipy two two-dimensional single precision matrices. A new matrix is returned which is the product of both matrices. The product is calculated using the well known matrix multiplication algorithm.

64.3.207 multiply(Tmatrix2_single,Tvector2_single):Tvector2_single

Synopsis: Give product of a two-dimendional single precision matrix and vector

Declaration:
```
operator *(const m: Tmatrix2_single;const v: Tvector2_single)
  : Tvector2_single
```

Visibility: default

Description: This operator allows you to multipy a two-dimensional single precision matrices with a two dimensional single precision vector. A new vector is returned which is the product of the matrix and the vector. The product is calculated using the well known matrix-vector multiplication algorithm.

64.3.208 multiply(Tmatrix3_double,Double):Tmatrix3_double

Synopsis: Multiply a three-dimensional double precision matrix by a scalar

Declaration:
```
operator *(const m: Tmatrix3_double;const x: Double) : Tmatrix3_double
```

Visibility: default

Description: This operator allows you to multiply a matrix with a scalar. All elements in the matrix are multiplied by the scalar, the result is returned as a new matrix.

64.3.209 multiply(Tmatrix3_double,Tmatrix3_double):Tmatrix3_double

Synopsis: Give product of two three-dimensional double precision matrices

Declaration:
```
operator *(const m1: Tmatrix3_double;const m2: Tmatrix3_double)
  : Tmatrix3_double
```

Visibility: default

Description: This operator allows you to multipy two two-dimensional single precision matrices. A new matrix is returned which is the product of both matrices. The product is calculated using the well known matrix multiplication algorithm.

64.3.210 multiply(Tmatrix3_double,Tvector3_double):Tvector3_double

Synopsis: Give product of a three-dimendional double precision matrix and vector

Declaration:
```
operator *(const m: Tmatrix3_double;const v: Tvector3_double)
  : Tvector3_double
```

Visibility: default

Description: This operator allows you to multipy a three-dimensional double precision matrices with a three dimensional double precision vector. A new vector is returned which is the product of the matrix and the vector. The product is calculated using the well known matrix-vector multiplication algorithm.

64.3.211 multiply(Tmatrix3_extended,extended):Tmatrix3_extended

Synopsis: Multiply a three-dimensional extended precision matrix by a scalar

Declaration:
```
operator *(const m: Tmatrix3_extended;const x: extended)
  : Tmatrix3_extended
```

Visibility: default

Description: This operator allows you to multiply a matrix with a scalar. All elements in the matrix are multiplied by the scalar, the result is returned as a new matrix.

64.3.212 multiply(Tmatrix3_extended,Tmatrix3_extended):Tmatrix3_extended

Synopsis: Give product of two three-dimensional extended precision matrices

Declaration:
```
operator *(const m1: Tmatrix3_extended;const m2: Tmatrix3_extended)
  : Tmatrix3_extended
```

Visibility: default

Description: This operator allows you to multipy two two-dimensional single precision matrices. A new matrix is returned which is the product of both matrices. The product is calculated using the well known matrix multiplication algorithm.

64.3.213 multiply(Tmatrix3_extended,Tvector3_extended):Tvector3_extended

Synopsis: Give product of a three-dimendional extended precision matrix and vector

Declaration:
```
operator *(const m: Tmatrix3_extended;const v: Tvector3_extended)
  : Tvector3_extended
```

Visibility: default

Description: This operator allows you to multipy a three-dimensional extended precision matrices with a three dimensional extended precision vector. A new vector is returned which is the product of the matrix and the vector. The product is calculated using the well known matrix-vector multiplication algorithm.

64.3.214 multiply(Tmatrix3_single,single):Tmatrix3_single

Synopsis: Multiply a three-dimensional single precision matrix by a scalar

Declaration:
```
operator *(const m: Tmatrix3_single;const x: single) : Tmatrix3_single
```

Visibility: default

Description: This operator allows you to multiply a matrix with a scalar. All elements in the matrix are multiplied by the scalar, the result is returned as a new matrix.

64.3.215 multiply(Tmatrix3_single,Tmatrix3_single):Tmatrix3_single

Synopsis: Give product of two three-dimensional single precision matrices

Declaration:
```
operator *(const m1: Tmatrix3_single;const m2: Tmatrix3_single)
  : Tmatrix3_single
```

Visibility: default

Description: This operator allows you to multipy two two-dimensional single precision matrices. A new matrix is returned which is the product of both matrices. The product is calculated using the well known matrix multiplication algorithm.

64.3.216 multiply(Tmatrix3_single,Tvector3_single):Tvector3_single

Synopsis: Give product of a three-dimendional single precision matrix and vector

Declaration:
```
operator *(const m: Tmatrix3_single;const v: Tvector3_single)
  : Tvector3_single
```

Visibility: default

Description: This operator allows you to multipy a three-dimensional single precision matrices with a three dimensional single precision vector. A new vector is returned which is the product of the matrix and the vector. The product is calculated using the well known matrix-vector multiplication algorithm.

64.3.217 multiply(Tmatrix4_double,Double):Tmatrix4_double

Synopsis: Multiply a four-dimensional double precision matrix by a scalar

Declaration:
```
operator *(const m: Tmatrix4_double;const x: Double) : Tmatrix4_double
```

Visibility: default

Description: This operator allows you to multiply a matrix with a scalar. All elements in the matrix are multiplied by the scalar, the result is returned as a new matrix.

64.3.218 multiply(Tmatrix4_double,Tmatrix4_double):Tmatrix4_double

Synopsis: Give product of two four-dimensional double precision matrices

Declaration:
```
operator *(const m1: Tmatrix4_double;const m2: Tmatrix4_double)
  : Tmatrix4_double
```

Visibility: default

Description: This operator allows you to multipy two two-dimensional single precision matrices. A new matrix is returned which is the product of both matrices. The product is calculated using the well known matrix multiplication algorithm.

64.3.219 multiply(Tmatrix4_double,Tvector4_double):Tvector4_double

Synopsis: Give product of a four-dimendional double precision matrix and vector

Declaration:
```
operator *(const m: Tmatrix4_double;const v: Tvector4_double)
  : Tvector4_double
```

Visibility: default

Description: This operator allows you to multipy a four-dimensional double precision matrices with a four dimensional double precision vector. A new vector is returned which is the product of the matrix and the vector. The product is calculated using the well known matrix-vector multiplication algorithm.

64.3.220 multiply(Tmatrix4_extended,extended):Tmatrix4_extended

Synopsis: Multiply a four-dimensional extended precision matrix by a scalar

Declaration:
```
operator *(const m: Tmatrix4_extended;const x: extended)
  : Tmatrix4_extended
```

Visibility: default

Description: This operator allows you to multiply a matrix with a scalar. All elements in the matrix are multiplied by the scalar, the result is returned as a new matrix.

64.3.221 multiply(Tmatrix4_extended,Tmatrix4_extended):Tmatrix4_extended

Synopsis: Give product of two four-dimensional extended precision matrices

Declaration:
```
operator *(const m1: Tmatrix4_extended;const m2: Tmatrix4_extended)
  : Tmatrix4_extended
```

Visibility: default

Description: This operator allows you to multipy two two-dimensional single precision matrices. A new matrix is returned which is the product of both matrices. The product is calculated using the well known matrix multiplication algorithm.

64.3.222 multiply(Tmatrix4_extended,Tvector4_extended):Tvector4_extended

Synopsis: Give product of a four-dimendional extended precision matrix and vector

Declaration:
```
operator *(const m: Tmatrix4_extended;const v: Tvector4_extended)
  : Tvector4_extended
```

Visibility: default

Description: This operator allows you to multipy a four-dimensional extended precision matrices with a four dimensional extended precision vector. A new vector is returned which is the product of the matrix and the vector. The product is calculated using the well known matrix-vector multiplication algorithm.

64.3.223 multiply(Tmatrix4_single,single):Tmatrix4_single

Synopsis: Multiply a four-dimensional single precision matrix by a scalar

Declaration:
```
operator *(const m: Tmatrix4_single;const x: single) : Tmatrix4_single
```

Visibility: default

Description: This operator allows you to multiply a matrix with a scalar. All elements in the matrix are multiplied by the scalar, the result is returned as a new matrix.

64.3.224 multiply(Tmatrix4_single,Tmatrix4_single):Tmatrix4_single

Synopsis: Give product of two four-dimensional single precision matrices

Declaration:
```
operator *(const m1: Tmatrix4_single;const m2: Tmatrix4_single)
                  : Tmatrix4_single
```

Visibility: default

Description: This operator allows you to multipy two two-dimensional single precision matrices. A new matrix is returned which is the product of both matrices. The product is calculated using the well known matrix multiplication algorithm.

64.3.225 multiply(Tmatrix4_single,Tvector4_single):Tvector4_single

Synopsis: Give product of a four-dimendional single precision matrix and vector

Declaration:
```
operator *(const m: Tmatrix4_single;const v: Tvector4_single)
                  : Tvector4_single
```

Visibility: default

Description: This operator allows you to multipy a four-dimensional single precision matrices with a four dimensional single precision vector. A new vector is returned which is the product of the matrix and the vector. The product is calculated using the well known matrix-vector multiplication algorithm.

64.3.226 multiply(Tvector2_double,Double):Tvector2_double

Synopsis: Multiply a two-dimensional double precision vector by a scalar

Declaration:
```
operator *(const x: Tvector2_double;y: Double) : Tvector2_double
```

Visibility: default

Description: This operator allows you to multiply a vector by a scalar value. Each vector element is multiplied by the scalar value; the result is returned as a new vector.

64.3.227 multiply(Tvector2_double,Tvector2_double):Tvector2_double

Synopsis: Multiply two vectors element wise

Declaration:
```
operator *(const x: Tvector2_double;const y: Tvector2_double)
                  : Tvector2_double
```

Visibility: default

Description: This operator returns a vector that contains the element by element multiplication of the two multiplied vectors.

64.3.228 multiply(Tvector2_extended,extended):Tvector2_extended

Synopsis: Multiply a two-dimensional extended precision vector by a scalar

Declaration:
```
operator *(const x: Tvector2_extended;y: extended) : Tvector2_extended
```

Visibility: default

Description: This operator allows you to multiply a vector by a scalar value. Each vector element is multiplied by the scalar value; the result is returned as a new vector.

64.3.229 multiply(Tvector2_extended,Tvector2_extended):Tvector2_extended

Synopsis: Multiply two vectors element wise

Declaration:
```
operator *(const x: Tvector2_extended;const y: Tvector2_extended)
  : Tvector2_extended
```

Visibility: default

Description: This operator returns a vector that contains the element by element multiplication of the two multiplied vectors.

64.3.230 multiply(Tvector2_single,single):Tvector2_single

Synopsis: Multiply a two-dimensional single precision vector by a scalar

Declaration:
```
operator *(const x: Tvector2_single;y: single) : Tvector2_single
```

Visibility: default

Description: This operator allows you to multiply a vector by a scalar value. Each vector element is multiplied by the scalar value; the result is returned as a new vector.

64.3.231 multiply(Tvector2_single,Tvector2_single):Tvector2_single

Synopsis: Multiply two vectors element wise

Declaration:
```
operator *(const x: Tvector2_single;const y: Tvector2_single)
  : Tvector2_single
```

Visibility: default

Description: This operator returns a vector that contains the element by element multiplication of the two multiplied vectors.

64.3.232 multiply(Tvector3_double,Double):Tvector3_double

Synopsis: Multiply a three-dimensional double precision vector by a scalar

Declaration:
```
operator *(const x: Tvector3_double;y: Double) : Tvector3_double
```

Visibility: default

Description: This operator allows you to multiply a vector by a scalar value. Each vector element is multiplied by the scalar value; the result is returned as a new vector.

64.3.233 multiply(Tvector3_double,Tvector3_double):Tvector3_double

Synopsis: Multiply two vectors element wise

Declaration:
```
operator *(const x: Tvector3_double;const y: Tvector3_double)
  : Tvector3_double
```

Visibility: default

Description: This operator returns a vector that contains the element by element multiplication of the two multiplied vectors.

64.3.234 multiply(Tvector3_extended,extended):Tvector3_extended

Synopsis: Multiply a three-dimensional extended precision vector by a scalar

Declaration:
```
operator *(const x: Tvector3_extended;y: extended) : Tvector3_extended
```

Visibility: default

Description: This operator allows you to multiply a vector by a scalar value. Each vector element is multiplied by the scalar value; the result is returned as a new vector.

64.3.235 multiply(Tvector3_extended,Tvector3_extended):Tvector3_extended

Synopsis: Multiply two vectors element wise

Declaration:
```
operator *(const x: Tvector3_extended;const y: Tvector3_extended)
  : Tvector3_extended
```

Visibility: default

Description: This operator returns a vector that contains the element by element multiplication of the two multiplied vectors.

64.3.236 multiply(Tvector3_single,single):Tvector3_single

Synopsis: Multiply a three-dimensional single precision vector by a scalar

Declaration:
```
operator *(const x: Tvector3_single;y: single) : Tvector3_single
```

Visibility: default

Description: This operator allows you to multiply a vector by a scalar value. Each vector element is multiplied by the scalar value; the result is returned as a new vector.

64.3.237 multiply(Tvector3_single,Tvector3_single):Tvector3_single

Synopsis: Multiply two vectors element wise

Declaration:
```
operator *(const x: Tvector3_single;const y: Tvector3_single)
  : Tvector3_single
```

Visibility: default

Description: This operator returns a vector that contains the element by element multiplication of the two multiplied vectors.

64.3.238 multiply(Tvector4_double,Double):Tvector4_double

Synopsis: Multiply a four-dimensional double precision vector by a scalar

Declaration:
```
operator *(const x: Tvector4_double;y: Double) : Tvector4_double
```

Visibility: default

Description: This operator allows you to multiply a vector by a scalar value. Each vector element is multiplied by the scalar value; the result is returned as a new vector.

64.3.239 multiply(Tvector4_double,Tvector4_double):Tvector4_double

Synopsis: Multiply two vectors element wise

Declaration:
```
operator *(const x: Tvector4_double;const y: Tvector4_double)
  : Tvector4_double
```

Visibility: default

Description: This operator returns a vector that contains the element by element multiplication of the two multiplied vectors.

64.3.240 multiply(Tvector4_extended,extended):Tvector4_extended

Synopsis: Multiply a four-dimensional extended precision vector by a scalar

Declaration:
```
operator *(const x: Tvector4_extended;y: extended) : Tvector4_extended
```

Visibility: default

Description: This operator allows you to multiply a vector by a scalar value. Each vector element is multiplied by the scalar value; the result is returned as a new vector.

64.3.241 multiply(Tvector4_extended,Tvector4_extended):Tvector4_extended

Synopsis: Multiply two vectors element wise

Declaration:
```
operator *(const x: Tvector4_extended;const y: Tvector4_extended)
  : Tvector4_extended
```

Visibility: default

Description: This operator returns a vector that contains the element by element multiplication of the two multiplied vectors.

64.3.242 multiply(Tvector4_single,single):Tvector4_single

Synopsis: Multiply a four-dimensional single precision vector by a scalar

Declaration:
```
operator *(const x: Tvector4_single;y: single) : Tvector4_single
```

Visibility: default

Description: This operator allows you to multiply a vector by a scalar value. Each vector element is multiplied by the scalar value; the result is returned as a new vector.

64.3.243 multiply(Tvector4_single,Tvector4_single):Tvector4_single

Synopsis: Multiply two vectors element wise

Declaration:
```
operator *(const x: Tvector4_single;const y: Tvector4_single)
                 : Tvector4_single
```

Visibility: default

Description: This operator returns a vector that contains the element by element multiplication of the two multiplied vectors.

64.3.244 negative(Tmatrix2_double):Tmatrix2_double

Synopsis: Negate two-dimensional double precision matrix.

Declaration:
```
operator -(const m1: Tmatrix2_double) : Tmatrix2_double
```

Visibility: default

Description: This operation returns a matrix with all elements negated.

64.3.245 negative(Tmatrix2_extended):Tmatrix2_extended

Synopsis: Negate two-dimensional extended precision matrix.

Declaration:
```
operator -(const m1: Tmatrix2_extended) : Tmatrix2_extended
```

Visibility: default

Description: This operation returns a matrix with all elements negated.

64.3.246 negative(Tmatrix2_single):Tmatrix2_single

Synopsis: Negate two-dimensional single precision matrix.

Declaration:
```
operator -(const m1: Tmatrix2_single) : Tmatrix2_single
```

Visibility: default

Description: This operation returns a matrix with all elements negated.

64.3.247 negative(Tmatrix3_double):Tmatrix3_double

Synopsis: Negate three-dimensional double precision matrix.

Declaration:
```
operator -(const m1: Tmatrix3_double) : Tmatrix3_double
```

Visibility: default

Description: This operation returns a matrix with all elements negated.

64.3.248 negative(Tmatrix3_extended):Tmatrix3_extended

Synopsis: Negate three-dimensional extended precision matrix.

Declaration:
```
operator -(const m1: Tmatrix3_extended) : Tmatrix3_extended
```

Visibility: default

Description: This operation returns a matrix with all elements negated.

64.3.249 negative(Tmatrix3_single):Tmatrix3_single

Synopsis: Negate three-dimensional single precision matrix.

Declaration: `operator -(const m1: Tmatrix3_single) : Tmatrix3_single`

Visibility: default

Description: This operation returns a matrix with all elements negated.

64.3.250 negative(Tmatrix4_double):Tmatrix4_double

Synopsis: Negate four-dimensional double precision matrix.

Declaration: `operator -(const m1: Tmatrix4_double) : Tmatrix4_double`

Visibility: default

Description: This operation returns a matrix with all elements negated.

64.3.251 negative(Tmatrix4_extended):Tmatrix4_extended

Synopsis: Negate four-dimensional extended precision matrix.

Declaration: `operator -(const m1: Tmatrix4_extended) : Tmatrix4_extended`

Visibility: default

Description: This operation returns a matrix with all elements negated.

64.3.252 negative(Tmatrix4_single):Tmatrix4_single

Synopsis: Negate four-dimensional single precision matrix.

Declaration: `operator -(const m1: Tmatrix4_single) : Tmatrix4_single`

Visibility: default

Description: This operation returns a matrix with all elements negated.

64.3.253 negative(Tvector2_double):Tvector2_double

Synopsis: Negate two-dimensional vector.

Declaration: `operator -(const x: Tvector2_double) : Tvector2_double`

Visibility: default

Description: This operation returns a vector in the opposite direction of the vector that is passed. In order to do so, all values in the vector are negated.

64.3.254 negative(Tvector2_extended):Tvector2_extended

Synopsis: Negate two-dimensional vector.

Declaration: `operator -(const x: Tvector2_extended) : Tvector2_extended`

Visibility: default

Description: This operation returns a vector in the opposite direction of the vector that is passed. In order to do so, all values in the vector are negated.

64.3.255 negative(Tvector2_single):Tvector2_single

Synopsis: Negate two-dimensional vector.

Declaration: `operator -(const x: Tvector2_single) : Tvector2_single`

Visibility: default

Description: This operation returns a vector in the opposite direction of the vector that is passed. In order to do so, all values in the vector are negated.

64.3.256 negative(Tvector3_double):Tvector3_double

Synopsis: Negate three-dimensional vector.

Declaration: `operator -(const x: Tvector3_double) : Tvector3_double`

Visibility: default

Description: This operation returns a vector in the opposite direction of the vector that is passed. In order to do so, all values in the vector are negated.

64.3.257 negative(Tvector3_extended):Tvector3_extended

Synopsis: Negate three-dimensional vector.

Declaration: `operator -(const x: Tvector3_extended) : Tvector3_extended`

Visibility: default

Description: This operation returns a vector in the opposite direction of the vector that is passed. In order to do so, all values in the vector are negated.

64.3.258 negative(Tvector3_single):Tvector3_single

Synopsis: Negate three-dimensional vector.

Declaration: `operator -(const x: Tvector3_single) : Tvector3_single`

Visibility: default

Description: This operation returns a vector in the opposite direction of the vector that is passed. In order to do so, all values in the vector are negated.

64.3.259 negative(Tvector4_double):Tvector4_double

Synopsis: Negate four-dimensional vector.

Declaration: `operator -(const x: Tvector4_double) : Tvector4_double`

Visibility: default

Description: This operation returns a vector in the opposite direction of the vector that is passed. In order to do so, all values in the vector are negated.

64.3.260 negative(Tvector4_extended):Tvector4_extended

Synopsis: Negate four-dimensional vector.

Declaration:
```
operator -(const x: Tvector4_extended) : Tvector4_extended
```

Visibility: default

Description: This operation returns a vector in the opposite direction of the vector that is passed. In order to do so, all values in the vector are negated.

64.3.261 negative(Tvector4_single):Tvector4_single

Synopsis: Negate four-dimensional vector.

Declaration:
```
operator -(const x: Tvector4_single) : Tvector4_single
```

Visibility: default

Description: This operation returns a vector in the opposite direction of the vector that is passed. In order to do so, all values in the vector are negated.

64.3.262 power(Tvector2_double,Tvector2_double):Double

Synopsis: Calculate the internal product of two vectors.

Declaration:
```
operator **(const x: Tvector2_double;const y: Tvector2_double) : Double
```

Visibility: default

Description: This operator returns the internal product of the two vectors, that is, the elements of the two vectors are element-wise multiplied, and then added together.

64.3.263 power(Tvector2_extended,Tvector2_extended):extended

Synopsis: Calculate the internal product of two vectors.

Declaration:
```
operator **(const x: Tvector2_extended;const y: Tvector2_extended)
                  : extended
```

Visibility: default

Description: This operator returns the internal product of the two vectors, that is, the elements of the two vectors are element-wise multiplied, and then added together.

64.3.264 power(Tvector2_single,Tvector2_single):single

Synopsis: Calculate the internal product of two vectors.

Declaration:
```
operator **(const x: Tvector2_single;const y: Tvector2_single) : single
```

Visibility: default

Description: This operator returns the internal product of the two vectors, that is, the elements of the two vectors are element-wise multiplied, and then added together.

64.3.265 power(Tvector3_double,Tvector3_double):Double

Synopsis: Calculate the internal product of two vectors.

Declaration:
```
operator **(const x: Tvector3_double;const y: Tvector3_double) : Double
```

Visibility: default

Description: This operator returns the internal product of the two vectors, that is, the elements of the two vectors are element-wise multiplied, and then added together.

64.3.266 power(Tvector3_extended,Tvector3_extended):extended

Synopsis: Calculate the internal product of two vectors.

Declaration:
```
operator **(const x: Tvector3_extended;const y: Tvector3_extended)
  : extended
```

Visibility: default

Description: This operator returns the internal product of the two vectors, that is, the elements of the two vectors are element-wise multiplied, and then added together.

64.3.267 power(Tvector3_single,Tvector3_single):single

Synopsis: Calculate the internal product of two vectors.

Declaration:
```
operator **(const x: Tvector3_single;const y: Tvector3_single) : single
```

Visibility: default

Description: This operator returns the internal product of the two vectors, that is, the elements of the two vectors are element-wise multiplied, and then added together.

64.3.268 power(Tvector4_double,Tvector4_double):Double

Synopsis: Calculate the internal product of two vectors.

Declaration:
```
operator **(const x: Tvector4_double;const y: Tvector4_double) : Double
```

Visibility: default

Description: This operator returns the internal product of the two vectors, that is, the elements of the two vectors are element-wise multiplied, and then added together.

64.3.269 power(Tvector4_extended,Tvector4_extended):extended

Synopsis: Calculate the internal product of two vectors.

Declaration:
```
operator **(const x: Tvector4_extended;const y: Tvector4_extended)
  : extended
```

Visibility: default

Description: This operator returns the internal product of the two vectors, that is, the elements of the two vectors are element-wise multiplied, and then added together.

64.3.270 power(Tvector4_single,Tvector4_single):single

Synopsis: Calculate the internal product of two vectors.

Declaration:
```
operator **(const x: Tvector4_single;const y: Tvector4_single) : single
```

Visibility: default

Description: This operator returns the internal product of the two vectors, that is, the elements of the two vectors are element-wise multiplied, and then added together.

64.3.271 subtract(Tmatrix2_double,Double):Tmatrix2_double

Synopsis: Subtract scalar to two-dimensional double precision matrix

Declaration:
```
operator -(const m: Tmatrix2_double;const x: Double) : Tmatrix2_double
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a matrix. The scalar is subtracted from all elements of the matrix, the result is returned as a new matrix.

64.3.272 subtract(Tmatrix2_double,Tmatrix2_double):Tmatrix2_double

Synopsis: Subtract a two-dimensional double precision matrix from another.

Declaration:
```
operator -(const m1: Tmatrix2_double;const m2: Tmatrix2_double)
                : Tmatrix2_double
```

Visibility: default

Description: This operator allows you to subtract a two-dimensional double precision matrix from another. A new matrix is returned with all elements of the two matrices subtracted from each other.

64.3.273 subtract(Tmatrix2_extended,extended):Tmatrix2_extended

Synopsis: Add scalar to two-dimensional extended precision matrix

Declaration:
```
operator -(const m: Tmatrix2_extended;const x: extended)
                : Tmatrix2_extended
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a matrix. The scalar is subtracted from all elements of the matrix, the result is returned as a new matrix.

64.3.274 subtract(Tmatrix2_extended,Tmatrix2_extended):Tmatrix2_extended

Synopsis: Subtract a two-dimensional extended precision matrix from another.

Declaration:
```
operator -(const m1: Tmatrix2_extended;const m2: Tmatrix2_extended)
                : Tmatrix2_extended
```

Visibility: default

Description: This operator allows you to subtract a two-dimensional extended precision matrix from another. A new matrix is returned with all elements of the two matrices subtracted from each other.

64.3.275 subtract(Tmatrix2_single,single):Tmatrix2_single

Synopsis: Subtract scalar to two-dimensional single precision matrix

Declaration:
```
operator -(const m: Tmatrix2_single;const x: single) : Tmatrix2_single
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a matrix. The scalar is subtracted from all elements of the matrix, the result is returned as a new matrix.

64.3.276 subtract(Tmatrix2_single,Tmatrix2_single):Tmatrix2_single

Synopsis: Subtract a two-dimensional single precision matrix from another.

Declaration:
```
operator -(const m1: Tmatrix2_single;const m2: Tmatrix2_single)
                : Tmatrix2_single
```

Visibility: default

Description: This operator allows you to subtract a two-dimensional single precision matrix from another. A new matrix is returned with all elements of the two matrices subtracted from each other.

64.3.277 subtract(Tmatrix3_double,Double):Tmatrix3_double

Synopsis: Add scalar to three-dimensional double precision matrix

Declaration:
```
operator -(const m: Tmatrix3_double;const x: Double) : Tmatrix3_double
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a matrix. The scalar is subtracted from all elements of the matrix, the result is returned as a new matrix.

64.3.278 subtract(Tmatrix3_double,Tmatrix3_double):Tmatrix3_double

Synopsis: Subtract a three-dimensional double precision matrix from another.

Declaration:
```
operator -(const m1: Tmatrix3_double;const m2: Tmatrix3_double)
                : Tmatrix3_double
```

Visibility: default

Description: This operator allows you to subtract a three-dimensional double precision matrix from another. A new matrix is returned with all elements of the two matrices subtracted from each other.

64.3.279 subtract(Tmatrix3_extended,extended):Tmatrix3_extended

Synopsis: Add scalar to three-dimensional extended precision matrix

Declaration:
```
operator -(const m: Tmatrix3_extended;const x: extended)
                : Tmatrix3_extended
```

Visibility: default

Description: This operator allows you to substract a scalar value from a matrix. The scalar is subtracted from all elements of the matrix, the result is returned as a new matrix.

64.3.280 subtract(Tmatrix3_extended,Tmatrix3_extended):Tmatrix3_extended

Synopsis: Subtract a three-dimensional extended precision matrix from another.

Declaration:
```
operator -(const m1: Tmatrix3_extended;const m2: Tmatrix3_extended)
  : Tmatrix3_extended
```

Visibility: default

Description: This operator allows you to subtract a three-dimensional extended precision matrix from another. A new matrix is returned with all elements of the two matrices subtracted from each other.

64.3.281 subtract(Tmatrix3_single,single):Tmatrix3_single

Synopsis: Add scalar to three-dimensional single precision matrix

Declaration:
```
operator -(const m: Tmatrix3_single;const x: single) : Tmatrix3_single
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a matrix. The scalar is subtracted from all elements of the matrix, the result is returned as a new matrix.

64.3.282 subtract(Tmatrix3_single,Tmatrix3_single):Tmatrix3_single

Synopsis: Subtract a three-dimensional single precision matrix from another.

Declaration:
```
operator -(const m1: Tmatrix3_single;const m2: Tmatrix3_single)
  : Tmatrix3_single
```

Visibility: default

Description: This operator allows you to subtract a three-dimensional single precision matrix from another. A new matrix is returned with all elements of the two matrices subtracted from each other.

64.3.283 subtract(Tmatrix4_double,Double):Tmatrix4_double

Synopsis: Add scalar to four-dimensional double precision matrix

Declaration:
```
operator -(const m: Tmatrix4_double;const x: Double) : Tmatrix4_double
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a matrix. The scalar is subtracted from all elements of the matrix, the result is returned as a new matrix.

64.3.284 subtract(Tmatrix4_double,Tmatrix4_double):Tmatrix4_double

Synopsis: Subtract a four-dimensional double precision matrix from another.

Declaration:
```
operator -(const m1: Tmatrix4_double;const m2: Tmatrix4_double)
  : Tmatrix4_double
```

Visibility: default

Description: This operator allows you to subtract a four-dimensional double precision matrix from another. A new matrix is returned with all elements of the two matrices subtracted from each other.

64.3.285 subtract(Tmatrix4_extended,extended):Tmatrix4_extended

Synopsis: Add scalar to four-dimensional extended precision matrix

Declaration:
```
operator -(const m: Tmatrix4_extended;const x: extended)
                : Tmatrix4_extended
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a matrix. The scalar is subtracted from all elements of the matrix, the result is returned as a new matrix.

64.3.286 subtract(Tmatrix4_extended,Tmatrix4_extended):Tmatrix4_extended

Synopsis: Subtract a four-dimensional extended precision matrix from another.

Declaration:
```
operator -(const m1: Tmatrix4_extended;const m2: Tmatrix4_extended)
                : Tmatrix4_extended
```

Visibility: default

Description: This operator allows you to subtract a four-dimensional extended precision matrix from another. A new matrix is returned with all elements of the two matrices subtracted from each other.

64.3.287 subtract(Tmatrix4_single,single):Tmatrix4_single

Synopsis: Add scalar to four-dimensional single precision matrix

Declaration:
```
operator -(const m: Tmatrix4_single;const x: single) : Tmatrix4_single
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a matrix. The scalar is subtracted from all elements of the matrix, the result is returned as a new matrix.

64.3.288 subtract(Tmatrix4_single,Tmatrix4_single):Tmatrix4_single

Synopsis: Subtract a four-dimensional single precision matrix from another.

Declaration:
```
operator -(const m1: Tmatrix4_single;const m2: Tmatrix4_single)
                : Tmatrix4_single
```

Visibility: default

Description: This operator allows you to subtract a four-dimensional single precision matrix from another. A new matrix is returned with all elements of the two matrices subtracted from each other.

64.3.289 subtract(Tvector2_double,Double):Tvector2_double

Synopsis: Subtract scalar from two-dimensional double precision vector

Declaration:
```
operator -(const x: Tvector2_double;y: Double) : Tvector2_double
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a vector. The scalar is subtracted from all elements of the vector, the result is returned as a new vector.

64.3.290 subtract(Tvector2_double,Tvector2_double):Tvector2_double

Synopsis: Subtract two-dimensional double precision vectors from each other

Declaration:
```
operator -(const x: Tvector2_double;const y: Tvector2_double)
                : Tvector2_double
```

Visibility: default

Description: This operator allows you to subtract two two-dimensional vectors with double precision from each other. The result is a new vector which consists of the difference of the individual elements of the two vectors.

64.3.291 subtract(Tvector2_extended,extended):Tvector2_extended

Synopsis: Subtract scalar from two-dimensional extended precision vector

Declaration:
```
operator -(const x: Tvector2_extended;y: extended) : Tvector2_extended
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a vector. The scalar is subtracted from all elements of the vector, the result is returned as a new vector.

64.3.292 subtract(Tvector2_extended,Tvector2_extended):Tvector2_extended

Synopsis: Subtract two-dimensional extended precision vectors from each other

Declaration:
```
operator -(const x: Tvector2_extended;const y: Tvector2_extended)
                : Tvector2_extended
```

Visibility: default

Description: This operator allows you to subtract two two-dimensional vectors with extended precision from each other. The result is a new vector which consists of the difference of the individual elements of the two vectors.

64.3.293 subtract(Tvector2_single,single):Tvector2_single

Synopsis: Subtract scalar from two-dimensional single precision vector

Declaration:
```
operator -(const x: Tvector2_single;y: single) : Tvector2_single
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a vector. The scalar is subtracted from all elements of the vector, the result is returned as a new vector.

64.3.294 subtract(Tvector2_single,Tvector2_single):Tvector2_single

Synopsis: Subtract two-dimensional single precision vectors from each other

Declaration:
```
operator -(const x: Tvector2_single;const y: Tvector2_single)
                : Tvector2_single
```

Visibility: default

Description: This operator allows you to subtract two two-dimensional vectors with single precision from each other. The result is a new vector which consists of the difference of the individual elements of the two vectors.

64.3.295 subtract(Tvector3_double,Double):Tvector3_double

Synopsis: Subtract scalar from three-dimensional double precision vector

Declaration:
```
operator -(const x: Tvector3_double;y: Double) : Tvector3_double
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a vector. The scalar is subtracted from all elements of the vector, the result is returned as a new vector.

64.3.296 subtract(Tvector3_double,Tvector3_double):Tvector3_double

Synopsis: Subtract three-dimensional double precision vectors from each other

Declaration:
```
operator -(const x: Tvector3_double;const y: Tvector3_double)
                : Tvector3_double
```

Visibility: default

Description: This operator allows you to subtract two two-dimensional vectors with double precision from each other. The result is a new vector which consists of the difference of the individual elements of the two vectors.

64.3.297 subtract(Tvector3_extended,extended):Tvector3_extended

Synopsis: Subtract scalar from three-dimensional extended precision vector

Declaration:
```
operator -(const x: Tvector3_extended;y: extended) : Tvector3_extended
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a vector. The scalar is subtracted from all elements of the vector, the result is returned as a new vector.

64.3.298 subtract(Tvector3_extended,Tvector3_extended):Tvector3_extended

Synopsis: Subtract three-dimensional extended precision vectors from each other

Declaration:
```
operator -(const x: Tvector3_extended;const y: Tvector3_extended)
                : Tvector3_extended
```

Visibility: default

Description: This operator allows you to subtract two three-dimensional vectors with extended precision from each other. The result is a new vector which consists of the difference of the individual elements of the two vectors.

64.3.299 subtract(Tvector3_single,single):Tvector3_single

Synopsis: Subtract scalar from three-dimensional single precision vector

Declaration:
```
operator -(const x: Tvector3_single;y: single) : Tvector3_single
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a vector. The scalar is subtracted from all elements of the vector, the result is returned as a new vector.

64.3.300 subtract(Tvector3_single,Tvector3_single):Tvector3_single

Synopsis: Subtract three-dimensional single precision vectors from each other

Declaration:
```
operator -(const x: Tvector3_single;const y: Tvector3_single)
                : Tvector3_single
```

Visibility: default

Description: This operator allows you to subtract two three-dimensional vectors with single precision from each other. The result is a new vector which consists of the difference of the individual elements of the two vectors.

64.3.301 subtract(Tvector4_double,Double):Tvector4_double

Synopsis: Subtract scalar from four-dimensional double precision vector

Declaration:
```
operator -(const x: Tvector4_double;y: Double) : Tvector4_double
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a vector. The scalar is subtracted from all elements of the vector, the result is returned as a new vector.

64.3.302 subtract(Tvector4_double,Tvector4_double):Tvector4_double

Synopsis: Subtract four-dimensional double precision vectors from each other

Declaration:
```
operator -(const x: Tvector4_double;const y: Tvector4_double)
                : Tvector4_double
```

Visibility: default

Description: This operator allows you to subtract two four-dimensional vectors with double precision from each other. The result is a new vector which consists of the difference of the individual elements of the two vectors.

64.3.303 subtract(Tvector4_extended,extended):Tvector4_extended

Synopsis: Subtract scalar from four-dimensional extended precision vector

Declaration:
```
operator -(const x: Tvector4_extended;y: extended) : Tvector4_extended
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a vector. The scalar is subtracted from all elements of the vector, the result is returned as a new vector.

64.3.304 subtract(Tvector4_extended,Tvector4_extended):Tvector4_extended

Synopsis: Subtract four-dimensional extended precision vectors from each other

Declaration:
```
operator -(const x: Tvector4_extended;const y: Tvector4_extended)
                : Tvector4_extended
```

Visibility: default

Description: This operator allows you to subtract two four-dimensional vectors with extended precision from each other. The result is a new vector which consists of the difference of the individual elements of the two vectors.

64.3.305 subtract(Tvector4_single,single):Tvector4_single

Synopsis: Subtract scalar from four-dimensional single precision vector

Declaration:
```
operator -(const x: Tvector4_single;y: single) : Tvector4_single
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a vector. The scalar is subtracted from all elements of the vector, the result is returned as a new vector.

64.3.306 subtract(Tvector4_single,Tvector4_single):Tvector4_single

Synopsis: Subtract four-dimensional single precision vectors from each other

Declaration:
```
operator -(const x: Tvector4_single;const y: Tvector4_single)
                : Tvector4_single
```

Visibility: default

Description: This operator allows you to subtract two four-dimensional vectors with single precision from each other. The result is a new vector which consists of the difference of the individual elements of the two vectors.

64.3.307 symmetricaldifference(Tvector3_double,Tvector3_double):Tvector3_double

Synopsis: Calculate the external product of two three-dimensional vectors

Declaration:
```
operator ><(const x: Tvector3_double;const y: Tvector3_double)
                : Tvector3_double
```

Visibility: default

Description: This operator returns the external product of two three dimensional vector. It is a vector orthonormal to the two multiplied vectors. The length of that vector is equal to the surface area of a parallelogram with the two vectors as sides.

The external product is often used to get a vector orthonormal to two other vectors, but of a predefined length. In order to do so, the result vector from the external product, is divided by its length, and then multiplied by the desired size.

64.3.308 symmetricaldifference(Tvector3_extended,Tvector3_extended):Tvector3_exten

Synopsis: Calculate the external product of two three-dimensional vectors

Declaration:
```
operator ><(const x: Tvector3_extended;const y: Tvector3_extended)
                : Tvector3_extended
```

Visibility: default

Description: This operator returns the external product of two three dimensional vector. It is a vector orthonormal to the two multiplied vectors. The length of that vector is equal to the surface area of a parallelogram with the two vectors as sides.

The external product is often used to get a vector orthonormal to two other vectors, but of a predefined length. In order to do so, the result vector from the external product, is divided by its length, and then multiplied by the desired size.

64.3.309 symmetricaldifference(Tvector3_single,Tvector3_single):Tvector3_single

Synopsis: Calculate the external product of two three-dimensional vectors

Declaration:
```
operator ><(const x: Tvector3_single;const y: Tvector3_single)
                  : Tvector3_single
```

Visibility: default

Description: This operator returns the external product of two three dimensional vector. It is a vector orthonormal to the two multiplied vectors. The length of that vector is equal to the surface area of a parallelogram with the two vectors as sides.

The external product is often used to get a vector orthonormal to two other vectors, but of a predefined length. In order to do so, the result vector from the external product, is divided by its length, and then multiplied by the desired size.

64.4 Tmatrix2_double

64.4.1 Description

The `Tmatrix2_double` object provides a matrix of 2*2 double precision scalars.

64.4.2 Method overview

Page	Method	Description
491	determinant	Calculates the determinant of the matrix.
491	get_column	Returns the c-th column of the matrix as vector.
491	get_row	Returns the r-th row of the matrix as vector.
491	init	Initializes the matrix, setting its elements to the values passed to the constructor.
490	init_identity	Initializes the matrix and sets its elements to the identity matrix.
490	init_zero	Initializes the matrix and sets its elements to zero
492	inverse	Calculates the inverse of the matrix.
491	set_column	Sets c-th column of the matrix with a vector.
491	set_row	Sets r-th row of the matrix with a vector.
492	transpose	Returns the transposition of the matrix.

64.4.3 Tmatrix2_double.init_zero

Synopsis: Initializes the matrix and sets its elements to zero

Declaration:
```
constructor init_zero
```

Visibility: default

64.4.4 Tmatrix2_double.init_identity

Synopsis: Initializes the matrix and sets its elements to the identity matrix.

Declaration:
```
constructor init_identity
```

Visibility: default

Description: Initializes the matrix and sets its elements to the identity matrix, that is, elements to 1 on the left-upper to right-lower diagonal, the rest zero.

64.4.5 Tmatrix2_double.init

Synopsis: Initializes the matrix, setting its elements to the values passed to the constructor.

Declaration: `constructor init(aa: Double;ab: Double;ba: Double;bb: Double)`

Visibility: default

Description: Initializes the matrix, setting its elements to the values passed to the constructor. The order of the values is left to right, then top to bottom.

64.4.6 Tmatrix2_double.get_column

Synopsis: Returns the c-th column of the matrix as vector.

Declaration: `function get_column(c: Byte) : Tvector2_double`

Visibility: default

Description: Returns the `c`-th column of the matrix as vector. The column numbering starts at 0.

64.4.7 Tmatrix2_double.get_row

Synopsis: Returns the r-th row of the matrix as vector.

Declaration: `function get_row(r: Byte) : Tvector2_double`

Visibility: default

Description: Returns the `r`-th row of the matrix as vector. The row numbering starts at 0.

64.4.8 Tmatrix2_double.set_column

Synopsis: Sets c-th column of the matrix with a vector.

Declaration: `procedure set_column(c: Byte;const v: Tvector2_double)`

Visibility: default

Description: Replaces the `c`-th column of the matrix with vector v. The column numbering starts at 0.

64.4.9 Tmatrix2_double.set_row

Synopsis: Sets r-th row of the matrix with a vector.

Declaration: `procedure set_row(r: Byte;const v: Tvector2_double)`

Visibility: default

Description: Replaces the `r`-th row of the matrix with vector v. The row numbering starts at 0.

64.4.10 Tmatrix2_double.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : Double`

Visibility: default

Description: Returns the determinant of the matrix.

64.4.11 Tmatrix2_double.inverse

Synopsis: Calculates the inverse of the matrix.

Declaration: `function inverse(Adeterminant: Double) : Tmatrix2_double`

Visibility: default

Description: Tmatrix2_double.inverse returns a new matrix that is the inverse of the matrix. You must pass the determinant of the matrix as parameter.

64.4.12 Tmatrix2_double.transpose

Synopsis: Returns the transposition of the matrix.

Declaration: `function transpose : Tmatrix2_double`

Visibility: default

Description: Tmatrix2_double.transpose returns a new matrix that is the transposition of the matrix, that is, the matrix with the x and y coordinates of the values swapped.

64.5 Tmatrix2_extended

64.5.1 Description

The `Tmatrix2_extended` object provides a matrix of 2*2 extended precision scalars.

64.5.2 Method overview

Page	Method	Description
494	determinant	Calculates the determinant of the matrix.
493	get_column	Returns the c-th column of the matrix as vector.
493	get_row	Returns the r-th row of the matrix as vector.
493	init	Initializes the matrix, setting its elements to the values passed to the constructor.
493	init_identity	Initializes the matrix and sets its elements to the identity matrix.
492	init_zero	Initializes the matrix and sets its elements to zero
494	inverse	Calculates the inverse of the matrix.
493	set_column	Sets c-th column of the matrix with a vector.
493	set_row	Sets r-th row of the matrix with a vector.
494	transpose	Returns the transposition of the matrix.

64.5.3 Tmatrix2_extended.init_zero

Synopsis: Initializes the matrix and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

64.5.4 Tmatrix2_extended.init_identity

Synopsis: Initializes the matrix and sets its elements to the identity matrix.

Declaration: `constructor init_identity`

Visibility: default

Description: Initializes the matrix and sets its elements to the identity matrix, that is, elements to 1 on the left-upper to right-lower diagonal, the rest zero.

64.5.5 Tmatrix2_extended.init

Synopsis: Initializes the matrix, setting its elements to the values passed to the constructor.

Declaration: `constructor init(aa: extended;ab: extended;ba: extended;bb: extended)`

Visibility: default

Description: Initializes the matrix, setting its elements to the values passed to the constructor. The order of the values is left to right, then top to bottom.

64.5.6 Tmatrix2_extended.get_column

Synopsis: Returns the c-th column of the matrix as vector.

Declaration: `function get_column(c: Byte) : Tvector2_extended`

Visibility: default

Description: Returns the `c`-th column of the matrix as vector. The column numbering starts at 0.

64.5.7 Tmatrix2_extended.get_row

Synopsis: Returns the r-th row of the matrix as vector.

Declaration: `function get_row(r: Byte) : Tvector2_extended`

Visibility: default

Description: Returns the `r`-th row of the matrix as vector. The row numbering starts at 0.

64.5.8 Tmatrix2_extended.set_column

Synopsis: Sets c-th column of the matrix with a vector.

Declaration: `procedure set_column(c: Byte;const v: Tvector2_extended)`

Visibility: default

Description: Replaces the `c`-th column of the matrix with vector v. The column numbering starts at 0.

64.5.9 Tmatrix2_extended.set_row

Synopsis: Sets r-th row of the matrix with a vector.

Declaration: `procedure set_row(r: Byte;const v: Tvector2_extended)`

Visibility: default

Description: Replaces the `r`-th row of the matrix with vector v. The row numbering starts at 0.

64.5.10 Tmatrix2_extended.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : extended`

Visibility: default

Description: Returns the determinant of the matrix.

64.5.11 Tmatrix2_extended.inverse

Synopsis: Calculates the inverse of the matrix.

Declaration: `function inverse(Adeterminant: extended) : Tmatrix2_extended`

Visibility: default

Description: Tmatrix2_extended.inverse returns a new matrix that is the inverse of the matrix. You must pass the determinant of the matrix as parameter.

64.5.12 Tmatrix2_extended.transpose

Synopsis: Returns the transposition of the matrix.

Declaration: `function transpose : Tmatrix2_extended`

Visibility: default

Description: Tmatrix2_extended.transpose returns a new matrix that is the transposition of the matrix, that is, the matrix with the x and y coordinates of the values swapped.

64.6 Tmatrix2_single

64.6.1 Description

The `Tmatrix2_single` object provides a matrix of 2*2 single precision scalars.

64.6.2 Method overview

Page	Method	Description
496	determinant	Calculates the determinant of the matrix.
495	get_column	Returns the c-th column of the matrix as vector.
495	get_row	Returns the r-th row of the matrix as vector.
495	init	Initializes the matrix, setting its elements to the values passed to the constructor.
495	init_identity	Initializes the matrix and sets its elements to the identity matrix.
495	init_zero	Initializes the matrix and sets its elements to zero
496	inverse	Calculates the inverse of the matrix.
495	set_column	Sets c-th column of the matrix with a vector.
496	set_row	Sets r-th row of the matrix with a vector.
496	transpose	Returns the transposition of the matrix.

64.6.3 Tmatrix2_single.init_zero

Synopsis: Initializes the matrix and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

64.6.4 Tmatrix2_single.init_identity

Synopsis: Initializes the matrix and sets its elements to the identity matrix.

Declaration: `constructor init_identity`

Visibility: default

Description: Initializes the matrix and sets its elements to the identity matrix, that is, elements to 1 on the left-upper to right-lower diagonal, the rest zero.

64.6.5 Tmatrix2_single.init

Synopsis: Initializes the matrix, setting its elements to the values passed to the constructor.

Declaration: `constructor init(aa: single;ab: single;ba: single;bb: single)`

Visibility: default

Description: Initializes the matrix, setting its elements to the values passed to the constructor. The order of the values is left to right, then top to bottom.

64.6.6 Tmatrix2_single.get_column

Synopsis: Returns the c-th column of the matrix as vector.

Declaration: `function get_column(c: Byte) : Tvector2_single`

Visibility: default

Description: Returns the `c`-th column of the matrix as vector. The column numbering starts at 0.

64.6.7 Tmatrix2_single.get_row

Synopsis: Returns the r-th row of the matrix as vector.

Declaration: `function get_row(r: Byte) : Tvector2_single`

Visibility: default

Description: Returns the `r`-th row of the matrix as vector. The row numbering starts at 0.

64.6.8 Tmatrix2_single.set_column

Synopsis: Sets c-th column of the matrix with a vector.

Declaration: `procedure set_column(c: Byte;const v: Tvector2_single)`

Visibility: default

Description: Replaces the `c`-th column of the matrix with vector v. The column numbering starts at 0.

64.6.9 Tmatrix2_single.set_row

Synopsis: Sets r-th row of the matrix with a vector.

Declaration: `procedure set_row(r: Byte;const v: Tvector2_single)`

Visibility: default

Description: Replaces the `r`-th row of the matrix with vector v. The row numbering starts at 0.

64.6.10 Tmatrix2_single.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : single`

Visibility: default

Description: Returns the determinant of the matrix.

64.6.11 Tmatrix2_single.inverse

Synopsis: Calculates the inverse of the matrix.

Declaration: `function inverse(Adeterminant: single) : Tmatrix2_single`

Visibility: default

Description: Tmatrix2_single.inverse returns a new matrix that is the inverse of the matrix. You must pass the determinant of the matrix as parameter.

64.6.12 Tmatrix2_single.transpose

Synopsis: Returns the transposition of the matrix.

Declaration: `function transpose : Tmatrix2_single`

Visibility: default

Description: Tmatrix2_single.transpose returns a new matrix that is the transposition of the matrix, that is, the matrix with the x and y coordinates of the values swapped.

64.7 Tmatrix3_double

64.7.1 Description

The `Tmatrix3_double` object provides a matrix of 3*3 double precision scalars.

64.7.2 Method overview

Page	Method	Description
498	determinant	Calculates the determinant of the matrix.
497	get_column	Returns the c-th column of the matrix as vector.
498	get_row	Returns the r-th row of the matrix as vector.
497	init	Initializes the matrix, setting its elements to the values passed to the constructor.
497	init_identity	Initializes the matrix and sets its elements to the identity matrix.
497	init_zero	Initializes the matrix and sets its elements to zero
498	inverse	Calculates the inverse of the matrix.
498	set_column	Sets c-th column of the matrix with a vector.
498	set_row	Sets r-th row of the matrix with a vector.
498	transpose	Returns the transposition of the matrix.

64.7.3 Tmatrix3_double.init_zero

Synopsis: Initializes the matrix and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

64.7.4 Tmatrix3_double.init_identity

Synopsis: Initializes the matrix and sets its elements to the identity matrix.

Declaration: `constructor init_identity`

Visibility: default

Description: Initializes the matrix and sets its elements to the identity matrix, that is, elements to 1 on the left-upper to right-lower diagonal, the rest zero.

64.7.5 Tmatrix3_double.init

Synopsis: Initializes the matrix, setting its elements to the values passed to the constructor.

Declaration:
```
constructor init(aa: Double;ab: Double;ac: Double;ba: Double;bb: Double;
                 bc: Double;ca: Double;cb: Double;cc: Double)
```

Visibility: default

Description: Initializes the matrix, setting its elements to the values passed to the constructor. The order of the values is left to right, then top to bottom.

64.7.6 Tmatrix3_double.get_column

Synopsis: Returns the c-th column of the matrix as vector.

Declaration: `function get_column(c: Byte) : Tvector3_double`

Visibility: default

Description: Returns the `c`-th column of the matrix as vector. The column numbering starts at 0.

64.7.7 Tmatrix3_double.get_row

Synopsis: Returns the r-th row of the matrix as vector.

Declaration: `function get_row(r: Byte) : Tvector3_double`

Visibility: default

Description: Returns the `r`-th row of the matrix as vector. The row numbering starts at 0.

64.7.8 Tmatrix3_double.set_column

Synopsis: Sets c-th column of the matrix with a vector.

Declaration: `procedure set_column(c: Byte;const v: Tvector3_double)`

Visibility: default

Description: Replaces the `c`-th column of the matrix with vector v. The column numbering starts at 0.

64.7.9 Tmatrix3_double.set_row

Synopsis: Sets r-th row of the matrix with a vector.

Declaration: `procedure set_row(r: Byte;const v: Tvector3_double)`

Visibility: default

Description: Replaces the `r`-th row of the matrix with vector v. The row numbering starts at 0.

64.7.10 Tmatrix3_double.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : Double`

Visibility: default

Description: Returns the determinant of the matrix.

64.7.11 Tmatrix3_double.inverse

Synopsis: Calculates the inverse of the matrix.

Declaration: `function inverse(Adeterminant: Double) : Tmatrix3_double`

Visibility: default

Description: Tmatrix3_double.inverse returns a new matrix that is the inverse of the matrix. You must pass the determinant of the matrix as parameter.

64.7.12 Tmatrix3_double.transpose

Synopsis: Returns the transposition of the matrix.

Declaration: `function transpose : Tmatrix3_double`

Visibility: default

Description: Tmatrix2_double.transpose returns a new matrix that is the transposition of the matrix, that is, the matrix with the x and y coordinates of the values swapped.

64.8 Tmatrix3_extended

64.8.1 Description

The `Tmatrix3_extended` object provides a matrix of 3*3 extended precision scalars.

64.8.2 Method overview

Page	Method	Description
500	determinant	Calculates the determinant of the matrix.
500	get_column	Returns the c-th column of the matrix as vector.
500	get_row	Returns the r-th row of the matrix as vector.
499	init	Initializes the matrix, setting its elements to the values passed to the constructor.
499	init_identity	Initializes the matrix and sets its elements to the identity matrix.
499	init_zero	Initializes the matrix and sets its elements to zero
500	inverse	Calculates the inverse of the matrix.
500	set_column	Sets r-th column of the matrix with a vector.
500	set_row	Sets r-th row of the matrix with a vector.
501	transpose	Returns the transposition of the matrix.

64.8.3 Tmatrix3_extended.init_zero

Synopsis: Initializes the matrix and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

64.8.4 Tmatrix3_extended.init_identity

Synopsis: Initializes the matrix and sets its elements to the identity matrix.

Declaration: `constructor init_identity`

Visibility: default

Description: Initializes the matrix and sets its elements to the identity matrix, that is, elements to 1 on the left-upper to right-lower diagonal, the rest zero.

64.8.5 Tmatrix3_extended.init

Synopsis: Initializes the matrix, setting its elements to the values passed to the constructor.

Declaration:
```
constructor init(aa: extended;ab: extended;ac: extended;ba: extended;
                 bb: extended;bc: extended;ca: extended;cb: extended;
                 cc: extended)
```

Visibility: default

Description: Initializes the matrix, setting its elements to the values passed to the constructor. The order of the values is left to right, then top to bottom.

64.8.6 Tmatrix3_extended.get_column

Synopsis: Returns the c-th column of the matrix as vector.

Declaration: `function get_column(c: Byte) : Tvector3_extended`

Visibility: default

Description: Returns the `c`-th column of the matrix as vector. The column numbering starts at 0.

64.8.7 Tmatrix3_extended.get_row

Synopsis: Returns the r-th row of the matrix as vector.

Declaration: `function get_row(r: Byte) : Tvector3_extended`

Visibility: default

Description: Returns the `r`-th row of the matrix as vector. The row numbering starts at 0.

64.8.8 Tmatrix3_extended.set_column

Synopsis: Sets r-th column of the matrix with a vector.

Declaration: `procedure set_column(c: Byte;const v: Tvector3_extended)`

Visibility: default

Description: Replaces the `c`-th column of the matrix with vector v. The column numbering starts at 0.

64.8.9 Tmatrix3_extended.set_row

Synopsis: Sets r-th row of the matrix with a vector.

Declaration: `procedure set_row(r: Byte;const v: Tvector3_extended)`

Visibility: default

Description: Replaces the `r`-th row of the matrix with vector v. The row numbering starts at 0.

64.8.10 Tmatrix3_extended.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : extended`

Visibility: default

Description: Returns the determinant of the matrix.

64.8.11 Tmatrix3_extended.inverse

Synopsis: Calculates the inverse of the matrix.

Declaration: `function inverse(Adeterminant: extended) : Tmatrix3_extended`

Visibility: default

Description: Tmatrix3_extended.inverse returns a new matrix that is the inverse of the matrix. You must pass the determinant of the matrix as parameter.

64.8.12 Tmatrix3_extended.transpose

Synopsis: Returns the transposition of the matrix.

Declaration: `function transpose : Tmatrix3_extended`

Visibility: default

Description: Tmatrix2_extended.transpose returns a new matrix that is the transposition of the matrix, that is, the matrix with the x and y coordinates of the values swapped.

64.9 Tmatrix3_single

64.9.1 Description

The `Tmatrix3_single` object provides a matrix of 3*3 single precision scalars.

64.9.2 Method overview

Page	Method	Description
502	determinant	Calculates the determinant of the matrix.
502	get_column	Returns the c-th column of the matrix as vector.
502	get_row	Returns the r-th row of the matrix as vector.
502	init	Initializes the matrix, setting its elements to the values passed to the constructor.
501	init_identity	Initializes the matrix and sets its elements to the identity matrix.
501	init_zero	Initializes the matrix and sets its elements to zero
503	inverse	Calculates the inverse of the matrix.
502	set_column	Sets c-th column of the matrix with a vector.
502	set_row	Sets r-th row of the matrix with a vector.
503	transpose	Returns the transposition of the matrix.

64.9.3 Tmatrix3_single.init_zero

Synopsis: Initializes the matrix and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

64.9.4 Tmatrix3_single.init_identity

Synopsis: Initializes the matrix and sets its elements to the identity matrix.

Declaration: `constructor init_identity`

Visibility: default

Description: Initializes the matrix and sets its elements to the identity matrix, that is, elements to 1 on the left-upper to right-lower diagonal, the rest zero.

64.9.5 Tmatrix3_single.init

Synopsis: Initializes the matrix, setting its elements to the values passed to the constructor.

Declaration:
```
constructor init(aa: single;ab: single;ac: single;ba: single;bb: single;
                 bc: single;ca: single;cb: single;cc: single)
```

Visibility: default

Description: Initializes the matrix, setting its elements to the values passed to the constructor. The order of the values is left to right, then top to bottom.

64.9.6 Tmatrix3_single.get_column

Synopsis: Returns the c-th column of the matrix as vector.

Declaration:
```
function get_column(c: Byte) : Tvector3_single
```

Visibility: default

Description: Returns the `c`-th column of the matrix as vector. The column numbering starts at 0.

64.9.7 Tmatrix3_single.get_row

Synopsis: Returns the r-th row of the matrix as vector.

Declaration:
```
function get_row(r: Byte) : Tvector3_single
```

Visibility: default

Description: Returns the `r`-th row of the matrix as vector. The row numbering starts at 0.

64.9.8 Tmatrix3_single.set_column

Synopsis: Sets c-th column of the matrix with a vector.

Declaration:
```
procedure set_column(c: Byte;const v: Tvector3_single)
```

Visibility: default

Description: Replaces the `c`-th column of the matrix with vector v. The column numbering starts at 0.

64.9.9 Tmatrix3_single.set_row

Synopsis: Sets r-th row of the matrix with a vector.

Declaration:
```
procedure set_row(r: Byte;const v: Tvector3_single)
```

Visibility: default

Description: Replaces the `r`-th row of the matrix with vector v. The row numbering starts at 0.

64.9.10 Tmatrix3_single.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration:
```
function determinant : single
```

Visibility: default

Description: Returns the determinant of the matrix.

64.9.11 Tmatrix3_single.inverse

Synopsis: Calculates the inverse of the matrix.

Declaration: `function inverse(Adeterminant: single) : Tmatrix3_single`

Visibility: default

Description: Tmatrix3_single.inverse returns a new matrix that is the inverse of the matrix. You must pass the determinant of the matrix as parameter.

64.9.12 Tmatrix3_single.transpose

Synopsis: Returns the transposition of the matrix.

Declaration: `function transpose : Tmatrix3_single`

Visibility: default

Description: Tmatrix2_single.transpose returns a new matrix that is the transposition of the matrix, that is, the matrix with the x and y coordinates of the values swapped.

64.10 Tmatrix4_double

64.10.1 Description

The `Tmatrix4_double` object provides a matrix of 4*4 double precision scalars.

64.10.2 Method overview

Page	Method	Description
505	determinant	Calculates the determinant of the matrix.
504	get_column	Returns the c-th column of the matrix as vector.
504	get_row	Returns the r-th row of the matrix as vector.
504	init	Initializes the matrix, setting its elements to the values passed to the constructor.
504	init_identity	Initializes the matrix and sets its elements to the identity matrix.
503	init_zero	Initializes the matrix and sets its elements to zero
505	inverse	Calculates the inverse of the matrix.
504	set_column	Sets c-th column of the matrix with a vector.
505	set_row	Sets r-th row of the matrix with a vector.
505	transpose	Returns the transposition of the matrix.

64.10.3 Tmatrix4_double.init_zero

Synopsis: Initializes the matrix and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

64.10.4 Tmatrix4_double.init_identity

Synopsis: Initializes the matrix and sets its elements to the identity matrix.

Declaration:
```
constructor init_identity
```

Visibility: default

Description: Initializes the matrix and sets its elements to the identity matrix, that is, elements to 1 on the left-upper to right-lower diagonal, the rest zero.

64.10.5 Tmatrix4_double.init

Synopsis: Initializes the matrix, setting its elements to the values passed to the constructor.

Declaration:
```
constructor init(aa: Double;ab: Double;ac: Double;ad: Double;ba: Double;
                 bb: Double;bc: Double;bd: Double;ca: Double;cb: Double;
                 cc: Double;cd: Double;da: Double;db: Double;dc: Double;
                 dd: Double)
```

Visibility: default

Description: Initializes the matrix, setting its elements to the values passed to the constructor. The order of the values is left to right, then top to bottom.

64.10.6 Tmatrix4_double.get_column

Synopsis: Returns the c-th column of the matrix as vector.

Declaration:
```
function get_column(c: Byte) : Tvector4_double
```

Visibility: default

Description: Returns the `c`-th column of the matrix as vector. The column numbering starts at 0.

64.10.7 Tmatrix4_double.get_row

Synopsis: Returns the r-th row of the matrix as vector.

Declaration:
```
function get_row(r: Byte) : Tvector4_double
```

Visibility: default

Description: Returns the `r`-th row of the matrix as vector. The row numbering starts at 0.

64.10.8 Tmatrix4_double.set_column

Synopsis: Sets c-th column of the matrix with a vector.

Declaration:
```
procedure set_column(c: Byte;const v: Tvector4_double)
```

Visibility: default

Description: Replaces the `c`-th column of the matrix with vector v. The column numbering starts at 0.

64.10.9 Tmatrix4_double.set_row

Synopsis: Sets r-th row of the matrix with a vector.

Declaration: `procedure set_row(r: Byte;const v: Tvector4_double)`

Visibility: default

Description: Replaces the `r`-th row of the matrix with vector v. The row numbering starts at 0.

64.10.10 Tmatrix4_double.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : Double`

Visibility: default

Description: Returns the determinant of the matrix. Note: Calculating the determinant of a 4*4 matrix requires quite a few operations.

64.10.11 Tmatrix4_double.inverse

Synopsis: Calculates the inverse of the matrix.

Declaration: `function inverse(Adeterminant: Double) : Tmatrix4_double`

Visibility: default

Description: Tmatrix4_double.inverse returns a new matrix that is the inverse of the matrix. You must pass the determinant of the matrix as parameter. Note: Calculating the inverse of a 4*4 matrix requires quite a few operations.

64.10.12 Tmatrix4_double.transpose

Synopsis: Returns the transposition of the matrix.

Declaration: `function transpose : Tmatrix4_double`

Visibility: default

Description: Tmatrix2_double.transpose returns a new matrix that is the transposition of the matrix, that is, the matrix with the x and y coordinates of the values swapped.

64.11 Tmatrix4_extended

64.11.1 Description

The `Tmatrix4_extended` object provides a matrix of 4*4 extended precision scalars.

64.11.2 Method overview

Page	Method	Description
507	determinant	Calculates the determinant of the matrix.
506	get_column	Returns the c-th column of the matrix as vector.
507	get_row	Returns the r-th row of the matrix as vector.
506	init	Initializes the matrix, setting its elements to the values passed to the constructor.
506	init_identity	Initializes the matrix and sets its elements to the identity matrix.
506	init_zero	Initializes the matrix and sets its elements to zero
507	inverse	Calculates the inverse of the matrix.
507	set_column	Sets c-th column of the matrix with a vector.
507	set_row	Sets r-th row of the matrix with a vector.
508	transpose	Returns the transposition of the matrix.

64.11.3 Tmatrix4_extended.init_zero

Synopsis: Initializes the matrix and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

64.11.4 Tmatrix4_extended.init_identity

Synopsis: Initializes the matrix and sets its elements to the identity matrix.

Declaration: `constructor init_identity`

Visibility: default

Description: Initializes the matrix and sets its elements to the identity matrix, that is, elements to 1 on the left-upper to right-lower diagonal, the rest zero.

64.11.5 Tmatrix4_extended.init

Synopsis: Initializes the matrix, setting its elements to the values passed to the constructor.

Declaration:
```
constructor init(aa: extended;ab: extended;ac: extended;ad: extended;
                 ba: extended;bb: extended;bc: extended;bd: extended;
                 ca: extended;cb: extended;cc: extended;cd: extended;
                 da: extended;db: extended;dc: extended;dd: extended)
```

Visibility: default

Description: Initializes the matrix, setting its elements to the values passed to the constructor. The order of the values is left to right, then top to bottom.

64.11.6 Tmatrix4_extended.get_column

Synopsis: Returns the c-th column of the matrix as vector.

Declaration: `function get_column(c: Byte) : Tvector4_extended`

Visibility: default

Description: Returns the `c`-th column of the matrix as vector. The column numbering starts at 0.

64.11.7 Tmatrix4_extended.get_row

Synopsis: Returns the r-th row of the matrix as vector.

Declaration: `function get_row(r: Byte) : Tvector4_extended`

Visibility: default

Description: Returns the `r`-th row of the matrix as vector. The row numbering starts at 0.

64.11.8 Tmatrix4_extended.set_column

Synopsis: Sets c-th column of the matrix with a vector.

Declaration: `procedure set_column(c: Byte;const v: Tvector4_extended)`

Visibility: default

Description: Replaces the `c`-th column of the matrix with vector v. The column numbering starts at 0.

64.11.9 Tmatrix4_extended.set_row

Synopsis: Sets r-th row of the matrix with a vector.

Declaration: `procedure set_row(r: Byte;const v: Tvector4_extended)`

Visibility: default

Description: Replaces the `r`-th row of the matrix with vector v. The row numbering starts at 0.

64.11.10 Tmatrix4_extended.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : extended`

Visibility: default

Description: Returns the determinant of the matrix. Note: Calculating the determinant of a 4*4 matrix requires quite a few operations.

64.11.11 Tmatrix4_extended.inverse

Synopsis: Calculates the inverse of the matrix.

Declaration: `function inverse(Adeterminant: extended) : Tmatrix4_extended`

Visibility: default

Description: Tmatrix4_extended.inverse returns a new matrix that is the inverse of the matrix. You must pass the determinant of the matrix as parameter. Note: Calculating the inverse of a 4*4 matrix requires quite a few operations.

64.11.12 Tmatrix4_extended.transpose

Synopsis: Returns the transposition of the matrix.

Declaration: `function transpose : Tmatrix4_extended`

Visibility: default

Description: Tmatrix2_extended.transpose returns a new matrix that is the transposition of the matrix, that is, the matrix with the x and y coordinates of the values swapped.

64.12 Tmatrix4_single

64.12.1 Description

The `Tmatrix4_single` object provides a matrix of 4*4 single precision scalars.

64.12.2 Method overview

Page	Method	Description
510	determinant	Calculates the determinant of the matrix.
509	get_column	Returns the c-th column of the matrix as vector.
509	get_row	Returns the r-th row of the matrix as vector.
509	init	Initializes the matrix, setting its elements to the values passed to the constructor.
508	init_identity	Initializes the matrix and sets its elements to the identity matrix.
508	init_zero	Initializes the matrix and sets its elements to zero
510	inverse	Calculates the inverse of the matrix.
509	set_column	Sets c-th column of the matrix with a vector.
509	set_row	Sets r-th row of the matrix with a vector.
510	transpose	Returns the transposition of the matrix.

64.12.3 Tmatrix4_single.init_zero

Synopsis: Initializes the matrix and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

64.12.4 Tmatrix4_single.init_identity

Synopsis: Initializes the matrix and sets its elements to the identity matrix.

Declaration: `constructor init_identity`

Visibility: default

Description: Initializes the matrix and sets its elements to the identity matrix, that is, elements to 1 on the left-upper to right-lower diagonal, the rest zero.

64.12.5 Tmatrix4_single.init

Synopsis: Initializes the matrix, setting its elements to the values passed to the constructor.

Declaration:
```
constructor init(aa: single;ab: single;ac: single;ad: single;ba: single;
                 bb: single;bc: single;bd: single;ca: single;cb: single;
                 cc: single;cd: single;da: single;db: single;dc: single;
                 dd: single)
```

Visibility: default

Description: Initializes the matrix, setting its elements to the values passed to the constructor. The order of the values is left to right, then top to bottom.

64.12.6 Tmatrix4_single.get_column

Synopsis: Returns the c-th column of the matrix as vector.

Declaration:
```
function get_column(c: Byte) : Tvector4_single
```

Visibility: default

Description: Returns the `c`-th column of the matrix as vector. The column numbering starts at 0.

64.12.7 Tmatrix4_single.get_row

Synopsis: Returns the r-th row of the matrix as vector.

Declaration:
```
function get_row(r: Byte) : Tvector4_single
```

Visibility: default

Description: Returns the `r`-th row of the matrix as vector. The row numbering starts at 0.

64.12.8 Tmatrix4_single.set_column

Synopsis: Sets c-th column of the matrix with a vector.

Declaration:
```
procedure set_column(c: Byte;const v: Tvector4_single)
```

Visibility: default

Description: Replaces the `c`-th column of the matrix with vector v. The column numbering starts at 0.

64.12.9 Tmatrix4_single.set_row

Synopsis: Sets r-th row of the matrix with a vector.

Declaration:
```
procedure set_row(r: Byte;const v: Tvector4_single)
```

Visibility: default

Description: Replaces the `r`-th row of the matrix with vector v. The row numbering starts at 0.

64.12.10 Tmatrix4_single.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : single`

Visibility: default

Description: Returns the determinant of the matrix. Note: Calculating the determinant of a 4*4 matrix requires quite a few operations.

64.12.11 Tmatrix4_single.inverse

Synopsis: Calculates the inverse of the matrix.

Declaration: `function inverse(Adeterminant: single) : Tmatrix4_single`

Visibility: default

Description: Tmatrix4_single.inverse returns a new matrix that is the inverse of the matrix. You must pass the determinant of the matrix as parameter. Note: Calculating the inverse of a 4*4 matrix requires quite a few operations.

64.12.12 Tmatrix4_single.transpose

Synopsis: Returns the transposition of the matrix.

Declaration: `function transpose : Tmatrix4_single`

Visibility: default

Description: Tmatrix2_single.transpose returns a new matrix that is the transposition of the matrix, that is, the matrix with the x and y coordinates of the values swapped.

64.13 Tvector2_double

64.13.1 Description

The `Tvector2_double` object provides a vector of two double precision scalars.

64.13.2 Method overview

Page	Method	Description
511	init	Initializes the vector, setting its elements to the values passed to the constructor.
511	init_one	Initializes the vector and sets its elements to one
510	init_zero	Initializes the vector and sets its elements to zero
511	length	Calculates the length of the vector.
511	squared_length	Calculates the squared length of the vector.

64.13.3 Tvector2_double.init_zero

Synopsis: Initializes the vector and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

64.13.4 Tvector2_double.init_one

Synopsis: Initializes the vector and sets its elements to one

Declaration: `constructor init_one`

Visibility: default

64.13.5 Tvector2_double.init

Synopsis: Initializes the vector, setting its elements to the values passed to the constructor.

Declaration: `constructor init(a: Double;b: Double)`

Visibility: default

64.13.6 Tvector2_double.length

Synopsis: Calculates the length of the vector.

Declaration: `function length : Double`

Visibility: default

Description: Calculate the length of the vector: length=sqrt(data[0]**2+data[1]**2). Try to use squared_length (511) if you are able to, as it is faster.

64.13.7 Tvector2_double.squared_length

Synopsis: Calculates the squared length of the vector.

Declaration: `function squared_length : Double`

Visibility: default

Description: Calculate the squared length of the vector: squared_length=data[0]**2+data[1]**2.

64.14 Tvector2_extended

64.14.1 Description

The `Tvector2_extended` object provides a vector of two extended precision scalars.

64.14.2 Method overview

Page	Method	Description
512	init	Initializes the vector, setting its elements to the values passed to the constructor.
512	init_one	Initializes the vector and sets its elements to one
512	init_zero	Initializes the vector and sets its elements to zero
512	length	Calculates the length of the vector.
512	squared_length	Calculates the squared length of the vector.

64.14.3 Tvector2_extended.init_zero

Synopsis: Initializes the vector and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

64.14.4 Tvector2_extended.init_one

Synopsis: Initializes the vector and sets its elements to one

Declaration: `constructor init_one`

Visibility: default

64.14.5 Tvector2_extended.init

Synopsis: Initializes the vector, setting its elements to the values passed to the constructor.

Declaration: `constructor init(a: extended;b: extended)`

Visibility: default

64.14.6 Tvector2_extended.length

Synopsis: Calculates the length of the vector.

Declaration: `function length : extended`

Visibility: default

Description: Calculate the length of the vector: length=sqrt(data[0]**2+data[1]**2). Try to use squared_length (512) if you are able to, as it is faster.

64.14.7 Tvector2_extended.squared_length

Synopsis: Calculates the squared length of the vector.

Declaration: `function squared_length : extended`

Visibility: default

Description: Calculate the squared length of the vector: squared_length=data[0]**2+data[1]**2.

64.15 Tvector2_single

64.15.1 Description

The `Tvector2_single` object provides a vector of two single precision scalars.

64.15.2 Method overview

Page	Method	Description
513	init	Initializes the vector, setting its elements to the values passed to the constructor.
513	init_one	Initializes the vector and sets its elements to one
513	init_zero	Initializes the vector and sets its elements to zero
513	length	Calculates the length of the vector.
513	squared_length	Calculates the squared length of the vector.

64.15.3 Tvector2_single.init_zero

Synopsis: Initializes the vector and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

64.15.4 Tvector2_single.init_one

Synopsis: Initializes the vector and sets its elements to one

Declaration: `constructor init_one`

Visibility: default

64.15.5 Tvector2_single.init

Synopsis: Initializes the vector, setting its elements to the values passed to the constructor.

Declaration: `constructor init(a: single;b: single)`

Visibility: default

64.15.6 Tvector2_single.length

Synopsis: Calculates the length of the vector.

Declaration: `function length : single`

Visibility: default

Description: Calculate the length of the vector: length=sqrt(data[0]**2+data[1]**2). Try to use squared_length (513) if you are able to, as it is faster.

64.15.7 Tvector2_single.squared_length

Synopsis: Calculates the squared length of the vector.

Declaration: `function squared_length : single`

Visibility: default

Description: Calculate the squared length of the vector: squared_length=data[0]**2+data[1]**2.

64.16 Tvector3_double

64.16.1 Description

The `Tvector3_double` object provides a vector of three double precision scalars.

64.16.2 Method overview

Page	Method	Description
514	init	Initializes the vector, setting its elements to the values passed to the constructor.
514	init_one	Initializes the vector and sets its elements to one
514	init_zero	Initializes the vector and sets its elements to zero
514	length	Calculates the length of the vector.
515	squared_length	Calculates the squared length of the vector.

64.16.3 Tvector3_double.init_zero

Synopsis: Initializes the vector and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

64.16.4 Tvector3_double.init_one

Synopsis: Initializes the vector and sets its elements to one

Declaration: `constructor init_one`

Visibility: default

64.16.5 Tvector3_double.init

Synopsis: Initializes the vector, setting its elements to the values passed to the constructor.

Declaration: `constructor init(a: Double;b: Double;c: Double)`

Visibility: default

64.16.6 Tvector3_double.length

Synopsis: Calculates the length of the vector.

Declaration: `function length : Double`

Visibility: default

Description: Calculate the length of the vector: length=sqrt(data[0]**2+data[1]**2+data[2]**2). Try to use squared_length (515) if you are able to, as it is faster.

64.16.7 Tvector3_double.squared_length

Synopsis: Calculates the squared length of the vector.

Declaration: `function squared_length : Double`

Visibility: default

Description: Calculate the squared length of the vector: squared_length=data[0]**2+data[1]**2+data[2]**2.

64.17 Tvector3_extended

64.17.1 Description

The `Tvector3_extended` object provides a vector of three extended precision scalars.

64.17.2 Method overview

Page	Method	Description
515	init	Initializes the vector, setting its elements to the values passed to the constructor.
515	init_one	Initializes the vector and sets its elements to one
515	init_zero	Initializes the vector and sets its elements to zero
516	length	Calculates the length of the vector.
516	squared_length	Calculates the squared length of the vector.

64.17.3 Tvector3_extended.init_zero

Synopsis: Initializes the vector and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

64.17.4 Tvector3_extended.init_one

Synopsis: Initializes the vector and sets its elements to one

Declaration: `constructor init_one`

Visibility: default

64.17.5 Tvector3_extended.init

Synopsis: Initializes the vector, setting its elements to the values passed to the constructor.

Declaration: `constructor init(a: extended;b: extended;c: extended)`

Visibility: default

64.17.6 Tvector3_extended.length

Synopsis: Calculates the length of the vector.

Declaration: `function length : extended`

Visibility: default

Description: Calculate the length of the vector: length=sqrt(data[0]**2+data[1]**2+data[2]**2). Try to use squared_length (516) if you are able to, as it is faster.

64.17.7 Tvector3_extended.squared_length

Synopsis: Calculates the squared length of the vector.

Declaration: `function squared_length : extended`

Visibility: default

Description: Calculate the squared length of the vector: squared_length=data[0]**2+data[1]**2+data[2]**2.

64.18 Tvector3_single

64.18.1 Description

The `Tvector3_single` object provides a vector of three single precision scalars.

64.18.2 Method overview

Page	Method	Description
517	init	Initializes the vector, setting its elements to the values passed to the constructor.
516	init_one	Initializes the vector and sets its elements to one
516	init_zero	Initializes the vector and sets its elements to zero
517	length	Calculates the length of the vector.
517	squared_length	Calculates the squared length of the vector.

64.18.3 Tvector3_single.init_zero

Synopsis: Initializes the vector and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

64.18.4 Tvector3_single.init_one

Synopsis: Initializes the vector and sets its elements to one

Declaration: `constructor init_one`

Visibility: default

64.18.5 Tvector3_single.init

Synopsis: Initializes the vector, setting its elements to the values passed to the constructor.

Declaration: `constructor init(a: single;b: single;c: single)`

Visibility: default

64.18.6 Tvector3_single.length

Synopsis: Calculates the length of the vector.

Declaration: `function length : single`

Visibility: default

Description: Calculate the length of the vector: length=sqrt(data[0]**2+data[1]**2+data[2]**2). Try to use squared_length (517) if you are able to, as it is faster.

64.18.7 Tvector3_single.squared_length

Synopsis: Calculates the squared length of the vector.

Declaration: `function squared_length : single`

Visibility: default

Description: Calculate the squared length of the vector: squared_length=data[0]**2+data[1]**2+data[2]**2.

64.19 Tvector4_double

64.19.1 Description

The `Tvector4_double` object provides a vector of four double precision scalars.

64.19.2 Method overview

Page	Method	Description
518	init	Initializes the vector, setting its elements to the values passed to the constructor.
518	init_one	Initializes the vector and sets its elements to one
517	init_zero	Initializes the vector and sets its elements to zero
518	length	Calculates the length of the vector.
518	squared_length	Calculates the squared length of the vector.

64.19.3 Tvector4_double.init_zero

Synopsis: Initializes the vector and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

64.19.4 Tvector4_double.init_one

Synopsis: Initializes the vector and sets its elements to one

Declaration: `constructor init_one`

Visibility: default

64.19.5 Tvector4_double.init

Synopsis: Initializes the vector, setting its elements to the values passed to the constructor.

Declaration: `constructor init(a: Double;b: Double;c: Double;d: Double)`

Visibility: default

64.19.6 Tvector4_double.length

Synopsis: Calculates the length of the vector.

Declaration: `function length : Double`

Visibility: default

Description: Calculate the length of the vector: length=sqrt(data[0]**2+data[1]**2+data[2]**2+data[3]**2). Try to use squared_length (518) if you are able to, as it is faster.

64.19.7 Tvector4_double.squared_length

Synopsis: Calculates the squared length of the vector.

Declaration: `function squared_length : Double`

Visibility: default

Description: Calculate the squared length of the vector: squared_length=data[0]**2+data[1]**2+data[2]**2+data[3]**2.

64.20 Tvector4_extended

64.20.1 Description

The `Tvector4_extended` object provides a vector of four extended precision scalars.

64.20.2 Method overview

Page	Method	Description
519	init	Initializes the vector, setting its elements to the values passed to the constructor.
519	init_one	Initializes the vector and sets its elements to one
519	init_zero	Initializes the vector and sets its elements to zero
519	length	Calculates the length of the vector.
519	squared_length	Calculates the squared length of the vector.

64.20.3 Tvector4_extended.init_zero

Synopsis: Initializes the vector and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

64.20.4 Tvector4_extended.init_one

Synopsis: Initializes the vector and sets its elements to one

Declaration: `constructor init_one`

Visibility: default

64.20.5 Tvector4_extended.init

Synopsis: Initializes the vector, setting its elements to the values passed to the constructor.

Declaration: `constructor init(a: extended;b: extended;c: extended;d: extended)`

Visibility: default

64.20.6 Tvector4_extended.length

Synopsis: Calculates the length of the vector.

Declaration: `function length : extended`

Visibility: default

Description: Calculate the length of the vector: length=sqrt(data[0]**2+data[1]**2+data[2]**2+data[3]**2). Try to use squared_length (519) if you are able to, as it is faster.

64.20.7 Tvector4_extended.squared_length

Synopsis: Calculates the squared length of the vector.

Declaration: `function squared_length : extended`

Visibility: default

Description: Calculate the squared length of the vector: squared_length=data[0]**2+data[1]**2+data[2]**2+data[3]**2.

64.21 Tvector4_single

64.21.1 Description

The `Tvector4_single` object provides a vector of four single precision scalars.

64.21.2 Method overview

Page	Method	Description
520	init	Initializes the vector, setting its elements to the values passed to the constructor.
520	init_one	Initializes the vector and sets its elements to one
520	init_zero	Initializes the vector and sets its elements to zero
520	length	Calculates the length of the vector.
520	squared_length	Calculates the squared length of the vector.

64.21.3 Tvector4_single.init_zero

Synopsis: Initializes the vector and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

64.21.4 Tvector4_single.init_one

Synopsis: Initializes the vector and sets its elements to one

Declaration: `constructor init_one`

Visibility: default

64.21.5 Tvector4_single.init

Synopsis: Initializes the vector, setting its elements to the values passed to the constructor.

Declaration: `constructor init(a: single;b: single;c: single;d: single)`

Visibility: default

64.21.6 Tvector4_single.length

Synopsis: Calculates the length of the vector.

Declaration: `function length : single`

Visibility: default

Description: Calculate the length of the vector: length=sqrt(data[0]**2+data[1]**2+data[2]**2+data[3]**2). Try to use squared_length (520) if you are able to, as it is faster.

64.21.7 Tvector4_single.squared_length

Synopsis: Calculates the squared length of the vector.

Declaration: `function squared_length : single`

Visibility: default

Description: Calculate the squared length of the vector: squared_length=data[0]**2+data[1]**2+data[2]**2+data[3]**2.

Chapter 65

Reference for unit 'mmx'

65.1 Overview

This document describes the MMX unit. This unit allows you to use the MMX capabilities of the Free Pascal compiler. It was written by Florian Klaempfl for the I386 processor. It should work on all platforms that use the Intel processor.

65.2 Constants, types and variables

65.2.1 Constants

```
is_amd_3d_cpu : Boolean = False
```

The is_amd_3d_cpu initialized constant allows you to determine if the computer has the AMD 3D extensions. It is set correctly in the unit's initialization code.

```
is_amd_3d_dsp_cpu : Boolean = False
```

The is_amd_3d_dsp_cpu initialized constant allows you to determine if the computer has the AMD 3D DSP extensions. It is set correctly in the unit's initialization code.

```
is_amd_3d_mmx_cpu : Boolean = False
```

The is_amd_3d_mmx_cpu initialized constant allows you to determine if the computer has the AMD 3D MMX extensions. It is set correctly in the unit's initialization code.

```
is_mmx_cpu : Boolean = False
```

The is_mmx_cpu initialized constant allows you to determine if the computer has MMX extensions. It is set correctly in the unit's initialization code.

```
is_sse2_cpu : Boolean = False
```

The is_sse2_cpu initialized constant allows you to determine if the computer has the SSE2 extensions. It is set correctly in the unit's initialization code.

```
is_sse_cpu : Boolean = False
```

The is_sse_cpu initialized constant allows you to determine if the computer has the SSE extensions. It is set correctly in the unit's initialization code.

65.2.2 Types

```
pmmxbyte = ^tmmxbyte
```

Pointer to tmmxbyte (522) array type

```
pmmxcardinal = ^tmmxcardinal
```

Pointer to tmmxcardinal (522) array type

```
pmmxinteger = ^tmmxinteger
```

Pointer to tmmxinteger (522) array type

```
pmmxlongint = ^tmmxlongint
```

Pointer to tmmxlongint (522) array type

```
pmmxshortint = ^tmmxshortint
```

Pointer to tmmxshortint (522) array type

```
pmmxsingle = ^tmmxsingle
```

Pointer to tmmxsingle (522) array type

```
pmmxword = ^tmmxword
```

Pointer to tmmxword (522) array type

```
tmmxbyte = Array[0..7] of Byte
```

Array of bytes, 64 bits in size

```
tmmxcardinal = Array[0..1] of Cardinal
```

Array of cardinals, 64 bits in size

```
tmmxinteger = Array[0..3] of Integer
```

Array of integers, 64 bits in size

```
tmmxlongint = Array[0..1] of LongInt
```

Array of longint, 64 bits in size

```
tmmxshortint = Array[0..7] of ShortInt
```

Array of shortints, 64 bits in size

```
tmmxsingle = Array[0..1] of single
```

Array of singles, 64 bits in size

```
tmmxword = Array[0..3] of Word
```

Array of words, 64 bits in size

65.3 Procedures and functions

65.3.1 emms

Synopsis: Reset floating point registers

Declaration: `procedure emms`

Visibility: default

Description: `Emms` sets all floating point registers to empty. This procedure must be called after you have used any `MMX` instructions, if you want to use floating point arithmetic. If you just want to move floating point data around, it isn't necessary to call this function, the compiler doesn't use the FPU registers when moving data. Only when doing calculations, you should use this function. The following code demonstrates this:

```
Program MMXDemo;
uses mmx;
var
   d1 : double;
   a : array[0..10000] of double;
   i : longint;
begin
   d1:=1.0;
{$mmx+}
   { floating point data is used, but we do _no_ arithmetic }
   for i:=0 to 10000 do
     a[i]:=d2;  { this is done with 64 bit moves }
{$mmx-}
   emms;   { clear fpu }
   { now we can do floating point arithmetic again }
end.
```

See also: femms (523)

65.3.2 femms

Synopsis: Reset floating point registers - AMD version

Declaration: `procedure femms`

Visibility: default

Description: `femms` executes the `femms` assembler instruction for AMD processors. it is not supported by all assemblers, hence it is coded as byte codes.

See also: emms (523)

Chapter 66

Reference for unit 'Mouse'

66.1 Overview

The Mouse unit implements a platform independent mouse handling interface. It is implemented identically on all platforms supported by Free Pascal and can be enhanced with custom drivers, should this be needed. It is intended to be used only in text-based screens, for instance in conjunction with the keyboard and video unit. No support for graphical screens is implemented, and there are (currently) no plans to implement this.

66.2 Writing a custom mouse driver

The mouse unit has support for adding a custom mouse driver. This can be used to add support for mouses not supported by the standard Free Pascal driver, but also to enhance an existing driver for instance to log mouse events or to implement a record and playback function.

The following unit shows how a mouse driver can be enhanced by adding some logging capabilities to the driver.

Listing: ./mouseex/logmouse.pp

```
unit logmouse;

interface

Procedure StartMouseLogging;
Procedure StopMouseLogging;
Function  IsMouseLogging : Boolean;
Procedure SetMouseLogFileName(FileName : String);


implementation

uses sysutils,Mouse;

var
  NewMouseDriver,
  OldMouseDriver : TMouseDriver;
  Active,Logging : Boolean;
  LogFileName : String;
  MouseLog : Text;
```

```
Function TimeStamp : String;

begin
  TimeStamp:=FormatDateTime('hh:nn:ss',Time());
end;

Procedure StartMouseLogging;

begin
  Logging:=True;
  Writeln(MouseLog,'Start logging mouse events at: ',TimeStamp);
end;

Procedure StopMouseLogging;

begin
  Writeln(MouseLog,'Stop logging mouse events at: ',TimeStamp);
  Logging:=False;
end;

Function IsMouseLogging : Boolean;

begin
  IsMouseLogging:=Logging;
end;

Procedure LogGetMouseEvent(Var Event : TMouseEvent);

Var
  M : TMouseEvent;

begin
  OldMouseDriver.GetMouseEvent(M);
  If Logging then
    begin
    Write(MouseLog,TimeStamp,': Mouse ');
    With M do
      begin
      Case Action of
        MouseActionDown : Write(MouseLog,'down');
        MouseActionUp   : Write(MouseLog,'up');
        MouseActionMove : Write(MouseLog,'move');
      end;
      Write(MouseLog,' event at ',X,',',Y);
      If (Buttons<>0) then
        begin
        Write(MouseLog,' for buttons: ');
        If (Buttons and MouseLeftbutton)<>0 then
          Write(MouseLog,'Left ');
        If (Buttons and MouseRightbutton)<>0 then
          Write(MouseLog,'Right ');
        If (Buttons and MouseMiddlebutton)<>0 then
          Write(MouseLog,'Middle ');
        end;
      Writeln(MouseLog);
      end;
    end;
```

```
end;

Procedure LogInitMouse;

begin
  OldMouseDriver.InitDriver();
  Assign(MouseLog,logFileName);
  Rewrite(MouseLog);
  Active:=True;
  StartMouseLogging;
end;

Procedure LogDoneMouse;

begin
  StopMouseLogging;
  Close(MouseLog);
  Active:=False;
  OldMouseDriver.DoneDriver();
end;

Procedure SetMouseLogFileName(FileName : String);

begin
  If Not Active then
    LogFileName:=FileName;
end;

Initialization
  GetMouseDriver(OldMouseDriver);
  NewMouseDriver:=OldMouseDriver;
  NewMouseDriver.GetMouseEvent:=@LogGetMouseEvent;
  NewMouseDriver.InitDriver:=@LogInitMouse;
  NewMouseDriver.DoneDriver:=@LogDoneMouse;
  LogFileName:='Mouse.log';
  Logging:=False;
  SetMouseDriver(NewMouseDriver);
end.
```

Chapter 67

Reference for unit 'Objects'

67.1 Overview

This document documents the objects unit. The unit was implemented by many people, and was mainly taken from the FreeVision sources. It has been ported to all supported platforms.

The methods and fields that are in a `Private` part of an object declaration have been left out of this documentation.

67.2 Constants, types and variables

67.2.1 Constants

```
coIndexError = -1
```

Collection list error: Index out of range

```
coOverflow = -2
```

Collection list error: Overflow

```
DefaultTPCompatible : Boolean = False
```

`DefaultTPCompatible` is used to initialize tstream.tpcompatible (**??**).

```
MaxBytes = 128 * 1024 * 128
```

Maximum data size (in bytes)

```
MaxCollectionSize = MaxBytes div (Pointer)
```

Maximum collection size (in items)

```
MaxPtrs = MaxBytes div (Pointer)
```

Maximum data size (in pointers)

```
MaxReadBytes = $7fffffff
```

Maximum data that can be read from a stream (not used)

```
MaxTPCompatibleCollectionSize = 65520 div 4
```

Maximum collection size (in items, same value as in TP)

```
MaxWords = MaxBytes div (Word)
```

Maximum data size (in words)

```
RCollection : TStreamRec = (ObjType: 50; VmtLink: (^(TCollection)); Load: @TCollecti
```

Default stream record for the TCollection (544) object.

```
RStrCollection : TStreamRec = (ObjType: 69; VmtLink: (^(TStrCollection)); Load: @TSt
```

Default stream record for the TStrCollection (583) object.

```
RStringCollection : TStreamRec = (ObjType: 51; VmtLink: (^(TStringCollection)); Load
```

Default stream record for the TStringCollection (593) object.

```
RStringList : TStreamRec = (ObjType: 52; VmtLink: (^(TStringList)); Load: @TStringLi
```

Default stream record for the TStringList (595) object.

```
RStrListMaker : TStreamRec = (ObjType: 52; VmtLink: (^(TStrListMaker)); Load: Nil; S
```

Default stream record for the TStrListMaker (597) object.

```
stCreate = $3C00
```

Stream initialization mode: Create new file

```
stError = -1
```

Stream error codes: Access error

```
stGetError = -5
```

Stream error codes: Get object error

```
stInitError = -2
```

Stream error codes: Initialize error

```
stOk = 0
```

Stream error codes: No error

```
stOpen = $3D02
```

Stream initialization mode: Read/write access

```
stOpenError = -8
```

Stream error codes: Error opening stream

```
stOpenRead = $3D00
```

Stream initialization mode: Read access only

```
stOpenWrite = $3D01
```

Stream initialization mode: Write access only

```
stPutError = -6
```

Stream error codes: Put object error

```
stReadError = -3
```

Stream error codes: Stream read error

```
StreamError : CodePointer = Nil
```

Pointer to default stream error handler.

```
stSeekError = -7
```

Stream error codes: Seek error in stream

```
stWriteError = -4
```

Stream error codes: Stream write error

```
vmtHeaderSize = 8
```

Size of the VMT header in an object (not used).

67.2.2 Types

```
AsciiZ = Array[0..255] of Char
```

Filename - null terminated array of characters.

```
FNameStr = String
```

Filename - shortstring version.

```
LongRec = packed record
  Hi : Word;
  Lo : Word;
end
```

Record describing a longint (in Words)

```
PBufStream = ^TBufStream
```

Pointer to TBufStream (540) object.

```
PByteArray = ^TByteArray
```

Pointer to TByteArray (532)

```
PCharSet = ^TCharSet
```

Pointer to TCharSet (532).

```
PCollection = ^TCollection
```

Pointer to TCollection (544) object.

```
PDosStream = ^TDosStream
```

Pointer to TDosStream (558) object.

```
PItemList = ^TItemList
```

Pointer to TItemList (532) object.

```
PMemoryStream = ^TMemoryStream
```

Pointer to TMemoryStream (563) object.

```
PObject = ^TObject
```

Pointer to TObject (565) object.

```
PPoint = ^TPoint
```

Pointer to TPoint (567) record.

```
PPointerArray = ^TPointerArray
```

Pointer to TPointerArray (532)

```
PRect = ^TRect
```

Pointer to TRect (567) object.

```
PResourceCollection = ^TResourceCollection
```

Pointer to TResourceCollection (573) object.

```
PResourceFile = ^TResourceFile
```

Pointer to TResourceFile (574) object.

```
PSortedCollection = ^TSortedCollection
```

Pointer to TSortedCollection (577) object.

```
PStrCollection = ^TStrCollection
```

Pointer to TStrCollection (583) object.

```
PStream = ^TStream
```

Pointer type to TStream (585)

```
PStreamRec = ^TStreamRec
```

Pointer to TStreamRec (532)

```
PStrIndex = ^TStrIndex
```

Pointer to TStrIndex (532) array.

```
PString = PShortString
```

Pointer to a shortstring.

```
PStringCollection = ^TStringCollection
```

Pointer to TStringCollection (593) object.

```
PStringList = ^TStringList
```

Pointer to TStringList (595) object.

```
PStrListMaker = ^TStrListMaker
```

Pointer to TStrListMaker (597) object.

```
PtrRec = packed record
  Ofs : Word;
  Seg : Word;
end
```

Record describing a pointer to a memory location.

```
PUnSortedStrCollection = ^TUnSortedStrCollection
```

Pointer to TUnSortedStrCollection (598) object.

```
PWordArray = ^TWordArray
```

Pointer to TWordArray (533)

```
Sw_Integer = LongInt
```

Alias for longint

```
Sw_Word = Cardinal
```

Alias for Cardinal

```
TByteArray = Array[0..MaxBytes-1] of Byte
```

Array with maxmimum allowed number of bytes.

```
TCharSet = Set of Char
```

Generic set of characters type.

```
TItemList = Array[0..MaxCollectionSize-1] of Pointer
```

Pointer array type used in a TCollection (544)

```
TPointerArray = Array[0..MaxPtrs-1] of Pointer
```

Array with maxmimum allowed number of pointers

```
TStreamRec = packed record
  ObjType : Sw_Word;
  VmtLink : pointer;
  Load : CodePointer;
  Store : CodePointer;
  Next : PStreamRec;
end
```

`TSreamRec` is used by the Objects unit streaming mechanism: when an object is registered, a `TStreamRec` record is added to a list of records. This list is used when objects need to be streamed from/streamed to a stream. It contains all the information needed to stream the object.

```
TStrIndex = Array[0..9999] of TStrIndexRec
```

Pointer array type used in a TStringList (595)

```
TStrIndexRec = packed record
  Key : Sw_Word;
  Count : Word;
  Offset : Word;
end
```

Record type used in a TStringList (595) to store the strings

```
TWordArray = Array[0..MaxWords-1] of Word
```

Array with maxmimum allowed number of words.

```
WordRec = packed record
  Hi : Byte;
  Lo : Byte;
end
```

Record describing a Word (in bytes)

67.2.3 Variables

```
invalidhandle : THandle
```

Value for invalid handle. Initial value for file stream handles or when the stream is closed.

67.3 Procedures and functions

67.3.1 Abstract

Synopsis: Abstract error handler.

Declaration: `procedure Abstract`

Visibility: default

Description: When implementing abstract methods, do not declare them as `abstract`. Instead, define them simply as `virtual`. In the implementation of such abstract methods, call the `Abstract` procedure. This allows explicit control of what happens when an abstract method is called.

The current implementation of `Abstract` terminates the program with a run-time error 211.

Errors: None.

67.3.2 CallPointerConstructor

Synopsis: Call a constructor with a pointer argument.

Declaration:
```
function CallPointerConstructor(Ctor: codepointer;Obj: pointer;
                                VMT: pointer;Param1: pointer) : pointer
```

Visibility: default

Description: `CallVoidConstructor` calls the constructor of an object. `Ctor` is the address of the constructor, `Obj` is a pointer to the instance. If it is `Nil`, then a new instance is allocated. `VMT` is a pointer to the object's VMT. `Param1` is passed to the constructor. The return value is a pointer to the instance.

Note that this can only be used on constructors that require a pointer as the sole argument. It can also be used to call a constructor with a single argument by reference.

Errors: If the constructor expects other arguments than a pointer, the stack may be corrupted.

See also: CallVoidConstructor (535), CallPointerMethod (534), CallVoidLocal (535), CallPointerLocal (534), CallVoidMethodLocal (536), CallPointerMethodLocal (534)

67.3.3 CallPointerLocal

Synopsis: Call a local nested function with a pointer argument

Declaration:
```
function CallPointerLocal(Func: codepointer;Frame: Pointer;
                          Param1: pointer) : pointer
```

Visibility: default

Description: `CallPointerLocal` calls the local procedure with address `Func`, where `Frame` is the frame of the wrapping function. It passes `Param1` to the local function.

Errors: If the local function expects other parameters than a pointer, the stack may become corrupted.

See also: CallPointerMethod (534), CallVoidMethod (535), CallVoidLocal (535), CallVoidMethodLocal (536), CallPointerMethodLocal (534), CallVoidConstructor (535), CallPointerConstructor (533)

67.3.4 CallPointerMethod

Synopsis: Call a method with a single pointer argument

Declaration:
```
function CallPointerMethod(Method: codepointer;Obj: pointer;
                           Param1: pointer) : pointer
```

Visibility: default

Description: `CallPointerMethod` calls the method with address `Method` for instance `Obj`. It passes `Param1` to the method as the single argument. It returns a pointer to the instance.

Errors: If the method expects other parameters than a single pointer, the stack may become corrupted.

See also: CallVoidMethod (535), CallVoidLocal (535), CallPointerLocal (534), CallVoidMethodLocal (536), CallPointerMethodLocal (534), CallVoidConstructor (535), CallPointerConstructor (533)

67.3.5 CallPointerMethodLocal

Synopsis: Call a local procedure of a method with a pointer argument

Declaration:
```
function CallPointerMethodLocal(Func: codepointer;Frame: Pointer;
                                Obj: pointer;Param1: pointer) : pointer
```

Visibility: default

Description: `CallPointerMethodLocal` calls the local procedure with address `Func`, where `Frame` is the frame of the wrapping method. It passes `Param1` to the local function.

Errors: If the local function expects other parameters than a pointer, the stack may become corrupted.

See also: CallPointerMethod (534), CallVoidMethod (535), CallPointerLocal (534), CallVoidLocal (535), CallVoidMethodLocal (536), CallVoidConstructor (535), CallPointerConstructor (533)

67.3.6 CallVoidConstructor

Synopsis: Call a constructor with no arguments

Declaration:
```
function CallVoidConstructor(Ctor: codepointer;Obj: pointer;
                             VMT: pointer) : pointer
```

Visibility: default

Description: `CallVoidConstructor` calls the constructor of an object. `Ctor` is the address of the constructor, `Obj` is a pointer to the instance. If it is `Nil`, then a new instance is allocated. `VMT` is a pointer to the object's VMT. The return value is a pointer to the instance.

Note that this can only be used on constructors that require no arguments.

Errors: If the constructor expects arguments, the stack may be corrupted.

See also: CallPointerConstructor (533), CallPointerMethod (534), CallVoidLocal (535), CallPointerLocal (534), CallVoidMethodLocal (536), CallPointerMethodLocal (534)

67.3.7 CallVoidLocal

Synopsis: Call a local nested procedure.

Declaration:
```
function CallVoidLocal(Func: codepointer;Frame: Pointer) : pointer
```

Visibility: default

Description: `CallVoidLocal` calls the local procedure with address `Func`, where `Frame` is the frame of the wrapping function.

Errors: If the local function expects parameters, the stack may become corrupted.

See also: CallPointerMethod (534), CallVoidMethod (535), CallPointerLocal (534), CallVoidMethodLocal (536), CallPointerMethodLocal (534), CallVoidConstructor (535), CallPointerConstructor (533)

67.3.8 CallVoidMethod

Synopsis: Call an object method

Declaration:
```
function CallVoidMethod(Method: codepointer;Obj: pointer) : pointer
```

Visibility: default

Description: `CallVoidMethod` calls the method with address `Method` for instance `Obj`. It returns a pointer to the instance.

Errors: If the method expects parameters, the stack may become corrupted.

See also: CallPointerMethod (534), CallVoidLocal (535), CallPointerLocal (534), CallVoidMethodLocal (536), CallPointerMethodLocal (534), CallVoidConstructor (535), CallPointerConstructor (533)

67.3.9 CallVoidMethodLocal

Synopsis: Call a local procedure of a method

Declaration:
```
function CallVoidMethodLocal(Func: codepointer;Frame: Pointer;
                             Obj: pointer) : pointer
```

Visibility: default

Description: `CallVoidMethodLocal` calls the local procedure with address `Func`, where `Frame` is the frame of the wrapping method.

Errors: If the local function expects parameters, the stack may become corrupted.

See also: CallPointerMethod (534), CallVoidMethod (535), CallPointerLocal (534), CallVoidLocal (535), CallPointerMethodLocal (534), CallVoidConstructor (535), CallPointerConstructor (533)

67.3.10 DisposeStr

Synopsis: Dispose of a shortstring which was allocated on the heap.

Declaration:
```
procedure DisposeStr(P: PString)
```

Visibility: default

Description: `DisposeStr` removes a dynamically allocated string from the heap.

For an example, see NewStr (537).

Errors: None.

See also: NewStr (537), SetStr (539)

67.3.11 LongDiv

Synopsis: Overflow safe divide

Declaration:
```
function LongDiv(X: LongInt;Y: Integer) : Integer
```

Visibility: default

Description: `LongDiv` divides `X` by `Y`. The result is of type `Integer` instead of type `Longint`, as you would get normally.

Errors: If Y is zero, a run-time error will be generated.

See also: LongMul (536)

67.3.12 LongMul

Synopsis: Overflow safe multiply.

Declaration:
```
function LongMul(X: Integer;Y: Integer) : LongInt
```

Visibility: default

Description: `LongMul` multiplies `X` with `Y`. The result is of type `Longint`. This avoids possible overflow errors you would normally get when multiplying `X` and `Y` that are too big.

Errors: None.

See also: LongDiv (536)

67.3.13 NewStr

Synopsis: Allocate a copy of a shortstring on the heap.

Declaration: `function NewStr(const S: string) : PString`

Visibility: default

Description: `NewStr` makes a copy of the string `S` on the heap, and returns a pointer to this copy. If the string is empty then `Nil` is returned.

The allocated memory is not based on the declared size of the string passed to `NewStr`, but is baed on the actual length of the string.

Errors: If not enough memory is available, an 'out of memory' error will occur.

See also: DisposeStr (536), SetStr (539)

Listing: ./objectex/ex40.pp

```
Program ex40;

{ Program to demonstrate the NewStr function }

Uses Objects;

Var S : String;
    P : PString;

begin
  S:='Some really cute string';
  P:=NewStr(S);
  If P^<>S then
    WriteIn ('Oh-oh... Something is wrong !!');
  DisposeStr(P);
end.
```

67.3.14 RegisterObjects

Synopsis: Register standard objects.

Declaration: `procedure RegisterObjects`

Visibility: default

Description: `RegisterObjects` registers the following objects for streaming:

1. `TCollection`, see TCollection (544).
2. `TStringCollection`, see TStringCollection (593).
3. `TStrCollection`, see TStrCollection (583).

Errors: None.

See also: RegisterType (538)

67.3.15 RegisterType

Synopsis: Register new object for streaming.

Declaration: `procedure RegisterType(var S: TStreamRec)`

Visibility: default

Description: `RegisterType` registers a new type for streaming. An object cannot be streamed unless it has been registered first. The stream record `S` needs to have the following fields set:

ObjType: Sw_Word This should be a unique identifier. Each possible type should have it's own identifier.

VmtLink: pointer This should contain a pointer to the VMT (Virtual Method Table) of the object you try to register.

Load : Pointeris a pointer to a method that initializes an instance of that object, and reads the initial values from a stream. This method should accept as it's sole argument a `PStream` type variable.

Store: Pointeris a pointer to a method that stores an instance of the object to a stream. This method should accept as it's sole argument a `PStream` type variable.

The VMT of the object can be retrieved with the following expression:

```
VmtLink: Ofs(TypeOf(MyType)^);
```

Errors: In case of error (if a object with the same `ObjType`) is already registered), run-time error 212 occurs.

Listing: ./objectex/myobject.pp

```
Unit MyObject;


Interface

Uses Objects;

Type
     PMyObject = ^TMyObject;
     TMyObject = Object(TObject)
       Field : Longint;
       Constructor Init;
       Constructor Load (Var Stream : TStream);
       Destructor Done;
       Procedure Store (Var Stream : TStream);
       Function  GetField : Longint;
       Procedure SetField (Value : Longint);
       end;

Implementation

Constructor TMyobject.Init;

begin
  Inherited Init;
  Field:=-1;
end;
```

```
Constructor TMyobject.Load (Var Stream : TStream);

begin
  Stream.Read(Field,Sizeof(Field));
end;

Destructor TMyObject.Done;

begin
end;

Function TMyObject.GetField : Longint;

begin
  GetField:=Field;
end;

Procedure TMyObject.SetField (Value : Longint);

begin
  Field:=Value;
end;

Procedure TMyObject.Store (Var Stream : TStream);

begin
  Stream.Write(Field,SizeOf(Field));
end;

Const MyObjectRec : TStreamRec = (
        Objtype : 666;
        vmtlink : Ofs(TypeOf(TMyObject)^);
        Load : @TMyObject.Load;
        Store : @TMyObject.Store;
        );

begin
  RegisterObjects;
  RegisterType (MyObjectRec);
end.
```

67.3.16 SetStr

Synopsis: Allocate a copy of a shortstring on the heap.

Declaration: `procedure SetStr(var p: PString;const s: string)`

Visibility: default

Description: `SetStr` makes a copy of the string `S` on the heap and returns the pointer to this copy in `P`. If `P` pointed to another string (i.e. was not `Nil`, the memory is released first. Contrary to NewStr (537), if the string is empty then a pointer to an empty string is returned.

The allocated memory is not based on the declared size of the string passed to `NewStr`, but is based on the actual length of the string.

Errors: If not enough memory is available, an 'out of memory' error will occur.

See also: DisposeStr (536), NewStr (537)

67.4 TBufStream

67.4.1 Description

`Bufstream` implements a buffered file stream. That is, all data written to the stream is written to memory first. Only when the buffer is full, or on explicit request, the data is written to disk.

Also, when reading from the stream, first the buffer is checked if there is any unread data in it. If so, this is read first. If not the buffer is filled again, and then the data is read from the buffer.

The size of the buffer is fixed and is set when constructing the file.

This is useful if you need heavy throughput for your stream, because it speeds up operations.

67.4.2 Method overview

Page	Method	Description
541	Close	Flush data and Close the file.
541	Done	Close the file and cleans up the instance.
541	Flush	FLush data from buffer, and write it to stream.
540	Init	Initialize an instance of `TBufStream` and open the file.
543	Open	Open the file if it is closed.
543	Read	Read data from the file to a buffer in memory.
542	Seek	Set current position in file.
542	Truncate	Flush buffer, and truncate the file at current position.
543	Write	Write data to the file from a buffer in memory.

67.4.3 TBufStream.Init

Synopsis: Initialize an instance of `TBufStream` and open the file.

Declaration: `constructor Init(FileName: FNameStr;Mode: Word;Size: Word)`

Visibility: default

Description: `Init` instantiates an instance of `TBufStream`. The name of the file that contains (or will contain) the data of the stream is given in `FileName`. The `Mode` parameter determines whether a new file should be created and what access rights you have on the file. It can be one of the following constants:

stCreateCreates a new file.

stOpenReadRead access only.

stOpenWriteWrite access only.

stOpenRead and write access.

The `Size` parameter determines the size of the buffer that will be created. It should be different from zero.

For an example see TBufStream.Flush (541).

Errors: On error, `Status` is set to `stInitError`, and `ErrorInfo` is set to the dos error code.

See also: TDosStream.Init (559), TBufStream.Done (541)

67.4.4 TBufStream.Done

Synopsis: Close the file and cleans up the instance.

Declaration: `destructor Done; Virtual`

Visibility: default

Description: `Done` flushes and closes the file if it was open and cleans up the instance of `TBufStream`.

For an example see TBufStream.Flush (541).

Errors: None.

See also: TDosStream.Done (559), TBufStream.Init (540), TBufStream.Close (541)

67.4.5 TBufStream.Close

Synopsis: Flush data and Close the file.

Declaration: `procedure Close; Virtual`

Visibility: default

Description: `Close` flushes and closes the file if it was open, and sets `Handle` to -1. Contrary to Done (541) it does not clean up the instance of `TBufStream`

For an example see TBufStream.Flush (541).

Errors: None.

See also: TStream.Close (589), TBufStream.Init (540), TBufStream.Done (541)

67.4.6 TBufStream.Flush

Synopsis: FLush data from buffer, and write it to stream.

Declaration: `procedure Flush; Virtual`

Visibility: default

Description: When the stream is in write mode, the contents of the buffer are written to disk, and the buffer position is set to zero. When the stream is in read mode, the buffer position is set to zero.

Errors: Write errors may occur if the file was in write mode. see Write (543) for more info on the errors.

See also: TStream.Close (589), TBufStream.Init (540), TBufStream.Done (541)

Listing: ./objectex/ex15.pp

```
Program ex15;

{ Program to demonstrate the TStream.Flush method }

Uses Objects;

Var L : String;
    P : PString;
    S : PBufStream; { Only one with Flush implemented. }

begin
```

```
  L:='Some constant string';
  { Buffer size of 100 }
  S:=New(PBufStream,Init('test.dat',stcreate,100));
  Writeln ('Writing "',L,'" to stream with handle ',S^.Handle);
  S^.WriteStr(@L);
  { At this moment, there is no data on disk yet. }
  S^.Flush;
  { Now there is. }
  S^.WriteStr(@L);
  { Close calls flush first }
  S^.Close;
  Writeln ('Closed stream. File handle is ',S^.Handle);
  S^.Open (stOpenRead);
  P:=S^.ReadStr;
  L:=P^;
  DisposeStr(P);
  Writeln ('Read "',L,'" from stream with handle ',S^.Handle);
  S^.Close;
  Dispose (S,Done);
end.
```

67.4.7 TBufStream.Truncate

Synopsis: Flush buffer, and truncate the file at current position.

Declaration: `procedure Truncate;  Virtual`

Visibility: default

Description: If the status of the stream is `stOK`, then `Truncate` tries to flush the buffer, and then truncates the stream size to the current file position.

For an example, see TDosStream.Truncate (560).

Errors: Errors can be those of Flush (541) or TDosStream.Truncate (560).

See also: TStream.Truncate (590), TDosStream.Truncate (560), TStream.GetSize (587)

67.4.8 TBufStream.Seek

Synopsis: Set current position in file.

Declaration: `procedure Seek(Pos: LongInt);  Virtual`

Visibility: default

Description: If the stream's status is `stOK`, then `Seek` sets the file position to `Pos`. `Pos` is a zero-based offset, counted from the beginning of the file.

For an example, see TStream.Seek (591);

Errors: In case an error occurs, the stream's status is set to `stSeekError`, and the OS error code is stored in `ErrorInfo`.

See also: TStream.Seek (591), TStream.GetPos (587)

67.4.9 TBufStream.Open

Synopsis: Open the file if it is closed.

Declaration: `procedure Open(OpenMode: Word);  Virtual`

Visibility: default

Description: If the stream's status is `stOK`, and the stream is closed then `Open` re-opens the file stream with mode `OpenMode`. This call can be used after a Close (541) call.

For an example, see TDosStream.Open (562).

Errors: If an error occurs when re-opening the file, then `Status` is set to `stOpenError`, and the OS error code is stored in `ErrorInfo`

See also: TStream.Open (589), TBufStream.Close (541)

67.4.10 TBufStream.Read

Synopsis: Read data from the file to a buffer in memory.

Declaration: `procedure Read(var Buf;Count: LongInt);  Virtual`

Visibility: default

Description: If the Stream is open and the stream status is `stOK` then `Read` will read `Count` bytes from the stream and place them in `Buf`.

`Read` will first try to read the data from the stream's internal buffer. If insufficient data is available, the buffer will be filled before contiunuing to read. This process is repeated until all needed data has been read.

For an example, see TStream.Read (592).

Errors: In case of an error, `Status` is set to `StReadError`, and `ErrorInfo` gets the OS specific error, or 0 when an attempt was made to read beyond the end of the stream.

See also: TStream.Read (592), TBufStream.Write (543)

67.4.11 TBufStream.Write

Synopsis: Write data to the file from a buffer in memory.

Declaration: `procedure Write(var Buf;Count: LongInt);  Virtual`

Visibility: default

Description: If the Stream is open and the stream status is `stOK` then `Write` will write `Count` bytes from `Buf` and place them in the stream.

`Write` will first try to write the data to the stream's internal buffer. When the internal buffer is full, then the contents will be written to disk. This process is repeated until all data has been written.

For an example, see TStream.Read (592).

Errors: In case of an error, `Status` is set to `StWriteError`, and `ErrorInfo` gets the OS specific error.

See also: TStream.Write (592), TBufStream.Read (543)

67.5 TCollection

67.5.1 Description

The `TCollection` object manages a collection of pointers or objects. It also provides a series of methods to manipulate these pointers or objects.

Whether or not objects are used depends on the kind of calls you use. All kinds come in 2 flavors, one for objects, one for pointers.

67.5.2 Method overview

Page	Method	Description
546	At	Return the item at a certain index.
554	AtDelete	Delete item at certain position.
553	AtFree	Free an item at the indicates position, calling it's destructor.
557	AtInsert	Insert an element at a certain position in the collection.
557	AtPut	Set collection item, overwriting an existing value.
553	Delete	Delete an item from the collection, but does not destroy it.
551	DeleteAll	Delete all elements from the collection. Objects are not destroyed.
545	Done	Clean up collection, release all memory.
556	Error	Set error code.
548	FirstThat	Return first item which matches a test.
555	ForEach	Execute procedure for each item in the list.
552	Free	Free item from collection, calling it's destructor.
550	FreeAll	Release all objects from the collection.
554	FreeItem	Destroy a non-nil item.
547	GetItem	Read one item off the stream.
546	IndexOf	Find the position of a certain item.
544	Init	Instantiate a new collection.
552	Insert	Insert a new item in the collection at the end.
548	LastThat	Return last item which matches a test.
545	Load	Initialize a new collection and load collection from a stream.
549	Pack	Remove all `>Nil` pointers from the collection.
558	PutItem	Put one item on the stream
556	SetLimit	Set maximum number of elements in the collection.
558	Store	Write collection to a stream.

67.5.3 TCollection.Init

Synopsis: Instantiate a new collection.

Declaration: `constructor Init(ALimit: Sw_Integer;ADelta: Sw_Integer)`

Visibility: default

Description: `Init` initializes a new instance of a collection. It sets the (initial) maximum number of items in the collection to `ALimit`. `ADelta` is the increase size : The number of memory places that will be allocated in case `ALimit` is reached, and another element is added to the collection.

For an example, see TCollection.ForEach (555).

Errors: None.

See also: TCollection.Load (545), TCollection.Done (545)

67.5.4 TCollection.Load

Synopsis: Initialize a new collection and load collection from a stream.

Declaration: `constructor Load(var S: TStream)`

Visibility: default

Description: `Load` initializes a new instance of a collection. It reads from stream `S` the item count, the item limit count, and the increase size. After that, it reads the specified number of items from the stream.

Errors: Errors returned can be those of GetItem (547).

See also: TCollection.Init (544), TCollection.GetItem (547), TCollection.Done (545)

Listing: ./objectex/ex22.pp

```
Program ex22;

{ Program to demonstrate the TCollection.Load method }

Uses Objects,MyObject; { For TMyObject definition and registration }

Var C : PCollection;
    M : PMyObject;
    I : Longint;
    S : PMemoryStream;

begin
  C:=New(PCollection,Init(100,10));
  For I:=1 to 100 do
    begin
    M:=New(PMyObject,Init);
    M^.SetField(100-I);
    C^.Insert(M);
    end;
  WriteIn ('Inserted ',C^.Count,' objects');
  S:=New(PMemorySTream,Init(1000,10));
  C^.Store(S^);
  C^.FreeAll;
  Dispose(C,Done);
  S^.Seek(0);
  C^.Load(S^);
  WriteIn ('Read ',C^.Count,' objects from stream.');
  Dispose(S,Done);
  Dispose(C,Done);
end.
```

67.5.5 TCollection.Done

Synopsis: Clean up collection, release all memory.

Declaration: `destructor Done;  Virtual`

Visibility: default

Description: `Done` frees all objects in the collection, and then releases all memory occupied by the instance.

For an example, see TCollection.ForEach (555).

Errors: None.

See also: TCollection.Init (544), TCollection.FreeAll (550)

67.5.6 TCollection.At

Synopsis: Return the item at a certain index.

Declaration: `function At(Index: Sw_Integer) : Pointer`

Visibility: default

Description: `At` returns the item at position `Index`.

Errors: If `Index` is less than zero or larger than the number of items in the collection, seepl{Error}{TCollection.Error} is called with `coIndexError` and `Index` as arguments, resulting in a run-time error.

See also: TCollection.Insert (552)

Listing: ./objectex/ex23.pp

```
Program ex23;

{ Program to demonstrate the TCollection.At method }

Uses Objects,MyObject; { For TMyObject definition and registration }

Var C : PCollection;
    M : PMyObject;
    I : Longint;

begin
  C:=New(PCollection,Init(100,10));
  For I:=1 to 100 do
    begin
    M:=New(PMyObject,Init);
    M^.SetField(100-I);
    C^.Insert(M);
    end;
  For I:=0 to C^.Count-1 do
    begin
    M:=C^.At(I);
    WriteIn ('Object ',i,' has field : ',M^.GetField);
    end;
  C^.FreeAll;
  Dispose(C,Done);
end.
```

67.5.7 TCollection.IndexOf

Synopsis: Find the position of a certain item.

Declaration: `function IndexOf(Item: Pointer) : Sw_Integer;  Virtual`

Visibility: default

Description: `IndexOf` returns the index of `Item` in the collection. If `Item` isn't present in the collection, -1 is returned.

Errors: If the item is not present, -1 is returned.

See also: TCollection.At (546), TCollection.GetItem (547), TCollection.Insert (552)

Listing: ./objectex/ex24.pp

```
Program ex24;

{ Program to demonstrate the TCollection.IndexOf method }

Uses Objects,MyObject; { For TMyObject definition and registration }

Var C : PCollection;
    M,Keep : PMyObject;
    I : Longint;

begin
  Randomize;
  C:=New(PCollection,Init(100,10));
  Keep:=Nil;
  For I:=1 to 100 do
    begin
    M:=New(PMyObject,Init);
    M^.SetField(I-1);
    If Random<0.1 then
      Keep:=M;
    C^.Insert(M);
    end;
  If Keep=Nil then
    begin
    Writeln ('Please run again. No object selected');
    Halt(1);
    end;
  Writeln ('Selected object has field : ',Keep^.GetField);
  Write ('Selected object has index : ',C^.IndexOf(Keep));
  Writeln (' should match it''s field.');
  C^.FreeAll;
  Dispose(C,Done);
end.
```

67.5.8 TCollection.GetItem

Synopsis: Read one item off the stream.

Declaration: `function GetItem(var S: TStream) : Pointer;  Virtual`

Visibility: default

Description: `GetItem` reads a single item off the stream `S`, and returns a pointer to this item. This method is used internally by the Load method, and should not be used directly.

Errors: Possible errors are the ones from TStream.Get (585).

See also: TStream.Get (585), TCollection.Store (558)

67.5.9 TCollection.LastThat

Synopsis: Return last item which matches a test.

Declaration: `function LastThat(Test: CodePointer) : Pointer`

Visibility: default

Description: This function returns the last item in the collection for which `Test` returns a non-nil result. `Test` is a function that accepts 1 argument: a pointer to an object, and that returns a pointer as a result.

Errors: None.

See also: TCollection.FirstThat (548)

Listing: ./objectex/ex25.pp

```
Program ex21;

{ Program to demonstrate the TCollection.Foreach method }

Uses Objects,MyObject; { For TMyObject definition and registration }

Var C : PCollection;
    M : PMyObject;
    I : Longint;

Function CheckField (Dummy: Pointer;P : PMyObject) : Longint;

begin
  If P^.GetField<56 then
    Checkfield:=1
  else
    CheckField:=0;
end;

begin
  C:=New(PCollection,Init(100,10));
  For I:=1 to 100 do
    begin
    M:=New(PMyObject,Init);
    M^.SetField(I);
    C^.Insert(M);
    end;
  Writeln ('Inserted ',C^.Count,' objects');
  Writeln ('Last one for which Field<56 has index (should be 54) : ',
           C^.IndexOf(C^.LastThat(@CheckField)));
  C^.FreeAll;
  Dispose(C,Done);
end.
```

67.5.10 TCollection.FirstThat

Synopsis: Return first item which matches a test.

Declaration: `function FirstThat(Test: CodePointer) : Pointer`

Visibility: default

Description: This function returns the first item in the collection for which `Test` returns a non-nil result. `Test` is a function that accepts 1 argument: a pointer to an object, and that returns a pointer as a result.

Errors: None.

See also: TCollection.LastThat (548)

Listing: ./objectex/ex26.pp

```
Program ex21;

{ Program to demonstrate the TCollection.FirstThat method }

Uses Objects,MyObject; { For TMyObject definition and registration }

Var C : PCollection;
    M : PMyObject;
    I : Longint;

Function CheckField (Dummy: Pointer;P : PMyObject) : Longint;

begin
  If P^.GetField>56 then
    Checkfield:=1
  else
    CheckField:=0;
end;

begin
  C:=New(PCollection,Init(100,10));
  For I:=1 to 100 do
    begin
    M:=New(PMyObject,Init);
    M^.SetField(I);
    C^.Insert(M);
    end;
  Writeln ('Inserted ',C^.Count,' objects');
  Writeln ('first one for which Field>56 has index (should be 56) : ',
           C^.IndexOf(C^.FirstThat(@CheckField)));
  C^.FreeAll;
  Dispose(C,Done);
end.
```

67.5.11 TCollection.Pack

Synopsis: Remove all `>Nil` pointers from the collection.

Declaration: `procedure Pack`

Visibility: default

Description: `Pack` removes all `Nil` pointers from the collection, and adjusts `Count` to reflect this change. No memory is freed as a result of this call. In order to free any memory, you can call `SetLimit` with an argument of `Count` after a call to `Pack`.

Errors: None.

See also: TCollection.SetLimit (556)

Listing: ./objectex/ex26.pp

```
Program ex21;

{ Program to demonstrate the TCollection.FirstThat method }

Uses Objects,MyObject; { For TMyObject definition and registration }

Var C : PCollection;
    M : PMyObject;
    I : Longint;

Function CheckField (Dummy: Pointer;P : PMyObject) : Longint;

begin
  If P^.GetField>56 then
    Checkfield:=1
  else
    CheckField:=0;
end;

begin
  C:=New(PCollection,Init(100,10));
  For I:=1 to 100 do
    begin
    M:=New(PMyObject,Init);
    M^.SetField(I);
    C^.Insert(M);
    end;
  Writeln ('Inserted ',C^.Count,' objects');
  Writeln ('first one for which Field>56 has index (should be 56) : ',
           C^.IndexOf(C^.FirstThat(@CheckField)));
  C^.FreeAll;
  Dispose(C,Done);
end.
```

67.5.12 TCollection.FreeAll

Synopsis: Release all objects from the collection.

Declaration: `procedure FreeAll`

Visibility: default

Description: `FreeAll` calls the destructor of each object in the collection. It doesn't release any memory occumpied by the collection itself, but it does set `Count` to zero.

See also: TCollection.DeleteAll (551), TCollection.FreeItem (554)

Listing: ./objectex/ex28.pp

```
Program ex28;

{ Program to demonstrate the TCollection.FreeAll method }

Uses Objects,MyObject; { For TMyObject definition and registration }

Var C : PCollection;
```

```
    M : PMyObject;
    I : Longint;

begin
  Randomize;
  C:=New(PCollection,Init(120,10));
  For I:=1 to 100 do
    begin
    M:=New(PMyObject,Init);
    M^.SetField(I-1);
    C^.Insert(M);
    end;
  Writeln('Added 100 Items.');
  C^.FreeAll;
  Writeln('Freed all objects.');
  Dispose(C,Done);
end.
```

67.5.13 TCollection.DeleteAll

Synopsis: Delete all elements from the collection. Objects are not destroyed.

Declaration: `procedure DeleteAll`

Visibility: default

Description: `DeleteAll` deletes all elements from the collection. It just sets the `Count` variable to zero. Contrary to FreeAll (550), `DeletAll` doesn't call the destructor of the objects.

Errors: None.

See also: TCollection.FreeAll (550), TCollection.Delete (553)

Listing: ./objectex/ex29.pp

```
Program ex29;

{
 Program to demonstrate the TCollection.DeleteAll method
 Compare with example 28, where FreeAll is used.
}

Uses Objects,MyObject; { For TMyObject definition and registration }

Var C : PCollection;
    M : PMyObject;
    I : Longint;

begin
  Randomize;
  C:=New(PCollection,Init(120,10));
  For I:=1 to 100 do
    begin
    M:=New(PMyObject,Init);
    M^.SetField(I-1);
    C^.Insert(M);
    end;
  Writeln('Added 100 Items.');
```

```
  C^.DeleteAll;
  Writeln ('Deleted all objects.');
  Dispose(C,Done);
end.
```

67.5.14 TCollection.Free

Synopsis: Free item from collection, calling it's destructor.

Declaration: `procedure Free(Item: Pointer)`

Visibility: default

Description: `Free` Deletes `Item` from the collection, and calls the destructor `Done` of the object.

Errors: If the `Item` is not in the collection, `Error` will be called with `coIndexError`.

See also: TCollection.FreeItem (554)

Listing: ./objectex/ex30.pp

```
Program ex30;

{ Program to demonstrate the TCollection.Free method }

Uses Objects,MyObject; { For TMyObject definition and registration }

Var C : PCollection;
    M : PMyObject;
    I : Longint;

begin
  Randomize;
  C:=New(PCollection,Init(120,10));
  For I:=1 to 100 do
    begin
    M:=New(PMyObject,Init);
    M^.SetField(I-1);
    C^.Insert(M);
    end;
  Writeln ('Added 100 Items.');
  With C^ do
    While Count>0 do Free(At(Count-1));
  Writeln ('Freed all objects.');
  Dispose(C,Done);
end.
```

67.5.15 TCollection.Insert

Synopsis: Insert a new item in the collection at the end.

Declaration: `procedure Insert(Item: Pointer); Virtual`

Visibility: default

Description: `Insert` inserts `Item` in the collection. `TCollection` inserts this item at the end, but descendent objects may insert it at another place.

Errors: None.

See also: TCollection.AtInsert (557), TCollection.AtPut (557)

67.5.16 TCollection.Delete

Synopsis: Delete an item from the collection, but does not destroy it.

Declaration: `procedure Delete(Item: Pointer)`

Visibility: default

Description: `Delete` deletes `Item` from the collection. It doesn't call the item's destructor, though. For this the Free (552) call is provided.

Errors: If the `Item` is not in the collection, `Error` will be called with `coIndexError`.

See also: TCollection.AtDelete (554), TCollection.Free (552)

Listing: ./objectex/ex31.pp

```
Program ex31;

{ Program to demonstrate the TCollection.Delete method }

Uses Objects,MyObject; { For TMyObject definition and registration }

Var C : PCollection;
    M : PMyObject;
    I : Longint;

begin
  Randomize;
  C:=New(PCollection,Init(120,10));
  For I:=1 to 100 do
    begin
    M:=New(PMyObject,Init);
    M^.SetField(I-1);
    C^.Insert(M);
    end;
  Writeln ('Added 100 Items.');
  With C^ do
    While Count>0 do Delete(At(Count-1));
  Writeln ('Freed all objects');
  Dispose(C,Done);
end.
```

67.5.17 TCollection.AtFree

Synopsis: Free an item at the indicates position, calling it's destructor.

Declaration: `procedure AtFree(Index: Sw_Integer)`

Visibility: default

Description: `AtFree` deletes the item at position `Index` in the collection, and calls the item's destructor if it is not `Nil`.

Errors: If `Index` isn't valid then Error (556) is called with `CoIndexError`.

See also: TCollection.Free (552), TCollection.AtDelete (554)

Listing: ./objectex/ex32.pp

```
Program ex32;

{ Program to demonstrate the TCollection.AtFree method }

Uses Objects,MyObject; { For TMyObject definition and registration }

Var C : PCollection;
    M : PMyObject;
    I : Longint;

begin
  Randomize;
  C:=New(PCollection,Init(120,10));
  For I:=1 to 100 do
    begin
    M:=New(PMyObject,Init);
    M^.SetField(I-1);
    C^.Insert(M);
    end;
  Writeln ('Added 100 Items');
  With C^ do
    While Count>0 do AtFree(Count-1);
  Writeln ('Freed all objects.');
  Dispose(C,Done);
end.
```

67.5.18 TCollection.FreeItem

Synopsis: Destroy a non-nil item.

Declaration: `procedure FreeItem(Item: Pointer);  Virtual`

Visibility: default

Description: `FreeItem` calls the destructor of `Item` if it is not nil.

Remark: This function is used internally by the TCollection object, and should not be called directly.

Errors: None.

See also: TCollection.Free (552), TCollection.AtFree (553)

67.5.19 TCollection.AtDelete

Synopsis: Delete item at certain position.

Declaration: `procedure AtDelete(Index: Sw_Integer)`

Visibility: default

Description: `AtDelete` deletes the pointer at position `Index` in the collection. It doesn't call the object's destructor.

Errors: If `Index` isn't valid then Error (556) is called with `CoIndexError`.

See also: TCollection.Delete (553)

Listing: ./objectex/ex33.pp

```
Program ex33;

{ Program to demonstrate the TCollection.AtDelete method }

Uses Objects,MyObject; { For TMyObject definition and registration }

Var C : PCollection;
    M : PMyObject;
    I : Longint;

begin
  Randomize;
  C:=New(PCollection,Init(120,10));
  For I:=1 to 100 do
    begin
    M:=New(PMyObject,Init);
    M^.SetField(I-1);
    C^.Insert(M);
    end;
  Writeln ('Added 100 Items.');
  With C^ do
    While Count>0 do AtDelete(Count-1);
  Writeln ('Freed all objects.');
  Dispose(C,Done);
end.
```

67.5.20 TCollection.ForEach

Synopsis: Execute procedure for each item in the list.

Declaration: `procedure ForEach(Action: CodePointer)`

Visibility: default

Description: `ForEach` calls `Action` for each element in the collection, and passes the element as an argument to `Action`.

`Action` is a procedural type variable that accepts a pointer as an argument.

Errors: None.

See also: TCollection.FirstThat (548), TCollection.LastThat (548)

Listing: ./objectex/ex21.pp

```
Program ex21;

{ Program to demonstrate the TCollection.Foreach method }

Uses Objects,MyObject; { For TMyObject definition and registration }

Var C : PCollection;
```

```
    M : PMyObject;
    I : Longint;

Procedure PrintField (Dummy: Pointer;P : PMyObject);

begin
  WriteIn ('Field : ',P^.GetField);
end;

begin
  C:=New(PCollection,Init(100,10));
  For I:=1 to 100 do
    begin
    M:=New(PMyObject,Init);
    M^.SetField(100-I);
    C^.Insert(M);
    end;
  WriteIn ('Inserted ',C^.Count,' objects');
  C^.ForEach(@PrintField);
  C^.FreeAll;
  Dispose(C,Done);
end.
```

67.5.21 TCollection.SetLimit

Synopsis: Set maximum number of elements in the collection.

Declaration: `procedure SetLimit(ALimit: Sw_Integer);  Virtual`

Visibility: default

Description: `SetLimit` sets the maximum number of elements in the collection. `ALimit` must not be less than `Count`, and should not be larger than `MaxCollectionSize`

For an example, see Pack (549).

Errors: None.

See also: TCollection.Init (544)

67.5.22 TCollection.Error

Synopsis: Set error code.

Declaration: `procedure Error(Code: Integer;Info: Integer);  Virtual`

Visibility: default

Description: `Error` is called by the various `TCollection` methods in case of an error condition. The default behaviour is to make a call to `RunError` with an error of `212-Code`.

This method can be overridden by descendent objects to implement a different error-handling.

See also: Abstract (533)

67.5.23 TCollection.AtPut

Synopsis: Set collection item, overwriting an existing value.

Declaration: `procedure AtPut(Index: Sw_Integer;Item: Pointer)`

Visibility: default

Description: `AtPut` sets the element at position `Index` in the collection to `Item`. Any previous value is overwritten.

For an example, see Pack (549).

Errors: If `Index` isn't valid then Error (556) is called with `CoIndexError`.

67.5.24 TCollection.AtInsert

Synopsis: Insert an element at a certain position in the collection.

Declaration: `procedure AtInsert(Index: Sw_Integer;Item: Pointer)`

Visibility: default

Description: `AtInsert` inserts `Item` in the collection at position `Index`, shifting all elements by one position. In case the current limit is reached, the collection will try to expand with a call to `SetLimit`

Errors: If `Index` isn't valid then Error (556) is called with `CoIndexError`. If the collection fails to expand, then `coOverFlow` is passd to `Error`.

See also: TCollection.Insert (552)

Listing: ./objectex/ex34.pp

```
Program ex34;

{ Program to demonstrate the TCollection.AtInsert method }

Uses Objects,MyObject; { For TMyObject definition and registration }

Var C : PCollection;
    M : PMyObject;
    I : Longint;

Procedure PrintField (Dummy: Pointer;P : PMyObject);

begin
  Writeln ('Field : ',P^.GetField);
end;


begin
  Randomize;
  C:=New(PCollection,Init(120,10));
  Writeln ('Inserting 100 records at random places.');
  For I:=1 to 100 do
    begin
    M:=New(PMyObject,Init);
    M^.SetField(I-1);
    If I=1 then
      C^.Insert(M)
```

```
    else
      With C^ do
        AtInsert(Random(Count),M);
    end;
  WriteIn ('Values : ');
  C^.Foreach(@PrintField);
  Dispose(C,Done);
end.
```

67.5.25 TCollection.Store

Synopsis: Write collection to a stream.

Declaration: `procedure Store(var S: TStream)`

Visibility: default

Description: `Store` writes the collection to the stream `S`. It does this by writeing the current `Count`, `Limit` and `Delta` to the stream, and then writing each item to the stream.

The contents of the stream are then suitable for instantiating another collection with Load (545).

For an example, see TCollection.Load (545).

Errors: Errors returned are those by TStream.Put (590).

See also: TCollection.Load (545), TCollection.PutItem (558)

67.5.26 TCollection.PutItem

Synopsis: Put one item on the stream

Declaration: `procedure PutItem(var S: TStream;Item: Pointer);  Virtual`

Visibility: default

Description: `PutItem` writes `Item` to stream `S`. This method is used internaly by the `TCollection` object, and should not be called directly.

Errors: Errors are those returned by TStream.Put (590).

See also: Store (558), GetItem (547)

67.6 TDosStream

67.6.1 Description

`TDosStream` is a stream that stores it's contents in a file. it overrides a couple of methods of TStream (585) for this.

In addition to the fields inherited from `TStream` (see TStream (585)), there are some extra fields, that describe the file. (mainly the name and the OS file handle)

No buffering in memory is done when using `TDosStream`. All data are written directly to the file. For a stream that buffers in memory, see TBufStream (540).

67.6.2 Method overview

Page	Method	Description
560	Close	Close the file.
559	Done	Closes the file and cleans up the instance.
559	Init	Instantiate a new instance of TDosStream.
562	Open	Open the file stream
562	Read	Read data from the stream to a buffer.
561	Seek	Set file position.
560	Truncate	Truncate the file on the current position.
563	Write	Write data from a buffer to the stream.

67.6.3 TDosStream.Init

Synopsis: Instantiate a new instance of TDosStream.

Declaration: `constructor Init(FileName: FNameStr;Mode: Word)`

Visibility: default

Description: `Init` instantiates an instance of `TDosStream`. The name of the file that contains (or will contain) the data of the stream is given in `FileName`. The `Mode` parameter determines whether a new file should be created and what access rights you have on the file. It can be one of the following constants:

stCreateCreates a new file.

stOpenReadRead access only.

stOpenWriteWrite access only.

stOpenRead and write access.

For an example, see TDosStream.Truncate (560).

Errors: On error, Status (**??**) is set to `stInitError`, and `ErrorInfo` is set to the dos error code.

See also: TDosStream.Done (559)

67.6.4 TDosStream.Done

Synopsis: Closes the file and cleans up the instance.

Declaration: `destructor Done;  Virtual`

Visibility: default

Description: `Done` closes the file if it was open and cleans up the instance of `TDosStream`.

for an example, see e.g. TDosStream.Truncate (560).

Errors: None.

See also: TDosStream.Init (559), TDosStream.Close (560)

67.6.5 TDosStream.Close

Synopsis: Close the file.

Declaration: `procedure Close; Virtual`

Visibility: default

Description: `Close` closes the file if it was open, and sets `Handle` to -1. Contrary to Done (559) it does not clean up the instance of `TDosStream`

For an example, see TDosStream.Open (562).

Errors: None.

See also: TStream.Close (589), TDosStream.Init (559), TDosStream.Done (559)

67.6.6 TDosStream.Truncate

Synopsis: Truncate the file on the current position.

Declaration: `procedure Truncate; Virtual`

Visibility: default

Description: If the status of the stream is `stOK`, then `Truncate` tries to truncate the stream size to the current file position.

Errors: If an error occurs, the stream's status is set to `stError` and `ErrorInfo` is set to the OS error code.

See also: TStream.Truncate (590), TStream.GetSize (587)

Listing: ./objectex/ex16.pp

```
Program ex16;

{ Program to demonstrate the TStream.Truncate method }

Uses Objects;

Var L : String;
    P : PString;
    S : PDosStream; { Only one with Truncate implemented. }

begin
  L:='Some constant string';
  { Buffer size of 100 }
  S:=New(PDosStream,Init('test.dat',stcreate));
  Writeln ('Writing "',L,'" to stream with handle ',S^.Handle);
  S^.WriteStr(@L);
  S^.WriteStr(@L);
  { Close calls flush first }
  S^.Close;
  S^.Open (stOpen);
  Writeln ('Size of stream is : ',S^.GetSize);
  P:=S^.ReadStr;
  L:=P^;
  DisposeStr(P);
  Writeln ('Read "',L,'" from stream with handle ',S^.Handle);
```

```
  S^.Truncate;
  Writeln ('Truncated stream. Size is : ',S^.GetSize);
  S^.Close;
  Dispose (S,Done);
end.
```

67.6.7 TDosStream.Seek

Synopsis: Set file position.

Declaration: `procedure Seek(Pos: LongInt);  Virtual`

Visibility: default

Description: If the stream's status is `stOK`, then `Seek` sets the file position to `Pos`. `Pos` is a zero-based offset, counted from the beginning of the file.

Errors: In case an error occurs, the stream's status is set to `stSeekError`, and the OS error code is stored in `ErrorInfo`.

See also: TStream.Seek (591), TStream.GetPos (587)

Listing: ./objectex/ex17.pp

```
Program ex17;

{ Program to demonstrate the TStream.Seek method }

Uses Objects;

Var L : String;
    Marker : Word;
    P : PString;
    S : PDosStream;

begin
  L:='Some constant string';
  { Buffer size of 100 }
  S:=New(PDosStream,Init('test.dat',stcreate));
  Writeln ('Writing "',L,'" to stream.');
  S^.WriteStr(@L);
  Marker:=S^.GetPos;
  Writeln ('Set marker at ',Marker);
  L:='Some other constant String';
  Writeln ('Writing "',L,'" to stream.');
  S^.WriteStr(@L);
  S^.Close;
  S^.Open (stOpenRead);
  Writeln ('Size of stream is : ',S^.GetSize);
  Writeln ('Seeking to marker');
  S^.Seek(Marker);
  P:=S^.ReadStr;
  L:=P^;
  DisposeStr(P);
  Writeln ('Read "',L,'" from stream.');
  S^.Close;
  Dispose (S,Done);
end.
```

67.6.8 TDosStream.Open

Synopsis: Open the file stream

Declaration: `procedure Open(OpenMode: Word);  Virtual`

Visibility: default

Description: If the stream's status is `stOK`, and the stream is closed then `Open` re-opens the file stream with mode `OpenMode`. This call can be used after a Close (560) call.

Errors: If an error occurs when re-opening the file, then `Status` is set to `stOpenError`, and the OS error code is stored in `ErrorInfo`

See also: TStream.Open (589), TDosStream.Close (560)

Listing: ./objectex/ex14.pp

```
Program ex14;

{ Program to demonstrate the TStream.Close method }

Uses Objects;

Var L : String;
    P : PString;
    S : PDosStream; { Only one with Close implemented. }

begin
  L:='Some constant string';
  S:=New(PDosStream,Init('test.dat',stcreate));
  Writeln ('Writing "',L,'" to stream with handle ',S^.Handle);
  S^.WriteStr(@L);
  S^.Close;
  Writeln ('Closed stream. File handle is ',S^.Handle);
  S^.Open (stOpenRead);
  P:=S^.ReadStr;
  L:=P^;
  DisposeStr(P);
  Writeln ('Read "',L,'" from stream with handle ',S^.Handle);
  S^.Close;
  Dispose (S,Done);
end.
```

67.6.9 TDosStream.Read

Synopsis: Read data from the stream to a buffer.

Declaration: `procedure Read(var Buf;Count: LongInt);  Virtual`

Visibility: default

Description: If the Stream is open and the stream status is `stOK` then `Read` will read `Count` bytes from the stream and place them in `Buf`.

For an example, see TStream.Read (592).

Errors: In case of an error, `Status` is set to `StReadError`, and `ErrorInfo` gets the OS specific error, or 0 when an attempt was made to read beyond the end of the stream.

See also: TStream.Read (592), TDosStream.Write (563)

67.6.10 TDosStream.Write

Synopsis: Write data from a buffer to the stream.

Declaration: `procedure Write(var Buf;Count: LongInt);  Virtual`

Visibility: default

Description: If the Stream is open and the stream status is `stOK` then `Write` will write `Count` bytes from `Buf` and place them in the stream.

For an example, see TStream.Read (592).

Errors: In case of an error, `Status` is set to `StWriteError`, and `ErrorInfo` gets the OS specific error.

See also: TStream.Write (592), TDosStream.Read (562)

67.7 TMemoryStream

67.7.1 Description

The `TMemoryStream` object implements a stream that stores it's data in memory. The data is stored on the heap, with the possibility to specify the maximum amout of data, and the size of the memory blocks being used.

See also: TStream (585)

67.7.2 Method overview

Page	Method	Description
564	Done	Clean up memory and destroy the object instance.
563	Init	Initialize memory stream, reserves memory for stream data.
565	Read	Read data from the stream to a location in memory.
564	Truncate	Set the stream size to the current position.
565	Write	Write data to the stream.

67.7.3 TMemoryStream.Init

Synopsis: Initialize memory stream, reserves memory for stream data.

Declaration: `constructor Init(ALimit: LongInt;ABlockSize: Word)`

Visibility: default

Description: `Init` instantiates a new `TMemoryStream` object. The memorystreamobject will initially allocate at least `ALimit` bytes memory, divided into memory blocks of size `ABlockSize`. The number of blocks needed to get to `ALimit` bytes is rounded up.

By default, the number of blocks is 1, and the size of a block is 8192. This is selected if you specify 0 as the blocksize.

For an example, see e.g TStream.CopyFrom (593).

Errors: If the stream cannot allocate the initial memory needed for the memory blocks, then the stream's status is set to `stInitError`.

See also: TMemoryStream.Done (564)

67.7.4 TMemoryStream.Done

Synopsis: Clean up memory and destroy the object instance.

Declaration: `destructor Done; Virtual`

Visibility: default

Description: `Done` releases the memory blocks used by the stream, and then cleans up the memory used by the stream object itself.

For an example, see e.g TStream.CopyFrom (593).

Errors: None.

See also: TMemoryStream.Init (563)

67.7.5 TMemoryStream.Truncate

Synopsis: Set the stream size to the current position.

Declaration: `procedure Truncate; Virtual`

Visibility: default

Description: `Truncate` sets the size of the memory stream equal to the current position. It de-allocates any memory-blocks that are no longer needed, so that the new size of the stream is the current position in the stream, rounded up to the first multiple of the stream blocksize.

Errors: If an error occurs during memory de-allocation, the stream's status is set to `stError`

See also: TStream.Truncate (590)

Listing: ./objectex/ex20.pp

```
Program ex20;

{ Program to demonstrate the TMemoryStream.Truncate method }

Uses Objects;

Var L : String;
    P : PString;
    S : PMemoryStream;
    I : Longint;

begin
  L:='Some constant string';
  { Buffer size of 100 }
  S:=New(PMemoryStream,Init(1000,100));
  Writeln ('Writing 100 times "',L,'" to stream.');
  For I:=1 to 100 do
    S^.WriteStr(@L);
  Writeln ('Finished.');
  S^.Seek(100);
  S^.Truncate;
  Writeln ('Truncated at byte 100.');
  Dispose (S,Done);
  Writeln ('Finished.');
end.
```

67.7.6 TMemoryStream.Read

Synopsis: Read data from the stream to a location in memory.

Declaration: `procedure Read(var Buf;Count: LongInt);  Virtual`

Visibility: default

Description: `Read` reads `Count` bytes from the stream to `Buf`. It updates the position of the stream.

For an example, see TStream.Read (592).

Errors: If there is not enough data available, no data is read, and the stream's status is set to `stReadError`.

See also: TStream.Read (592), TMemoryStream.Write (565)

67.7.7 TMemoryStream.Write

Synopsis: Write data to the stream.

Declaration: `procedure Write(var Buf;Count: LongInt);  Virtual`

Visibility: default

Description: `Write` copies `Count` bytes from `Buf` to the stream. It updates the position of the stream.

If not enough memory is available to hold the extra `Count` bytes, then the stream will try to expand, by allocating as much blocks with size `BlkSize` (as specified in the constuctor call Init (563)) as needed.

For an example, see TStream.Read (592).

Errors: If the stream cannot allocate more memory, then the status is set to `stWriteError`

See also: TStream.Write (592), TMemoryStream.Read (565)

67.8 TObject

67.8.1 Description

This type serves as the basic object for all other objects in the Objects unit.

67.8.2 Method overview

Page	Method	Description
567	Done	Destroy an object.
566	Free	Destroy an object and release all memory.
565	Init	Construct (initialize) a new object
566	Is_Object	Check whether a pointer points to an object.

67.8.3 TObject.Init

Synopsis: Construct (initialize) a new object

Declaration: `constructor Init`

Visibility: default

Description: Instantiates a new object of type `TObject`. It fills the instance up with Zero bytes.

For an example, see Free (566)

Errors: None.

See also: TObject.Free (566), TObject.Done (567)

67.8.4 TObject.Free

Synopsis: Destroy an object and release all memory.

Declaration: `procedure Free`

Visibility: default

Description: `Free` calls the destructor of the object, and releases the memory occupied by the instance of the object.

Errors: No checking is performed to see whether `self` is `nil` and whether the object is indeed allocated on the heap.

See also: TObject.Init (565), TObject.Done (567)

Listing: ./objectex/ex7.pp

```
program ex7;

{ Program to demonstrate the TObject.Free call }

Uses Objects;

Var O : PObject;

begin
  // Allocate memory for object.
  O:=New(PObject,Init);
  // Free memory of object.
  O^.free;
end.
```

67.8.5 TObject.Is_Object

Synopsis: Check whether a pointer points to an object.

Declaration: `function Is_Object(P: Pointer) : Boolean`

Visibility: default

Description: `Is_Object` returns `True` if the pointer `P` points to an instance of a `TObject` descendent, it returns `false` otherwise.

67.8.6 TObject.Done

Synopsis: Destroy an object.

Declaration: `destructor Done; Virtual`

Visibility: default

Description: `Done`, the destructor of `TObject` does nothing. It is mainly intended to be used in the TObject.Free (566) method.

The destructore Done does not free the memory occupied by the object.

Errors: None.

See also: TObject.Free (566), TObject.Init (565)

Listing: ./objectex/ex8.pp

```
program ex8;

{ Program to demonstrate the TObject.Done call }

Uses Objects;

Var O : PObject;

begin
  // Allocate memory for object.
  O:=New(PObject,Init);
  O^.Done;
end.
```

67.9 TPoint

67.9.1 Description

Record describing a point in a 2 dimensional plane.

67.10 TRect

67.10.1 Description

Describes a rectangular region in a plane.

67.10.2 Method overview

Page	Method	Description
572	Assign	Set rectangle corners.
569	Contains	Determine if a point is inside the rectangle
569	Copy	Copy cornerpoints from another rectangle.
568	Empty	Is the surface of the rectangle zero
569	Equals	Do the corners of the rectangles match
572	Grow	Expand rectangle with certain size.
570	Intersect	Reduce rectangle to intersection with another rectangle
571	Move	Move rectangle along a vector.
570	Union	Enlarges rectangle to encompas another rectangle.

67.10.3 TRect.Empty

Synopsis: Is the surface of the rectangle zero

Declaration: `function Empty : Boolean`

Visibility: default

Description: `Empty` returns `True` if the rectangle defined by the corner points `A`, `B` has zero or negative surface.

Errors: None.

See also: TRect.Equals (569), TRect.Contains (569)

Listing: ./objectex/ex1.pp

```
Program ex1;

{ Program to demonstrate TRect.Empty }

Uses objects;


Var ARect,BRect : TRect;
    P : TPoint;

begin
  With ARect.A do
    begin
    X:=10;
    Y:=10;
    end;
  With ARect.B do
    begin
    X:=20;
    Y:=20;
    end;
  { Offset B by (5,5) }
  With BRect.A do
    begin
    X:=15;
    Y:=15;
    end;
  With BRect.B do
    begin
    X:=25;
    Y:=25;
    end;
  { Point }
  With P do
    begin
    X:=15;
    Y:=15;
    end;
  Writeln ('A empty : ',ARect.Empty);
  Writeln ('B empty : ',BRect.Empty);
  Writeln ('A Equals B : ',ARect.Equals(BRect));
  Writeln ('A Contains (15,15) : ',ARect.Contains(P));
end.
```

67.10.4 TRect.Equals

Synopsis: Do the corners of the rectangles match

Declaration: `function Equals(R: TRect) : Boolean`

Visibility: default

Description: `Equals` returns `True` if the rectangle has the same corner points `A,B` as the rectangle R, and `False` otherwise.

For an example, see TRect.Empty (568)

Errors: None.

See also: TRect.Empty (568), TRect.Contains (569)

67.10.5 TRect.Contains

Synopsis: Determine if a point is inside the rectangle

Declaration: `function Contains(P: TPoint) : Boolean`

Visibility: default

Description: `Contains` returns `True` if the point `P` is contained in the rectangle (including borders), `False` otherwise.

Errors: None.

See also: TRect.Intersect (570), TRect.Equals (569)

67.10.6 TRect.Copy

Synopsis: Copy cornerpoints from another rectangle.

Declaration: `procedure Copy(R: TRect)`

Visibility: default

Description: Assigns the rectangle R to the object. After the call to `Copy`, the rectangle R has been copied to the object that invoked `Copy`.

Errors: None.

See also: TRect.Assign (572)

Listing: ./objectex/ex2.pp

```
Program ex2;

{ Program to demonstrate TRect.Copy }

Uses objects;

Var ARect,BRect,CRect : TRect;

begin
  ARect.Assign(10,10,20,20);
  BRect.Assign(15,15,25,25);
```

```
  CRect.Copy(ARect);
  If ARect.Equals(CRect) Then
    WriteIn ('ARect equals CRect')
  Else
    WriteIn ('ARect does not equal CRect !');
end.
```

67.10.7 TRect.Union

Synopsis: Enlarges rectangle to encompas another rectangle.

Declaration: `procedure Union(R: TRect)`

Visibility: default

Description: `Union` enlarges the current rectangle so that it becomes the union of the current rectangle with the rectangle `R`.

Errors: None.

See also: TRect.Intersect (570)

Listing: ./objectex/ex3.pp

```
Program ex3;

{ Program to demonstrate TRect.Union }

Uses objects;


Var ARect,BRect,CRect : TRect;

begin
  ARect.Assign(10,10,20,20);
  BRect.Assign(15,15,25,25);
  { CRect is union of ARect and BRect }
  CRect.Assign(10,10,25,25);
  { Calculate it explicitly}
  ARect.Union(BRect);
  If ARect.Equals(CRect) Then
    WriteIn ('ARect equals CRect')
  Else
    WriteIn ('ARect does not equal CRect !');
end.
```

67.10.8 TRect.Intersect

Synopsis: Reduce rectangle to intersection with another rectangle

Declaration: `procedure Intersect(R: TRect)`

Visibility: default

Description: `Intersect` makes the intersection of the current rectangle with `R`. If the intersection is empty, then the rectangle is set to the empty rectangle at coordinate (0,0).

Errors: None.

See also: TRect.Union (570)

Listing: ./objectex/ex4.pp

```
Program ex4;

{ Program to demonstrate TRect.Intersect }

Uses objects;


Var ARect,BRect,CRect : TRect;

begin
  ARect.Assign(10,10,20,20);
  BRect.Assign(15,15,25,25);
  { CRect is intersection of ARect and BRect }
  CRect.Assign(15,15,20,20);
  { Calculate it explicitly}
  ARect.Intersect(BRect);
  If ARect.Equals(CRect) Then
    WriteIn ('ARect equals CRect')
  Else
    WriteIn ('ARect does not equal CRect !');
  BRect.Assign(25,25,30,30);
  Arect.Intersect(BRect);
  If ARect.Empty Then
    WriteIn ('ARect is empty');
end.
```

67.10.9 TRect.Move

Synopsis: Move rectangle along a vector.

Declaration: `procedure Move(ADX: Sw_Integer;ADY: Sw_Integer)`

Visibility: default

Description: `Move` moves the current rectangle along a vector with components `(ADX,ADY)`. It adds `ADX` to the X-coordinate of both corner points, and `ADY` to both end points.

Errors: None.

See also: TRect.Grow (572)

Listing: ./objectex/ex5.pp

```
Program ex5;

{ Program to demonstrate TRect.Move }

Uses objects;


Var ARect,BRect : TRect;
```

```
begin
  ARect.Assign(10,10,20,20);
  ARect.Move(5,5);
  // Brect should be where new ARect is.
  BRect.Assign(15,15,25,25);
  If ARect.Equals(BRect) Then
    WriteIn ('ARect equals BRect')
  Else
    WriteIn ('ARect does not equal BRect !');
end.
```

67.10.10 TRect.Grow

Synopsis: Expand rectangle with certain size.

Declaration: `procedure Grow(ADX: Sw_Integer;ADY: Sw_Integer)`

Visibility: default

Description: `Grow` expands the rectangle with an amount `ADX` in the `X` direction (both on the left and right side of the rectangle, thus adding a length 2*ADX to the width of the rectangle), and an amount `ADY` in the `Y` direction (both on the top and the bottom side of the rectangle, adding a length 2*ADY to the height of the rectangle.

`ADX` and `ADY` can be negative. If the resulting rectangle is empty, it is set to the empty rectangle at `(0,0)`.

Errors: None.

See also: TRect.Move (571)

Listing: ./objectex/ex6.pp

```
Program ex6;

{ Program to demonstrate TRect.Grow }

Uses objects;


Var ARect,BRect : TRect;

begin
  ARect.Assign(10,10,20,20);
  ARect.Grow(5,5);
  // Brect should be where new ARect is.
  BRect.Assign(5,5,25,25);
  If ARect.Equals(BRect) Then
    WriteIn ('ARect equals BRect')
  Else
    WriteIn ('ARect does not equal BRect !');
end.
```

67.10.11 TRect.Assign

Synopsis: Set rectangle corners.

Declaration: `procedure Assign(XA: Sw_Integer;YA: Sw_Integer;XB: Sw_Integer; YB: Sw_Integer)`

Visibility: default

Description: `Assign` sets the corner points of the rectangle to `(XA,YA)` and `(Xb,Yb)`.

For an example, see TRect.Copy (569).

Errors: None.

See also: TRect.Copy (569)

67.11 TResourceCollection

67.11.1 Description

A `TResourceCollection` manages a collection of resource names. It stores the position and the size of a resource, as well as the name of the resource. It stores these items in records that look like this:

```
TYPE
   TResourceItem = packed RECORD
      Posn: LongInt;
      Size: LongInt;
      Key : String;
   End;
   PResourceItem = ^TResourceItem;
```

It overrides some methods of `TStringCollection` in order to accomplish this.

Remark: Remark that the `TResourceCollection` manages the names of the resources and their assiciated positions and sizes, it doesn't manage the resources themselves.

67.11.2 Method overview

Page	Method	Description
574	FreeItem	Release memory occupied by item.
574	GetItem	Read an item from the stream.
573	KeyOf	Return the key of an item in the collection.
574	PutItem	Write an item to the stream.

67.11.3 TResourceCollection.KeyOf

Synopsis: Return the key of an item in the collection.

Declaration: `function KeyOf(Item: Pointer) : Pointer;  Virtual`

Visibility: default

Description: `KeyOf` returns the key of an item in the collection. For resources, the key is a pointer to the string with the resource name.

Errors: None.

See also: TStringCollection.Compare (594)

67.11.4 TResourceCollection.GetItem

Synopsis: Read an item from the stream.

Declaration: `function GetItem(var S: TStream) : Pointer;  Virtual`

Visibility: default

Description: `GetItem` reads a resource item from the stream `S`. It reads the position, size and name from the stream, in that order. It DOES NOT read the resource itself from the stream.

The resulting item is not inserted in the collection. This call is manly for internal use by the TCollection.Load (545) method.

Errors: Errors returned are those by TStream.Read (592)

See also: TCollection.Load (545), TStream.Read (592)

67.11.5 TResourceCollection.FreeItem

Synopsis: Release memory occupied by item.

Declaration: `procedure FreeItem(Item: Pointer);  Virtual`

Visibility: default

Description: `FreeItem` releases the memory occupied by `Item`. It de-allocates the name, and then the resourceitem record.

It does NOT remove the item from the collection.

Errors: None.

See also: TCollection.FreeItem (554)

67.11.6 TResourceCollection.PutItem

Synopsis: Write an item to the stream.

Declaration: `procedure PutItem(var S: TStream;Item: Pointer);  Virtual`

Visibility: default

Description: `PutItem` writes `Item` to the stream `S`. It does this by writing the position and size and name of the resource item to the stream.

This method is used primarily by the Store (558) method.

Errors: Errors returned are those by TStream.Write (592).

See also: TCollection.Store (558)

67.12 TResourceFile

67.12.1 Description

TResourceFile (574) represents the resources in a binary file image.

67.12.2 Method overview

Page	Method	Description
575	Count	Number of resources in the file
577	Delete	Delete a resource from the file
575	Done	Destroy the instance and remove it from memory.
576	Flush	Writes the resources to the stream.
576	Get	Return a resource by key name.
575	Init	Instantiate a new instance.
576	KeyAt	Return the key of the item at a certain position.
577	Put	Set a resource by key name.
576	SwitchTo	Write resources to a new stream.

67.12.3 TResourceFile.Init

Synopsis: Instantiate a new instance.

Declaration: `constructor Init(AStream: PStream)`

Visibility: default

Description: `Init` instantiates a new instance of a `TResourceFile` object. If `AStream` is not nil then it is considered as a stream describing an executable image on disk.

`Init` will try to position the stream on the start of the resources section, and read all resources from the stream.

Errors: None.

See also: TResourceFile.Done (575)

67.12.4 TResourceFile.Done

Synopsis: Destroy the instance and remove it from memory.

Declaration: `destructor Done;  Virtual`

Visibility: default

Description: `Done` cleans up the instance of the `TResourceFile` Object. If `Stream` was specified at initialization, then `Stream` is disposed of too.

Errors: None.

See also: TResourceFile.Init (575)

67.12.5 TResourceFile.Count

Synopsis: Number of resources in the file

Declaration: `function Count : Sw_Integer`

Visibility: default

Description: `Count` returns the number of resources. If no resources were read, zero is returned.

Errors: None.

See also: TResourceFile.Init (575)

67.12.6 TResourceFile.KeyAt

Synopsis: Return the key of the item at a certain position.

Declaration: `function KeyAt(I: Sw_Integer) : string`

Visibility: default

Description: `KeyAt` returns the key (the name) of the `I`-th resource.

Errors: In case `I` is invalid, `TCollection.Error` will be executed.

See also: TResourceFile.Get (576)

67.12.7 TResourceFile.Get

Synopsis: Return a resource by key name.

Declaration: `function Get(Key: string) : PObject`

Visibility: default

Description: `Get` returns a pointer to a instance of a resource identified by `Key`. If `Key` cannot be found in the list of resources, then `Nil` is returned.

Errors: Errors returned may be those by `TStream.Get`

67.12.8 TResourceFile.SwitchTo

Synopsis: Write resources to a new stream.

Declaration: `function SwitchTo(AStream: PStream;Pack: Boolean) : PStream`

Visibility: default

Description: `SwitchTo` switches to a new stream to hold the resources in. `AStream` will be the new stream after the call to `SwitchTo`.

If `Pack` is true, then all the known resources will be copied from the current stream to the new stream (`AStream`). If `Pack` is `False`, then only the current resource is copied.

The return value is the value of the original stream: `Stream`.

The `Modified` flag is set as a consequence of this call.

Errors: Errors returned can be those of TStream.Read (592) and TStream.Write (592).

See also: TResourceFile.Flush (576)

67.12.9 TResourceFile.Flush

Synopsis: Writes the resources to the stream.

Declaration: `procedure Flush`

Visibility: default

Description: If the `Modified` flag is set to `True`, then `Flush` writes the resources to the stream `Stream`. It sets the `Modified` flag to true after that.

Errors: Errors can be those by TStream.Seek (591) and TStream.Write (592).

See also: TResourceFile.SwitchTo (576)

67.12.10 TResourceFile.Delete

Synopsis: Delete a resource from the file

Declaration: `procedure Delete(Key: string)`

Visibility: default

Description: `Delete` deletes the resource identified by `Key` from the collection. It sets the `Modified` flag to true.

Errors: None.

See also: TResourceFile.Flush (576)

67.12.11 TResourceFile.Put

Synopsis: Set a resource by key name.

Declaration: `procedure Put(Item: PObject;Key: string)`

Visibility: default

Description: `Put` sets the resource identified by `Key` to `Item`. If no such resource exists, a new one is created. The item is written to the stream.

Errors: Errors returned may be those by TStream.Put (590) and `TStream.Seek`

See also: Get (576)

67.13 TSortedCollection

67.13.1 Description

`TSortedCollection` is an abstract class, implementing a sorted collection. You should never use an instance of `TSortedCollection` directly, instead you should declare a descendent type, and override the Compare (579) method.

Because the collection is ordered, `TSortedCollection` overrides some `TCollection` methods, to provide faster routines for lookup.

The Compare (579) method decides how elements in the collection should be ordered. Since `TCollection` has no way of knowing how to order pointers, you must override the compare method.

Additionally, `TCollection` provides a means to filter out duplicates. if you set `Duplicates` to `False` (the default) then duplicates will not be allowed.

The example below defines a descendent of `TSortedCollection` which is used in the examples.

67.13.2 Method overview

Page	Method	Description
579	Compare	Compare two items in the collection.
579	IndexOf	Return index of an item in the collection.
578	Init	Instantiates a new instance of a `TSortedCollection`
581	Insert	Insert new item in collection.
578	KeyOf	Return the key of an item
578	Load	Instantiates a new instance of a `TSortedCollection` and loads it from stream.
580	Search	Search for item with given key.
582	Store	Write the collection to the stream.

67.13.3 TSortedCollection.Init

Synopsis: Instantiates a new instance of a `TSortedCollection`

Declaration:
```
constructor Init(ALimit: Sw_Integer;ADelta: Sw_Integer)
```

Visibility: default

Description: `Init` calls the inherited constuctor (see TCollection.Init (544)) and sets the `Duplicates` flag to false.

You should not call this method directly, since `TSortedCollection` is a abstract class. Instead, the descendent classes should call it via the `inherited` keyword.

Errors: None.

See also: TSortedCollection.Load (578), TCollection.Done (545)

67.13.4 TSortedCollection.Load

Synopsis: Instantiates a new instance of a `TSortedCollection` and loads it from stream.

Declaration:
```
constructor Load(var S: TStream)
```

Visibility: default

Description: `Load` calls the inherited constuctor (see TCollection.Load (545)) and reads the `Duplicates` flag from the stream..

You should not call this method directly, since `TSortedCollection` is a abstract class. Instead, the descendent classes should call it via the `inherited` keyword.

For an example, see TCollection.Load (545).

Errors: None.

See also: TSortedCollection.Init (578), TCollection.Done (545)

67.13.5 TSortedCollection.KeyOf

Synopsis: Return the key of an item

Declaration:
```
function KeyOf(Item: Pointer) : Pointer;  Virtual
```

Visibility: default

Description: `KeyOf` returns the key associated with `Item`. `TSortedCollection` returns the item itself as the key, descendent objects can override this method to calculate a (unique) key based on the item passed (such as hash values).

`Keys` are used to sort the objects, they are used to search and sort the items in the collection. If descendent types override this method then it allows possibly for faster search/sort methods based on keys rather than on the objects themselves.

Errors: None.

See also: TSortedCollection.IndexOf (579), TSortedCollection.Compare (579)

67.13.6 TSortedCollection.IndexOf

Synopsis: Return index of an item in the collection.

Declaration: `function IndexOf(Item: Pointer) : Sw_Integer;  Virtual`

Visibility: default

Description: `IndexOf` returns the index of `Item` in the collection. It searches for the object based on it's key. If duplicates are allowed, then it returns the index of last object that matches `Item`.

In case `Item` is not found in the collection, -1 is returned.

For an example, see TCollection.IndexOf (546)

Errors: None.

See also: TSortedCollection.Search (580), TSortedCollection.Compare (579)

67.13.7 TSortedCollection.Compare

Synopsis: Compare two items in the collection.

Declaration: `function Compare(Key1: Pointer;Key2: Pointer) : Sw_Integer;  Virtual`

Visibility: default

Description: `Compare` is an abstract method that should be overridden by descendent objects in order to compare two items in the collection. This method is used in the Search (580) method and in the Insert (581) method to determine the ordering of the objects.

The function should compare the two keys of items and return the following function results:

Result < 0 If `Key1` is logically before `Key2` (`Key1<Key2`)

Result = 0 If `Key1` and `Key2` are equal. (`Key1=Key2`)

Result > 0 If `Key1` is logically after `Key2` (`Key1>Key2`)

Errors: An 'abstract run-time error' will be generated if you call `TSortedCollection.Compare` directly.

See also: TSortedCollection.IndexOf (579), TSortedCollection.Search (580)

Listing: ./objectex/mysortc.pp

```
Unit MySortC;

Interface

Uses Objects;

Type
  PMySortedCollection = ^TMySortedCollection;
  TMySortedCollection = Object(TSortedCollection)
      Function Compare (Key1,Key2 : Pointer): Sw_integer; virtual;
      end;

Implementation

Uses MyObject;

Function TMySortedCollection.Compare (Key1,Key2 : Pointer) :sw_integer;

begin
  Compare:=PMyobject(Key1)^.GetField - PMyObject(Key2)^.GetField;
end;

end.
```

67.13.8 TSortedCollection.Search

Synopsis: Search for item with given key.

Declaration: `function Search(Key: Pointer;var Index: Sw_Integer) : Boolean;  Virtual`

Visibility: default

Description: `Search` looks for the item with key `Key` and returns the position of the item (if present) in the collection in `Index`.

Instead of a linear search as `TCollection` does, `TSortedCollection` uses a binary search based on the keys of the objects. It uses the Compare (579) function to implement this search.

If the item is found, `Search` returns `True`, otherwise `False` is returned.

Errors: None.

See also: TCollection.IndexOf (546)

Listing: ./objectex/ex36.pp

```
Program ex36;

{ Program to demonstrate the TSortedCollection.Insert method }

Uses Objects,MyObject,MySortC;
 { For TMyObject,TMySortedCollection definition and registration }

Var C : PSortedCollection;
    M : PMyObject;
    I : Longint;

Procedure PrintField (Dummy: Pointer;P : PMyObject);
```

```
begin
  WriteIn ('Field : ',P^.GetField);
end;


begin
  Randomize;
  C:=New(PMySortedCollection,Init(120,10));
  C^.Duplicates:=True;
  WriteIn ('Inserting 100 records at random places.');
  For I:=1 to 100 do
    begin
    M:=New(PMyObject,Init);
    M^.SetField(Random(100));
    C^.Insert(M)
    end;
  M:=New(PMyObject,Init);
  Repeat;
    Write ('Value to search for (-1 stops) :');
    read (I);
    If I<>-1 then
      begin
      M^.SetField(i);
      If Not C^.Search (M,I) then
        WriteIn ('No such value found')
      else
        begin
        Write ('Value ',PMyObject(C^.At(I))^.GetField);
        WriteIn (' present at position ',I);
        end;
      end;
  Until I=-1;
  Dispose(M,Done);
  Dispose(C,Done);
end.
```

67.13.9 TSortedCollection.Insert

Synopsis: Insert new item in collection.

Declaration: `procedure Insert(Item: Pointer);  Virtual`

Visibility: default

Description: `Insert` inserts an item in the collection at the correct position, such that the collection is ordered at all times. You should never use Atinsert (557), since then the collection ordering is not guaranteed.

If `Item` is already present in the collection, and `Duplicates` is `False`, the item will not be inserted.

Errors: None.

See also: TCollection.AtInsert (557)

Listing: ./objectex/ex35.pp

```
Program ex35;

{ Program to demonstrate the TSortedCollection.Insert method }

Uses Objects,MyObject,MySortC;
 { For TMyObject,TMySortedCollection definition and registration }

Var C : PSortedCollection;
    M : PMyObject;
    I : Longint;

Procedure PrintField (Dummy: Pointer;P : PMyObject);

begin
  Writeln ('Field : ',P^.GetField);
end;


begin
  Randomize;
  C:=New(PMySortedCollection,Init(120,10));
  Writeln ('Inserting 100 records at random places.');
  For I:=1 to 100 do
    begin
    M:=New(PMyObject,Init);
    M^.SetField(Random(100));
    C^.Insert(M)
    end;
  Writeln ('Values : ');
  C^.Foreach(@PrintField);
  Dispose(C,Done);
end.
```

67.13.10 TSortedCollection.Store

Synopsis: Write the collection to the stream.

Declaration: `procedure Store(var S: TStream)`

Visibility: default

Description: `Store` writes the collection to the stream `S`. It does this by calling the inherited TCollection.Store (558), and then writing the `Duplicates` flag to the stream.

After a `Store`, the collection can be loaded from the stream with the constructor Load (578)

For an example, see TCollection.Load (545).

Errors: Errors can be those of TStream.Put (590).

See also: TSortedCollection.Load (578)

67.14 TStrCollection

67.14.1 Description

The `TStrCollection` object manages a sorted collection of null-terminated strings (pchar strings). To this end, it overrides the Compare (579) method of `TSortedCollection`, and it introduces methods to read/write strings from a stream.

67.14.2 Method overview

Page	Method	Description
583	Compare	Compare two strings in the collection.
584	FreeItem	Free null-terminated string from the collection.
584	GetItem	Read a null-terminated string from the stream.
584	PutItem	Write a null-terminated string to the stream.

67.14.3 TStrCollection.Compare

Synopsis: Compare two strings in the collection.

Declaration: `function Compare(Key1: Pointer;Key2: Pointer) : Sw_Integer;  Virtual`

Visibility: default

Description: `TStrCollection` overrides the `Compare` function so it compares the two keys as if they were pointers to strings. The compare is done case sensitive. It returns

-1if the first string is alphabetically earlier than the second string.

0if the two strings are equal.

1if the first string is alphabetically later than the second string.

Errors: None.

See also: TSortedCollection.Compare (579)

Listing: ./objectex/ex38.pp

```
Program ex38;

{ Program to demonstrate the TStrCollection.Compare method }

Uses Objects,Strings;

Var C : PStrCollection;
    S : String;
    I : longint;
    P : Pchar;

begin
  Randomize;
  C:=New(PStrCollection,Init(120,10));
  C^.Duplicates:=True; { Duplicates allowed }
  Writeln ('Inserting 100 records at random places.');
  For I:=1 to 100 do
    begin
    Str(Random(100),S);
```

```
    S:='String with value '+S;
    P:=StrAlloc(Length(S)+1);
    C^.Insert(StrPCopy(P,S));
    end;
  For I:=0 to 98 do
    With C^ do
      If Compare (At(I),At(I+1))=0 then
        WriteIn ('Duplicate string found at position ',I);
  Dispose(C,Done);
end.
```

67.14.4 TStrCollection.GetItem

Synopsis: Read a null-terminated string from the stream.

Declaration: `function GetItem(var S: TStream) : Pointer;  Virtual`

Visibility: default

Description: `GetItem` reads a null-terminated string from the stream `S` and returns a pointer to it. It doesn't insert the string in the collection.

This method is primarily introduced to be able to load and store the collection from and to a stream.

Errors: The errors returned are those of TStream.StrRead (586).

See also: TStrCollection.PutItem (584)

67.14.5 TStrCollection.FreeItem

Synopsis: Free null-terminated string from the collection.

Declaration: `procedure FreeItem(Item: Pointer);  Virtual`

Visibility: default

Description: `TStrCollection` overrides `FreeItem` so that the string pointed to by `Item` is disposed from memory.

Errors: None.

See also: TCollection.FreeItem (554)

67.14.6 TStrCollection.PutItem

Synopsis: Write a null-terminated string to the stream.

Declaration: `procedure PutItem(var S: TStream;Item: Pointer);  Virtual`

Visibility: default

Description: `PutItem` writes the string pointed to by `Item` to the stream `S`.

This method is primarily used in the `Load` and `Store` methods, and should not be used directly.

Errors: Errors are those of TStream.StrWrite (591).

See also: TStrCollection.GetItem (584)

67.15 TStream

67.15.1 Description

The `TStream` object is the ancestor for all streaming objects, i.e. objects that have the capability to store and retrieve data.

It defines a number of methods that are common to all objects that implement streaming, many of them are virtual, and are only implemented in the descendent types.

Programs should not instantiate objects of type TStream directly, but instead instantiate a descendant type, such as `TDosStream`, `TMemoryStream`.

See also: PStream (531), TDosStream (558), TMemoryStream (563)

67.15.2 Method overview

Page	Method	Description
589	Close	Close the stream
593	CopyFrom	Copy data from another stream.
591	Error	Set stream status
590	Flush	Flush the stream data from the buffer, if any.
585	Get	Read an object definition from the stream.
587	GetPos	Return current position in the stream
587	GetSize	Return the size of the stream.
585	Init	Constructor for TStream instance
589	Open	Open the stream
590	Put	Write an object to the stream.
592	Read	Read data from stream to buffer.
588	ReadStr	Read a shortstring from the stream.
589	Reset	Reset the stream
591	Seek	Set stream position.
586	StrRead	Read a null-terminated string from the stream.
591	StrWrite	Write a null-terminated string to the stream.
590	Truncate	Truncate the stream size on current position.
592	Write	Write a number of bytes to the stream.
591	WriteStr	Write a pascal string to the stream.

67.15.3 TStream.Init

Synopsis: Constructor for TStream instance

Declaration: `constructor Init`

Visibility: default

Description: `Init` initializes a TStream instance. Descendent streams should always call the inherited `Init`.

67.15.4 TStream.Get

Synopsis: Read an object definition from the stream.

Declaration: `function Get : PObject`

Visibility: default

Description: `Get` reads an object definition from a stream, and returns a pointer to an instance of this object.

Errors: On error, TStream.Status (**??**) is set, and `NIL` is returned.

See also: TStream.Put (590)

Listing: ./objectex/ex9.pp

```
Program ex9;

{ Program to demonstrate TStream.Get and TStream.Put }

Uses Objects,MyObject;  { Definition and registration of TMyObject}

Var Obj : PMyObject;
    S : PStream;

begin
  Obj:=New(PMyObject,Init);
  Obj^.SetField($1111);
  Writeln('Field value : ',Obj^.GetField);
  { Since Stream is an abstract type, we instantiate a TMemoryStream }
  S:=New(PMemoryStream,Init(100,10));
  S^.Put(Obj);
  Writeln('Disposing object');
  S^.Seek(0);
  Dispose(Obj,Done);
  Writeln('Reading object');
  Obj:=PMyObject(S^.Get);
  Writeln('Field Value : ',Obj^.GetField);
  Dispose(Obj,Done);
end.
```

67.15.5 TStream.StrRead

Synopsis: Read a null-terminated string from the stream.

Declaration: `function StrRead : PChar`

Visibility: default

Description: `StrRead` reads a string from the stream, allocates memory for it, and returns a pointer to a null-terminated copy of the string on the heap.

Errors: On error, `Nil` is returned.

See also: TStream.StrWrite (591), TStream.ReadStr (588)

Listing: ./objectex/ex10.pp

```
Program ex10;

{
Program to demonstrate the TStream.StrRead TStream.StrWrite functions
}

Uses objects;
```

```
Var P : PChar;
    S : PStream;

begin
  P:='Constant Pchar string';
  Writeln ('Writing to stream : "',P,'"');
  S:=New(PMemoryStream,Init(100,10));
  S^.StrWrite(P);
  S^.Seek(0);
  P:=Nil;
  P:=S^.StrRead;
  DisPose (S,Done);
  Writeln ('Read from stream : "',P,'"');
  Freemem(P,Strlen(P)+1);
end.
```

67.15.6 TStream.GetPos

Synopsis: Return current position in the stream

Declaration: `function GetPos : LongInt;  Virtual`

Visibility: default

Description: If the stream's status is `stOk`, `GetPos` returns the current position in the stream. Otherwise it returns `-1`

Errors: `-1` is returned if the status is an error condition.

See also: TStream.Seek (591), TStream.GetSize (587)

Listing: ./objectex/ex11.pp

```
Program ex11;

{ Program to demonstrate the TStream.GetPos function }

Uses objects;

Var L : String;
    S : PStream;

begin
  L:='Some kind of string';
  S:=New(PMemoryStream,Init(100,10));
  Writeln ('Stream position before write: ',S^.GetPos);
  S^.WriteStr(@L);
  Writeln ('Stream position after write : ',S^.GetPos);
  Dispose(S,Done);
end.
```

67.15.7 TStream.GetSize

Synopsis: Return the size of the stream.

Declaration: `function GetSize : LongInt;  Virtual`

Visibility: default

Description: If the stream's status is `stOk` then `GetSize` returns the size of the stream, otherwise it returns `-1`.

Errors: `-1` is returned if the status is an error condition.

See also: TStream.Seek (591), TStream.GetPos (587)

Listing: ./objectex/ex12.pp

```
Program ex12;

{ Program to demonstrate the TStream.GetSize function }

Uses objects;

Var L : String;
    S : PStream;

begin
  L:='Some kind of string';
  S:=New(PMemoryStream,Init(100,10));
  Writeln ('Stream size before write: ',S^.GetSize);
  S^.WriteStr(@L);
  Writeln ('Stream size after write : ',S^.GetSize);
  Dispose(S,Done);
end.
```

67.15.8 TStream.ReadStr

Synopsis: Read a shortstring from the stream.

Declaration: `function ReadStr : PString`

Visibility: default

Description: `ReadStr` reads a string from the stream, copies it to the heap and returns a pointer to this copy. The string is saved as a pascal string, and hence is NOT null terminated.

Errors: On error (e.g. not enough memory), `Nil` is returned.

See also: TStream.StrRead (586)

Listing: ./objectex/ex13.pp

```
Program ex13;

{
Program to demonstrate the TStream.ReadStr TStream.WriteStr functions
}

Uses objects;

Var P : PString;
    L : String;
    S : PStream;

begin
```

```
  L:='Constant string line';
  Writeln ('Writing to stream : "',L,'"');
  S:=New(PMemoryStream,Init(100,10));
  S^.WriteStr(@L);
  S^.Seek(0);
  P:=S^.ReadStr;
  L:=P^;
  DisposeStr(P);
  DisPose (S,Done);
  Writeln ('Read from stream : "',L,'"');
end.
```

67.15.9 TStream.Open

Synopsis: Open the stream

Declaration: `procedure Open(OpenMode: Word);  Virtual`

Visibility: default

Description: `Open` is an abstract method, that should be overridden by descendent objects. Since opening a stream depends on the stream's type this is not surprising.

For an example, see TDosStream.Open (562).

Errors: None.

See also: TStream.Close (589), TStream.Reset (589)

67.15.10 TStream.Close

Synopsis: Close the stream

Declaration: `procedure Close;  Virtual`

Visibility: default

Description: `Close` is an abstract method, that should be overridden by descendent objects. Since Closing a stream depends on the stream's type this is not surprising.

for an example, see TDosStream.Open (562).

Errors: None.

See also: TStream.Open (589), TStream.Reset (589)

67.15.11 TStream.Reset

Synopsis: Reset the stream

Declaration: `procedure Reset`

Visibility: default

Description: `Reset` sets the stream's status to `0`, as well as the ErrorInfo

Errors: None.

See also: TStream.Open (589), TStream.Close (589)

67.15.12 TStream.Flush

Synopsis: Flush the stream data from the buffer, if any.

Declaration: `procedure Flush; Virtual`

Visibility: default

Description: `Flush` is an abstract method that should be overridden by descendent objects. It serves to enable the programmer to tell streams that implement a buffer to clear the buffer.

for an example, see TBufStream.Flush (541).

Errors: None.

See also: TStream.Truncate (590)

67.15.13 TStream.Truncate

Synopsis: Truncate the stream size on current position.

Declaration: `procedure Truncate; Virtual`

Visibility: default

Description: `Truncate` is an abstract procedure that should be overridden by descendent objects. It serves to enable the programmer to truncate the size of the stream to the current file position.

For an example, see TDosStream.Truncate (560).

Errors: None.

See also: TStream.Seek (591)

67.15.14 TStream.Put

Synopsis: Write an object to the stream.

Declaration: `procedure Put(P: PObject)`

Visibility: default

Description: `Put` writes the object pointed to by `P.P` should be non-nil. The object type must have been registered with RegisterType (538).

After the object has been written, it can be read again with Get (585).

For an example, see TStream.Get (585);

Errors: No check is done whether P is `Nil` or not. Passing `Nil` will cause a run-time error 216 to be generated. If the object has not been registered, the status of the stream will be set to `stPutError`.

See also: TStream.Get (585)

67.15.15 TStream.StrWrite

Synopsis: Write a null-terminated string to the stream.

Declaration: `procedure StrWrite(P: PChar)`

Visibility: default

Description: `StrWrite` writes the null-terminated string `P` to the stream. `P` can only be 65355 bytes long.

For an example, see TStream.StrRead (586).

Errors: None.

See also: TStream.WriteStr (591), TStream.StrRead (586), TStream.ReadStr (588)

67.15.16 TStream.WriteStr

Synopsis: Write a pascal string to the stream.

Declaration: `procedure WriteStr(P: PString)`

Visibility: default

Description: `StrWrite` writes the pascal string pointed to by `P` to the stream.

For an example, see TStream.ReadStr (588).

Errors: None.

See also: TStream.StrWrite (591), TStream.StrRead (586), TStream.ReadStr (588)

67.15.17 TStream.Seek

Synopsis: Set stream position.

Declaration: `procedure Seek(Pos: LongInt);  Virtual`

Visibility: default

Description: Seek sets the position to `Pos`. This position is counted from the beginning, and is zero based. (i.e. seeek(0) sets the position pointer on the first byte of the stream)

For an example, see TDosStream.Seek (561).

Errors: If `Pos` is larger than the stream size, `Status` is set to `StSeekError`.

See also: TStream.GetPos (587), TStream.GetSize (587)

67.15.18 TStream.Error

Synopsis: Set stream status

Declaration: `procedure Error(Code: Integer;Info: Integer);  Virtual`

Visibility: default

Description: `Error` sets the stream's status to `Code` and `ErrorInfo` to `Info`. If the `StreamError` procedural variable is set, `Error` executes it, passing `Self` as an argument.

This method should not be called directly from a program. It is intended to be used in descendent objects.

Errors: None.

67.15.19 TStream.Read

Synopsis: Read data from stream to buffer.

Declaration: `procedure Read(var Buf;Count: LongInt); Virtual`

Visibility: default

Description: `Read` is an abstract method that should be overridden by descendent objects.

`Read` reads `Count` bytes from the stream into `Buf`. It updates the position pointer, increasing it's value with `Count`. `Buf` must be large enough to contain `Count` bytes.

Errors: No checking is done to see if `Buf` is large enough to contain `Count` bytes.

See also: TStream.Write (592), TStream.ReadStr (588), TStream.StrRead (586)

Listing: ./objectex/ex18.pp

```
program ex18;

{ Program to demonstrate the TStream.Read method }

Uses Objects;

Var Buf1,Buf2 : Array[1..1000] of Byte;
    I : longint;
    S : PMemorySTream;

begin
  For I:=1 to 1000 do
    Buf1[I]:=Random(1000);
  Buf2:=Buf1;
  S:=New(PMemoryStream,Init(100,10));
  S^.Write(Buf1,SizeOf(Buf1));
  S^.Seek(0);
  For I:=1 to 1000 do
    Buf1[I]:=0;
  S^.Read(Buf1,SizeOf(Buf1));
  For I:=1 to 1000 do
    If Buf1[I]<>buf2[i] then
      WriteIn ('Buffer differs at position ',I);
  Dispose(S,Done);
end.
```

67.15.20 TStream.Write

Synopsis: Write a number of bytes to the stream.

Declaration: `procedure Write(var Buf;Count: LongInt); Virtual`

Visibility: default

Description: `Write` is an abstract method that should be overridden by descendent objects.

`Write` writes `Count` bytes to the stream from `Buf`. It updates the position pointer, increasing it's value with `Count`.

For an example, see TStream.Read (592).

Errors: No checking is done to see if `Buf` actually contains `Count` bytes.

See also: TStream.Read (592), TStream.WriteStr (591), TStream.StrWrite (591)

67.15.21 TStream.CopyFrom

Synopsis: Copy data from another stream.

Declaration: `procedure CopyFrom(var S: TStream;Count: LongInt)`

Visibility: default

Description: `CopyFrom` reads Count bytes from stream `S` and stores them in the current stream. It uses the Read (592) method to read the data, and the Write (592) method to write in the current stream.

Errors: None.

See also: Read (592), Write (592)

Listing: ./objectex/ex19.pp

```
Program ex19;

{ Program to demonstrate the TStream.CopyFrom function }

Uses objects;

Var P : PString;
    L : String;
    S1,S2 : PStream;

begin
  L:='Constant string line';
  Writeln ('Writing to stream 1 : "',L,'"');
  S1:=New(PMemoryStream,Init(100,10));
  S2:=New(PMemoryStream,Init(100,10));
  S1^.WriteStr(@L);
  S1^.Seek(0);
  Writeln ('Copying contents of stream 1 to stream 2');
  S2^.Copyfrom(S1^,S1^.GetSize);
  S2^.Seek(0);
  P:=S2^.ReadStr;
  L:=P^;
  DisposeStr(P);
  Dispose (S1,Done);
  Dispose (S2,Done);
  Writeln ('Read from stream 2 : "',L,'"');
end.
```

67.16 TStringCollection

67.16.1 Description

The `TStringCollection` object manages a sorted collection of pascal strings. To this end, it overrides the Compare (579) method of `TSortedCollection`, and it introduces methods to read/write strings from a stream.

67.16.2 Method overview

Page	Method	Description
594	Compare	Compare two strings in the collection.
595	FreeItem	Dispose a string in the collection from memory.
594	GetItem	Get string from the stream.
595	PutItem	Write a string to the stream.

67.16.3 TStringCollection.GetItem

Synopsis: Get string from the stream.

Declaration: `function GetItem(var S: TStream) : Pointer;  Virtual`

Visibility: default

Description: `GetItem` reads a string from the stream `S` and returns a pointer to it. It doesn't insert the string in the collection.

This method is primarily introduced to be able to load and store the collection from and to a stream.

Errors: The errors returned are those of TStream.ReadStr (588).

See also: TStringCollection.PutItem (595)

67.16.4 TStringCollection.Compare

Synopsis: Compare two strings in the collection.

Declaration: `function Compare(Key1: Pointer;Key2: Pointer) : Sw_Integer;  Virtual`

Visibility: default

Description: `TStringCollection` overrides the `Compare` function so it compares the two keys as if they were pointers to strings. The compare is done case sensitive. It returns the following results:

-1 if the first string is alphabetically earlier than the second string.

0 if the two strings are equal.

1 if the first string is alphabetically later than the second string.

Errors: None.

See also: TSortedCollection.Compare (579)

Listing: ./objectex/ex37.pp

```
Program ex37;

{ Program to demonstrate the TStringCollection.Compare method }

Uses Objects;

Var C : PStringCollection;
    S : String;
    I : longint;

begin
  Randomize;
```

```
  C:=New(PStringCollection,Init(120,10));
  C^.Duplicates:=True; { Duplicates allowed }
  WriteLn ('Inserting 100 records at random places.');
  For I:=1 to 100 do
    begin
    Str(Random(100),S);
    S:='String with value '+S;
    C^.Insert(NewStr(S));
    end;
  For I:=0 to 98 do
    With C^ do
    If Compare (At(i),At(I+1))=0 then
      WriteLn ('Duplicate string found at position ',i);
  Dispose(C,Done);
end.
```

67.16.5 TStringCollection.FreeItem

Synopsis: Dispose a string in the collection from memory.

Declaration: `procedure FreeItem(Item: Pointer);  Virtual`

Visibility: default

Description: `TStringCollection` overrides `FreeItem` so that the string pointed to by `Item` is disposed from memory.

Errors: None.

See also: TCollection.FreeItem (554)

67.16.6 TStringCollection.PutItem

Synopsis: Write a string to the stream.

Declaration: `procedure PutItem(var S: TStream;Item: Pointer);  Virtual`

Visibility: default

Description: `PutItem` writes the string pointed to by `Item` to the stream `S`.

This method is primarily used in the `Load` and `Store` methods, and should not be used directly.

Errors: Errors are those of TStream.WriteStr (591).

See also: TStringCollection.GetItem (594)

67.17 TStringList

67.17.1 Description

A `TStringList` object can be used to read a collection of strings stored in a stream. If you register this object with the RegisterType (538) function, you cannot register the `TStrListMaker` object.

67.17.2 Method overview

Page	Method	Description
596	Done	Clean up the instance
596	Get	Return a string by key name
596	Load	Load stringlist from stream.

67.17.3 TStringList.Load

Synopsis: Load stringlist from stream.

Declaration: `constructor Load(var S: TStream)`

Visibility: default

Description: The `Load` constructor reads the `TStringList` object from the stream `S`. It also reads the descriptions of the strings from the stream. The string descriptions are stored as an array of `TstrIndexrec` records, where each record describes a string on the stream. These records are kept in memory.

Errors: If an error occurs, a stream error is triggered.

See also: TStringList.Done (596)

67.17.4 TStringList.Done

Synopsis: Clean up the instance

Declaration: `destructor Done;  Virtual`

Visibility: default

Description: The `Done` destructor frees the memory occupied by the string descriptions, and destroys the object.

Errors: None.

See also: Load (596), TObject.Done (567)

67.17.5 TStringList.Get

Synopsis: Return a string by key name

Declaration: `function Get(Key: Sw_Word) : string`

Visibility: default

Description: `Get` reads the string with key `Key` from the list of strings on the stream, and returns this string. If there is no string with such a key, an empty string is returned.

Errors: If no string with key `Key` is found, an empty string is returned. A stream error may result if the stream doesn't contain the needed strings.

See also: TStrListMaker.Put (597)

67.18 TStrListMaker

67.18.1 Description

The `TStrListMaker` object can be used to generate a stream with strings, which can be read with the `TStringList` object. If you register this object with the RegisterType (538) function, you cannot register the `TStringList` object.

67.18.2 Method overview

Page	Method	Description
597	Done	Clean up the instance and free all related memory.
597	Init	Instantiate a new instance of `TStrListMaker`
597	Put	Add a new string to the list with associated key.
598	Store	Write the strings to the stream.

67.18.3 TStrListMaker.Init

Synopsis: Instantiate a new instance of `TStrListMaker`

Declaration: `constructor Init(AStrSize: Sw_Word;AIndexSize: Sw_Word)`

Visibility: default

Description: The `Init` constructor creates a new instance of the `TstrListMaker` object. It allocates `AStrSize` bytes on the heap to hold all the strings you wish to store. It also allocates enough room for `AIndexSize` key description entries (of the type `TStrIndexrec`).

`AStrSize` must be large enough to contain all the strings you wish to store. If not enough memory is allocated, other memory will be overwritten. The same is true for `AIndexSize` : maximally `AIndexSize` strings can be written to the stream.

Errors: None.

See also: TObject.Init (565), TStrListMaker.Done (597)

67.18.4 TStrListMaker.Done

Synopsis: Clean up the instance and free all related memory.

Declaration: `destructor Done;  Virtual`

Visibility: default

Description: The `Done` destructor de-allocates the memory for the index description records and the string data, and then destroys the object.

Errors: None.

See also: TObject.Done (567), TStrListMaker.Init (597)

67.18.5 TStrListMaker.Put

Synopsis: Add a new string to the list with associated key.

Declaration: `procedure Put(Key: Sw_Word;S: string)`

Visibility: default

Description: `Put` adds they string `S` with key `Key` to the collection of strings. This action doesn't write the string to a stream. To write the strings to the stream, see the Store (598) method.

Errors: None.

See also: TStrListMaker.Store (598)

67.18.6 TStrListMaker.Store

Synopsis: Write the strings to the stream.

Declaration: `procedure Store(var S: TStream)`

Visibility: default

Description: `Store` writes the collection of strings to the stream `S`. The collection can then be read with the `TStringList` object.

Errors: A stream error may occur when writing the strings to the stream.

See also: TStringList.Load (596), TStrListMaker.Put (597)

67.19 TUnSortedStrCollection

67.19.1 Description

The `TUnSortedStrCollection` object manages an unsorted list of strings. To this end, it overrides the TSortedCollection.Insert (581) method to add strings at the end of the collection, rather than in the alphabetically correct position.

Take care, the Search (580) and IndexOf (546) methods will not work on an unsorted string collection.

67.19.2 Method overview

Page	Method	Description
598	Insert	Insert a new string in the collection.

67.19.3 TUnSortedStrCollection.Insert

Synopsis: Insert a new string in the collection.

Declaration: `procedure Insert(Item: Pointer); Virtual`

Visibility: default

Description: `Insert` inserts a string at the end of the collection, instead of on it's alphabetical place, resulting in an unsorted collection of strings.

Errors: None.

See also: TCollection.Insert (552)

Listing: ./objectex/ex39.pp

```
Program ex39;

{ Program to demonstrate the TUnsortedStrCollection.Insert method }

Uses Objects,Strings;

Var C : PUnsortedStrCollection;
    S : String;
    I : longint;
    P : Pchar;

begin
  Randomize;
  C:=New(PUnsortedStrCollection,Init(120,10));
  Writeln ('Inserting 100 records at random places.');
  For I:=1 to 100 do
    begin
    Str(Random(100),S);
    S:='String with value '+S;
    C^.Insert(NewStr(S));
    end;
  For I:=0 to 99 do
    Writeln (I:2,': ',PString(C^.At(i))^ );
  Dispose(C,Done);
end.
```

Chapter 68

Reference for unit 'objpas'

68.1 Overview

The objpas unit is meant for compatibility with Object Pascal as implemented by Delphi. The unit is loaded automatically by the Free Pascal compiler whenever the `Delphi` or `objfpc` mode is entered, either through the command line switches `-Sd` or `-Sh` or with the `{$MODE DELPHI}` or `{$MODE OBJFPC}` directives.

It redefines some basic pascal types, introduces some functions for compatibility with Delphi's system unit, and introduces some methods for the management of the resource string tables.

68.2 Constants, types and variables

68.2.1 Constants

```
MaxInt = MaxLongint
```

Maximum value for Integer (600) type.

68.2.2 Types

```
Integer = LongInt
```

In `OBJPAS` mode and in `DELPHI` mode, an `Integer` has a size of 32 bit. In TP or regular FPC mode, an integer is 16 bit.

```
IntegerArray = Array[0..$efffffff] of Integer
```

Generic array of integer (600)

```
PInteger = ^Integer
```

Pointer to Integer (600) type.

```
PIntegerArray = ^IntegerArray
```

Pointer to TIntegerArray (601) type.

```
PointerArray = Array[0..512*1024*1024-2] of Pointer
```

Generic Array of pointers.

```
PPointerArray = ^PointerArray
```

Pointer to PointerArray (601)

```
PString = PAnsiString
```

Pointer to ansistring type.

```
TBoundArray = Array of Integer
```

Array of integer, used in interfaces.

```
TIntegerArray = IntegerArray
```

Alias for IntegerArray (600)

```
TPointerArray = PointerArray
```

Alias for PointerArray (601)

Chapter 69

Reference for unit 'ports'

69.1 Overview

The ports unit implements the `port` constructs found in Turbo Pascal. It uses classes and default array properties to do this.

The unit exists on linux, os/2 and dos. It is implemented only for compatibility with Turbo Pascal. It's usage is discouraged, because using ports is not portable programming, and the operating system may not even allow it (for instance Windows).

Under linux, your program must be run as root, or the `IOPerm` call must be set in order to set appropriate permissions on the port access.

69.2 Constants, types and variables

69.2.1 Variables

`port : tport`

Default instance of type TPort (603). Do not free. This variable is initialized in the unit initialization code, and freed at finalization.

Since there is a default property for a variable of this type, a sentence as

```
port[221]:=12;
```

Will result in the integer 12 being written to port 221, if port is defined as a variable of type `tport`

`portb : tport`

Default instance of type TPort (603). Do not free. This variable is initialized in the unit initialization code, and freed at finalization.

Since there is a default property for a variable of this type, a sentence as

```
portb[221]:=12;
```

Will result in the byte 12 being written to port 221, if port is defined as a variable of type `tport`

```
portl : tportl
```

Default instance of type TPortL (604). Do not free. This variable is initialized in the unit initialization code, and freed at finalization.

Since there is a default property for a variable of this type, a sentence as

```
portl[221]:=12;
```

Will result in the longint 12 being written to port 221, if port is defined as a variable of type `tport`

```
portw : tportw
```

Default instance of type TPortW (604). Do not free. This variable is initialized in the unit initialization code, and freed at finalization.

Since there is a default property for a variable of this type, a sentence as

```
portw[221]:=12;
```

Will result in the word 12 being written to port 221, if port is defined as a variable of type `tport`

69.3 tport

69.3.1 Description

The `TPort` type is implemented specially for access to the ports in a TP compatible manner. There is no need to create an instance of this type: the standard TP variables are instantiated at unit initialization.

See also: port (602), TPortW (604), TPortL (604)

69.3.2 Property overview

Page	Properties	Access	Description
603	pp	rw	Access integer-sized port by port number

69.3.3 tport.pp

Synopsis: Access integer-sized port by port number

Declaration: `Property pp[w: LongInt]: Byte; default`

Visibility: public

Access: Read,Write

Description: Access integer-sized port by port number

69.4 tportl

69.4.1 Description

The `TPortL` type is implemented specially for access to the ports in a TP compatible manner. There is no need to create an instance of this type: the standard TP variables are instantiated at unit initialization.

See also: portw (603), TPort (603), TPortL (604)

69.4.2 Property overview

Page	Properties	Access	Description
604	pp	rw	Access Longint-sized port by port number

69.4.3 tportl.pp

Synopsis: Access Longint-sized port by port number

Declaration: `Property pp[w: LongInt]: LongInt; default`

Visibility: public

Access: Read,Write

Description: Access Longint-sized port by port number

69.5 tportw

69.5.1 Description

The `TPortW` type is implemented specially for access to the ports in a TP compatible manner. There is no need to create an instance of this type: the standard TP variables are instantiated at unit initialization.

See also: portw (603), TPort (603), TPortL (604)

69.5.2 Property overview

Page	Properties	Access	Description
604	pp	rw	Access word-sized port by port number

69.5.3 tportw.pp

Synopsis: Access word-sized port by port number

Declaration: `Property pp[w: LongInt]: Word; default`

Visibility: public

Access: Read,Write

Description: Access word-sized port by port number

Chapter 70

Reference for unit 'printer'

70.1 Overview

This chapter describes the printer unit for Free Pascal. It was written for DOS by Florian Klaempfl, and it was written for Linux by Michael Van Canneyt, and has been ported to Windows and OS/2 as well. Its basic functionality is the same for all supported systems, although there are minor differences on Linux and UNIX.

Chapter 71

Reference for unit 'sharemem'

71.1 Overview

sharemem implements a shared memory manager. Including this unit will replace the standard memory manager with a memory manager which uses shared memory. This means the memory allocated by this unit can be managed by a program and a DLL if they both use the shared memory manager: it allows, amongst other things, to pass ansistrings or unicode strings from a program to a DLL and vice versa.

This unit does not implement any routines: all actions to replace the memory manager are performed in the initialization section of the unit. The unit should be placed as the first unit in a program or DLL's uses section, memory corruption may occur if the unit is not placed first.

This unit requires the fpcmemdll.dll library to be distributed with both program and dll that use this unit. This DLL is distributed with the windows Free Pascal distribution.

Chapter 72

Reference for unit 'Sockets'

72.1 Used units

Table 72.1: Used units by unit 'Sockets'

Name	Page
BaseUnix	51
System	622
unixtype	747

72.2 Overview

This document describes the SOCKETS unit for Free Pascal. it was written for linux by Michael Van Canneyt, and ported to Windows by Florian Klaempfl.

Chapter 73

Reference for unit 'strings'

73.1 Overview

This chapter describes the `STRINGS` unit for Free Pascal. This unit is system independent, and therefore works on all supported platforms.

73.2 Procedures and functions

73.2.1 stralloc

Synopsis: Allocate memory for a new null-terminated string on the heap

Declaration: `function stralloc(L: SizeInt) : pchar`

Visibility: default

Description: `StrAlloc` reserves memory on the heap for a string with length `Len`, terminating `#0` included, and returns a pointer to it.

Errors: If there is not enough memory, a run-time error occurs.

See also: StrNew (617), StrPCopy (618)

73.2.2 strcat

Synopsis: Concatenate 2 null-terminated strings.

Declaration: `function strcat(dest: pchar;source: pchar) : pchar`

Visibility: default

Description: Attaches `Source` to `Dest` and returns `Dest`.

Errors: No length checking is performed.

See also: StrLCat (613)

Listing: ./stringex/ex11.pp

```
Program Example11;

Uses strings;

{ Program to demonstrate the StrCat function. }

Const P1 : PChar = 'This is a PChar String.';

Var P2 : PChar;

begin
  P2:=StrAlloc (StrLen(P1)*2+1);
  StrMove (P2,P1,StrLen(P1)+1); { P2=P1 }
  StrCat (P2,P1);               { Append P2 once more }
  Writeln ('P2 : ',P2);
  StrDispose(P2);
end.
```

73.2.3 strcomp

Synopsis: Compare 2 null-terminated strings, case sensitive.

Declaration: `function strcomp(str1: pchar;str2: pchar) : SizeInt`

Visibility: default

Description: Compares the null-terminated strings `S1` and `S2`. The result is

- A negative `SizeInt` when `S1<S2`.
- 0 when `S1=S2`.
- A positive `SizeInt` when `S1>S2`.

For an example, see StrLComp (614).

Errors: None.

See also: StrLComp (614), StrIComp (612), StrLIComp (615)

73.2.4 strcopy

Synopsis: Copy a null-terminated string

Declaration: `function strcopy(dest: pchar;source: pchar) : pchar;  Overload`

Visibility: default

Description: Copy the null terminated string in `Source` to `Dest`, and returns a pointer to `Dest`. `Dest` needs enough room to contain `Source`, i.e. `StrLen(Source)+1` bytes.

Errors: No length checking is performed.

See also: StrPCopy (618), StrLCopy (614), StrECopy (610)

Listing: ./stringex/ex4.pp

```
Program Example4;

Uses strings;

{ Program to demonstrate the StrCopy function. }

Const P : PCHar = 'This is a PCHAR string.';

var PP : PChar;

begin
  PP:=StrAlloc(Strlen(P)+1);
  STrCopy (PP,P);
  If StrComp (PP,P)<>0 then
    WriteIn ('Oh-oh problems...')
  else
    WriteIn ('All is well : PP=',PP);
  StrDispose(PP);
end.
```

73.2.5 strdispose

Synopsis: disposes of a null-terminated string on the heap

Declaration: `procedure strdispose(p: pchar)`

Visibility: default

Description: Removes the string in `P` from the heap and releases the memory.

Errors: None.

See also: StrNew (617)

Listing: ./stringex/ex17.pp

```
Program Example17;

Uses strings;

{ Program to demonstrate the StrDispose function. }

Const P1 : PChar = 'This is a PChar string';

var P2 : PChar;

begin
  P2:=StrNew (P1);
  WriteIn ('P2 : ',P2);
  StrDispose(P2);
end.
```

73.2.6 strecopy

Synopsis: Copy a null-terminated string, return a pointer to the end.

Declaration: `function strecopy(dest: pchar;source: pchar) : pchar`

Visibility: default

Description: Copies the Null-terminated string in `Source` to `Dest`, and returns a pointer to the end (i.e. the terminating Null-character) of the copied string.

Errors: No length checking is performed.

See also: StrLCopy (614), StrCopy (609)

Listing: ./stringex/ex6.pp

```
Program Example6;

Uses strings;

{ Program to demonstrate the StrECopy function. }

Const P : PChar = 'This is a PCHAR string.';

Var PP : PChar;

begin
  PP:=StrAlloc (StrLen(P)+1);
  If Longint(StrECopy(PP,P))-Longint(PP)<>StrLen(P) then
    Writeln('Something is wrong here !')
  else
    Writeln ('PP= ',PP);
  StrDispose(PP);
end.
```

73.2.7 strend

Synopsis: Return a pointer to the end of a null-terminated string

Declaration: `function strend(p: pchar) : pchar`

Visibility: default

Description: Returns a pointer to the end of `P`. (i.e. to the terminating null-character.

Errors: None.

See also: StrLen (615)

Listing: ./stringex/ex7.pp

```
Program Example6;

Uses strings;

{ Program to demonstrate the StrEnd function. }

Const P : PChar = 'This is a PCHAR string.';

begin
  If Longint(StrEnd(P))-Longint(P)<>StrLen(P) then
    Writeln('Something is wrong here !')
```

```
  else
    WriteIn ('All is well..');
end.
```

73.2.8 stricomp

Synopsis: Compare 2 null-terminated strings, case insensitive.

Declaration: `function stricomp(str1: pchar;str2: pchar) : SizeInt`

Visibility: default

Description: Compares the null-terminated strings `S1` and `S2`, ignoring case. The result is

- A negative `SizeInt` when `S1<S2`.
- 0 when `S1=S2`.
- A positive `SizeInt` when `S1>S2`.

Errors: None.

See also: StrLComp (614), StrComp (609), StrLIComp (615)

Listing: ./stringex/ex8.pp

```
Program Example8;

Uses strings;

{ Program to demonstrate the StrLComp function. }

Const P1 : PChar = 'This is the first string.';
      P2 : PCHar = 'This is the second string.';

Var L : Longint;

begin
  Write ('P1 and P2 are ');
  If StrComp (P1,P2)<>0 then write ('NOT ');
  write ('equal. The first ');
  L:=1;
  While StrLComp(P1,P2,L)=0 do inc (L);
  dec(l);
  WriteIn (l,' characters are the same.');
end.
```

73.2.9 stripos

Synopsis: Return the position of a substring in a string, case insensitive.

Declaration: `function stripos(str1: pchar;str2: pchar) : pchar`

Visibility: default

Description: `stripos` returns the position of `str2` in `str1`. It searches in a case-insensitive manner, and if it finds a match, it returns a pointer to the location of the match. If no match is found, `Nil` is returned.

Errors: No checks are done on the validity of the pointers, and the pointers are assumed to point to a properly null-terminated string. If either of these conditions are not met, a run-time error may follow.

See also: striscan (613), strpos (619)

73.2.10 striscan

Synopsis: Scan a string for a character, case-insensitive

Declaration: `function striscan(p: pchar;c: Char) : pchar`

Visibility: default

Description: `striscan` does the same as strscan (620) but compares the characters case-insensitively. It returns a pointer to the first occurrence of the character `c` in the null-terminated string `p`, or `Nil` if `c` is not present in the string.

See also: strscan (620), strriscan (619)

73.2.11 strlcat

Synopsis: Concatenate 2 null-terminated strings, with length boundary.

Declaration: `function strlcat(dest: pchar;source: pchar;l: SizeInt) : pchar`

Visibility: default

Description: Adds `L` characters from `Source` to `Dest`, and adds a terminating null-character. Returns `Dest`.

Errors: None.

See also: StrCat (608)

Listing: ./stringex/ex12.pp

```
Program Example12;

Uses strings;

{ Program to demonstrate the StrLCat function. }

Const P1 : PChar = '1234567890';

Var P2 : PChar;

begin
  P2:=StrAlloc (StrLen(P1)*2+1);
  P2^:=#0; { Zero length }
  StrCat (P2,P1);
  StrLCat (P2,P1,5);
  WriteIn ('P2 = ',P2);
  StrDispose(P2)
end.
```

73.2.12 strlcomp

Synopsis: Compare limited number of characters of 2 null-terminated strings

Declaration: `function strlcomp(str1: pchar;str2: pchar;l: SizeInt) : SizeInt`

Visibility: default

Description: Compares maximum `L` characters of the null-terminated strings `S1` and `S2`. The result is

- A negative `SizeInt` when `S1<S2`.
- 0 when `S1=S2`.
- A positive `SizeInt` when `S1>S2`.

Errors: None.

See also: StrComp (609), StrIComp (612), StrLIComp (615)

Listing: ./stringex/ex8.pp

```
Program Example8;

Uses strings;

{ Program to demonstrate the StrLComp function. }

Const P1 : PChar = 'This is the first string.';
      P2 : PCHar = 'This is the second string.';

Var L : Longint;

begin
  Write ('P1 and P2 are ');
  If StrComp (P1,P2)<>0 then write ('NOT ');
  write ('equal. The first ');
  L:=1;
  While StrLComp(P1,P2,L)=0 do inc (L);
  dec(l);
  Writeln (l,' characters are the same.');
end.
```

73.2.13 strlcopy

Synopsis: Copy a null-terminated string, limited in length.

Declaration: `function strlcopy(dest: pchar;source: pchar;maxlen: SizeInt) : pchar ; Overload`

Visibility: default

Description: Copies `MaxLen` characters from `Source` to `Dest`, and makes `Dest` a null terminated string.

Errors: No length checking is performed.

See also: StrCopy (609), StrECopy (610)

Listing: ./stringex/ex5.pp

```
Program Example5;

Uses strings;

{ Program to demonstrate the StrLCopy function. }

Const P : PCHar = '123456789ABCDEF';

var PP : PCHar;

begin
  PP:=StrAlloc(11);
  WriteIn ('First 10 characters of P : ',StrLCopy (PP,P,10));
  StrDispose(PP);
end.
```

73.2.14 strlen

Synopsis: Length of a null-terminated string.

Declaration: `function strlen(p: pchar) : sizeint`

Visibility: default

Description: Returns the length of the null-terminated string `P`. If `P` equals `Nil`, then zero (0) is returned.

Errors: None.

See also: StrNew (617)

Listing: ./stringex/ex1.pp

```
Program Example1;

Uses strings;

{ Program to demonstrate the StrLen function. }

Const P : PChar = 'This is a constant pchar string';

begin
  WriteIn ('P         : ',p);
  WriteIn ('length(P) : ',StrLen(P));
end.
```

73.2.15 strlicomp

Synopsis: Compare limited number of characters in 2 null-terminated strings, ignoring case.

Declaration: `function strlicomp(str1: pchar;str2: pchar;l: SizeInt) : SizeInt`

Visibility: default

Description: Compares maximum `L` characters of the null-terminated strings `S1` and `S2`, ignoring case. The result is

- A negative `SizeInt` when `S1<S2`.
- 0 when `S1=S2`.
- A positive `SizeInt` when `S1>S2`.

For an example, see StrIComp (612)

Errors: None.

See also: StrLComp (614), StrComp (609), StrIComp (612)

73.2.16 strlower

Synopsis: Convert null-terminated string to all-lowercase.

Declaration: `function strlower(p: pchar) : pchar`

Visibility: default

Description: Converts `P` to an all-lowercase string. Returns `P`.

Errors: None.

See also: StrUpper (620)

Listing: ./stringex/ex14.pp

```
Program Example14;

Uses strings;

{ Program to demonstrate the StrLower and StrUpper functions. }

Const
    P1 : PChar = 'THIS IS AN UPPERCASE PCHAR STRING';
    P2 : PChar = 'this is a lowercase string';

begin
  WriteLn ('Uppercase : ',StrUpper(P2));
  StrLower (P1);
  WriteLn ('Lowercase : ',P1);
end.
```

73.2.17 strmove

Synopsis: Move a null-terminated string to new location.

Declaration: `function strmove(dest: pchar;source: pchar;l: SizeInt) : pchar`

Visibility: default

Description: Copies `MaxLen` characters from `Source` to `Dest`. No terminating null-character is copied. Returns `Dest`

Errors: None.

See also: StrLCopy (614), StrCopy (609)

Listing: ./stringex/ex10.pp

```
Program Example10;

Uses strings;

{ Program to demonstrate the StrMove function. }

Const P1 : PCHAR = 'This is a pchar string.';


Var P2 : Pchar;

begin
  P2:=StrAlloc(StrLen(P1)+1);
  StrMove (P2,P1,StrLen(P1)+1); { P2:=P1 }
  Writeln ('P2 = ',P2);
  StrDispose(P2);
end.
```

73.2.18 strnew

Synopsis: Allocate room for new null-terminated string.

Declaration: `function strnew(p: pchar) : pchar`

Visibility: default

Description: Copies `P` to the Heap, and returns a pointer to the copy.

Errors: Returns `Nil` if no memory was available for the copy.

See also: StrCopy (609), StrDispose (610)

Listing: ./stringex/ex16.pp

```
Program Example16;

Uses strings;

{ Program to demonstrate the StrNew function. }

Const P1 : PChar = 'This is a PChar string';

var P2 : PChar;

begin
  P2:=StrNew (P1);
  If P1=P2 then
    writeln ('This can''t be happening...')
  else
    writeln ('P2 : ',P2);
  StrDispose(P2);
end.
```

73.2.19 strpas

Synopsis: Convert a null-terminated string to a shortstring.

Declaration: `function strpas(p: pchar) : shortstring`

Visibility: default

Description: Converts a null terminated string in `P` to a Pascal string, and returns this string. The string is truncated at 255 characters.

Errors: None.

See also: StrPCopy (618)

Listing: ./stringex/ex3.pp

```
Program Example3;

Uses strings;

{ Program to demonstrate the StrPas function. }

Const P : PChar = 'This is a PCHAR string';

var S : string;

begin
  S:=StrPas (P);
  Writeln ('S : ',S);
end.
```

73.2.20 strpcopy

Synopsis: Copy a pascal string to a null-terminated string

Declaration: `function strpcopy(d: pchar;const s: string) : pchar`

Visibility: default

Description: Converts the Pascal string in `Se` to a Null-terminated string, and copies it to `D`. `D` needs enough room to contain the string `Source`, i.e. `Length(S)+1` bytes.

Errors: No length checking is performed.

See also: StrPas (618)

Listing: ./stringex/ex2.pp

```
Program Example2;

Uses strings;

{ Program to demonstrate the StrPCopy function. }

Const S = 'This is a normal string.';

Var P : Pchar;
```

```
begin
  p:=StrAlloc (length(S)+1);
  if StrPCopy (P,S)<>P then
    WriteIn ('This is impossible !!')
  else
    writeln (P);
  StrDispose(P);
end.
```

73.2.21 strpos

Synopsis: Search for a null-terminated substring in a null-terminated string

Declaration: `function strpos(str1: pchar;str2: pchar) : pchar`

Visibility: default

Description: Returns a pointer to the first occurrence of `S2` in `S1`. If `S2` does not occur in `S1`, returns `Nil`.

Errors: None.

See also: StrScan (620), StrRScan (620)

Listing: ./stringex/ex15.pp

```
Program Example15;

Uses strings;

{ Program to demonstrate the StrPos function. }

Const P : PChar = 'This is a PChar string.';
      S : Pchar = 'is';
begin
  WriteIn ('Position of ''is'' in P : ',sizeint(StrPos(P,S))-sizeint(P));
end.
```

73.2.22 strriscan

Synopsis: Scan a string reversely for a character, case-insensitive

Declaration: `function strriscan(p: pchar;c: Char) : pchar`

Visibility: default

Description: `strriscan` does the same as strrscan (620) but compares the characters case-insensitively. It returns a pointer to the last occurrence of the character `c` in the null-terminated string `p`, or `Nil` if `c` is not present in the string.

See also: strrscan (620), striscan (613)

73.2.23 strrscan

Synopsis: Find last occurrence of a character in a null-terminated string.

Declaration: `function strrscan(p: pchar;c: Char) : pchar`

Visibility: default

Description: Returns a pointer to the last occurrence of the character `C` in the null-terminated string `P`. If `C` does not occur, returns `Nil`.

For an example, see StrScan (620).

Errors: None.

See also: StrScan (620), StrPos (619)

73.2.24 strscan

Synopsis: Find first occurrence of a character in a null-terminated string.

Declaration: `function strscan(p: pchar;c: Char) : pchar`

Visibility: default

Description: Returns a pointer to the first occurrence of the character `C` in the null-terminated string `P`. If `C` does not occur, returns `Nil`.

Errors: None.

See also: StrRScan (620), StrPos (619)

Listing: ./stringex/ex13.pp

```
Program Example13;

Uses strings;

{ Program to demonstrate the StrScan and StrRScan functions. }

Const P : PChar = 'This is a PCHAR string.';
      S : Char = 's' ;

begin
  Writeln ('P, starting from first ''s'' : ',StrScan(P,s));
  Writeln ('P, starting from last ''s'' : ',StrRScan(P,s));
end.
```

73.2.25 strupper

Synopsis: Convert null-terminated string to all-uppercase

Declaration: `function strupper(p: pchar) : pchar`

Visibility: default

Description: Converts `P` to an all-uppercase string. Returns `P`.

For an example, see StrLower (616)

Errors: None.

See also: StrLower (616)

Chapter 74

Reference for unit 'strutils'

74.1 Used units

Table 74.1: Used units by unit 'strutils'

Name	Page
System	622

Chapter 75

Reference for unit 'System'

75.1 Overview

The system unit contains the standard supported functions of Free Pascal. It is the same for all platforms. Basically it is the same as the system unit provided with Borland or Turbo Pascal.

Functions are listed in alphabetical order. Arguments of functions or procedures that are optional are put between square brackets.

The pre-defined constants and variables are listed in the first section. The second section contains an overview of all functions, grouped by functionality, and the last section contains the supported functions and procedures.

75.2 Unicode and codepage support

The system unit works with Short-, Ansi-, and UnicodeString routines for all string related operations.

Ansistrings are code-page aware, which means that code page information is associated with them. For most routines, the support for converting these code pages is natural. For some routines, care must be taken when converting from codepage-aware strings to widestring.

The codepage conversion support is influenced by the following variables:

Table 75.1:

Name	Description
DefaultSystemCodePage (622)	Actual code page to use when CP_ACP (622) is encountered
DefaultUnicodeCodePage (622)	Code page for new unicode strings
DefaultFileSystemCodePage (622)	Codepage to use when sending strings to single-byte OS filesystem routines.
DefaultRTLFileSystemCodePage (622)	Codepage to use when receiving strings from single-byte OS filesystem routines.

The windows code page identifiers are used. There are 3 special codepage identifiers:

Table 75.2:

Name	Description
CP_ACP (622)	Currently set default syststem cpdepage
CP_OEMCP (622)	OEM (console) code page (only on windows)
CP_NONE (622)	Indicates absence of code page information for a string
DefaultRTLFileSystemCodePage (622)	

The following routines may perform code page conversions:

Table 75.3:

Name	Description
LowerCase (622)	Return lowercase version of a string.
UpCase (622)	Convert a string to all uppercase.
GetDir (622)	Return the current directory
MkDir (622)	Create a new directory.
ChDir (622)	Change current working directory.
RmDir (622)	Remove directory when empty.
Assign (622)	Assign a name to a file
Erase (622)	Delete a file from disk
Rename (622)	Rename file on disk
Read (622)	Read from a text file into variable
ReadLn (622)	Read from a text file into variable and goto next line
Write (622)	Write variable to a text file
WriteLn (622)	Write variable to a text file and append newline
ReadStr (622)	Read variables from a string
WriteStr (622)	Write variables to a string
Insert (622)	Insert one string in another.
Copy (622)	Copy part of a string.
Delete (622)	Delete part of a string.
SetString (622)	Set length of a string and copy buffer.

All these routines exist also in Unicode versions.

Note that for conversion of codepages and unicode strings, a unicode manager must be present. On windows, the system is used for this. On unix, one of the fpwidestring or cwstring units must be used.

75.3 Miscellaneous functions

Functions that do not belong in one of the other categories.

Table 75.4:

Name	Description
Assert (622)	Conditionally abort program with error
Break (622)	Abort current loop
Continue (622)	Next cycle in current loop
Exclude (622)	Exclude an element from a set
Exit (622)	Exit current function or procedure
Include (622)	Include an element into a set
LongJmp (622)	Jump to execution point
Ord (622)	Return ordinal value of enumerated type
Pred (622)	Return previous value of ordinal type
SetJmp (622)	Mark execution point for jump
SizeOf (622)	Return size of variable or type
Succ (622)	Return next value of ordinal type

75.4 Operating System functions

Functions that are connected to the operating system.

Table 75.5:

Name	Description
Chdir (622)	Change working directory
Getdir (622)	Return current working directory
Halt (622)	Halt program execution
Paramcount (622)	Number of parameters with which program was called
Paramstr (622)	Retrieve parameters with which program was called
Mkdir (622)	Make a directory
Rmdir (622)	Remove a directory
Runerror (622)	Abort program execution with error condition

75.5 String handling

All things connected to string handling.

Table 75.6:

Name	Description
BinStr (622)	Construct binary representation of integer
Chr (622)	Convert ASCII code to character
Concat (622)	Concatenate two strings
Copy (622)	Copy part of a string
Delete (622)	Delete part of a string
HexStr (622)	Construct hexadecimal representation of integer
Insert (622)	Insert one string in another
Length (622)	Return length of string
Lowercase (622)	Convert string to all-lowercase
OctStr (622)	Construct octal representation of integer
Pos (622)	Calculate position of one string in another
SetLength (622)	Set length of a string
SetString (622)	Set contents and length of a string
Str (622)	Convert number to string representation
StringOfChar (622)	Create string consisting of a number of characters
Upcase (622)	Convert string to all-uppercase
Val (622)	Convert string to number

75.6 Mathematical routines

Functions connected to calculating and coverting numbers.

Table 75.7:

Name	Description
Abs (622)	Calculate absolute value
Arctan (622)	Calculate inverse tangent
Cos (622)	Calculate cosine of angle
Dec (622)	Decrease value of variable
Exp (622)	Exponentiate
Frac (622)	Return fractional part of floating point value
Hi (622)	Return high byte/word of value
Inc (622)	Increase value of variable
Int (622)	Calculate integer part of floating point value
Ln (622)	Calculate logarithm
Lo (622)	Return low byte/word of value
Odd (622)	Is a value odd or even ?
Pi (622)	Return the value of pi
Random (622)	Generate random number
Randomize (622)	Initialize random number generator
Round (622)	Round floating point value to nearest integer number
Sin (622)	Calculate sine of angle
Sqr (622)	Calculate the square of a value
Sqrt (622)	Calculate the square root of a value
Swap (622)	Swap high and low bytes/words of a variable
Trunc (622)	Truncate a floating point value

75.7 Memory management functions

Functions concerning memory issues.

Table 75.8:

Name	Description
Addr (622)	Return address of variable
Assigned (622)	Check if a pointer is valid
CompareByte (622)	Compare 2 memory buffers byte per byte
CompareChar (622)	Compare 2 memory buffers byte per byte
CompareDWord (622)	Compare 2 memory buffers byte per byte
CompareWord (622)	Compare 2 memory buffers byte per byte
CSeg (622)	Return code segment
Dispose (622)	Free dynamically allocated memory
DSeg (622)	Return data segment
FillByte (622)	Fill memory region with 8-bit pattern
FillChar (622)	Fill memory region with certain character
FillDWord (622)	Fill memory region with 32-bit pattern
FillQWord (622)	Fill memory region with 64-bit pattern
FillWord (622)	Fill memory region with 16-bit pattern
Freemem (622)	Release allocated memory
Getmem (622)	Allocate new memory
GetMemoryManager (622)	Return current memory manager
High (622)	Return highest index of open array or enumerated
IndexByte (622)	Find byte-sized value in a memory range
IndexChar (622)	Find char-sized value in a memory range
IndexDWord (622)	Find DWord-sized (32-bit) value in a memory range
IndexQWord (622)	Find QWord-sized value in a memory range
IndexWord (622)	Find word-sized value in a memory range
IsMemoryManagerSet (622)	Is the memory manager set
Low (622)	Return lowest index of open array or enumerated
Move (622)	Move data from one location in memory to another
MoveChar0 (622)	Move data till first zero character
New (622)	Dynamically allocate memory for variable
Ofs (622)	Return offset of variable
Ptr (622)	Combine segment and offset to pointer
ReAllocMem (622)	Resize a memory block on the heap
Seg (622)	Return segment
SetMemoryManager (622)	Set a memory manager
Sptr (622)	Return current stack pointer
SSeg (622)	Return stack segment register value

75.8 File handling functions

Functions concerning input and output from and to file.

Table 75.9:

Name	Description
Append (622)	Open a file in append mode
Assign (622)	Assign a name to a file
Blockread (622)	Read data from a file into memory
Blockwrite (622)	Write data from memory to a file
Close (622)	Close a file
Eof (622)	Check for end of file
Eoln (622)	Check for end of line
Erase (622)	Delete file from disk
Filepos (622)	Position in file
Filesize (622)	Size of file
Flush (622)	Write file buffers to disk
IOresult (622)	Return result of last file IO operation
Read (622)	Read from file into variable
Readln (622)	Read from file into variable and goto next line
Rename (622)	Rename file on disk
Reset (622)	Open file for reading
Rewrite (622)	Open file for writing
Seek (622)	Set file position
SeekEof (622)	Set file position to end of file
SeekEoln (622)	Set file position to end of line
SetTextBuf (622)	Set size of file buffer
Truncate (622)	Truncate the file at position
Write (622)	Write variable to file
WriteLn (622)	Write variable to file and append newline

75.9 Run-Time Error behaviour

The sytem unit handles errors by default by generating a run-time error, and halting the program with an exit code equal to the run-time error number.

This behaviour changes when the SysUtils (628) unit is used. In that case, all run-time errors are converted to exceptions: most run-time errors have their own exception class.

If these exceptions are caught, the program code decides what to do with it. If the exception is not caught, the program will exit through the default exception handler.

When the system unit documentation refers to run-time errors, the above should be kept in mind.

Chapter 76

Reference for unit 'sysutils'

76.1 Used units

Table 76.1: Used units by unit 'sysutils'

Name	Page
Linux	343
System	622

76.2 Overview

This documentation describes the sysutils unit. The sysutils unit was started by Gertjan Schouten, and completed by Michael Van Canneyt. It aims to be compatible to the Delphi sysutils unit, but in contrast with the latter, it is designed to work on multiple platforms. It is implemented on all supported platforms.

76.3 Localization support

Localization support depends on various constants and structures being initialized correctly. On Windows and OS/2 this is done automatically: a widestring manager is installed by default which helps taking care of the current locale when performing various operations on strings. The various internationalization settings (date/time format, currency, language etc) are also initialized correctly on these platforms.

On unixes, the widestring support is in a separate unit: cwstring, which loads the various needed functions from the C library. It should be added manually to the uses clause of your program. No internationalization (or localisation) settings are applied by this unit, these must be initialized separately by including the clocale unit in the uses clause of your program.

76.4 Unicode and codepage awareness

The many functions that deal with filenames in the sysutils routines have been changed from `AnsiString` to RawByteString so they do not perform implict codepage conversions to the ansi code page. At the

same time, overloaded versions that accept a unicode string have been created.

For routines that access actual OS functions using single-byte string APIS, the strings are converted to ensure that the OS routine receives a string with the correct encoding when using single-byte strings. This encoding is normally the DefaultFileSystemCodePage (622) encoding.

On systems with a unicode I/O API (2-byte strings), the native API is used, meaning that unicode strings will be passed on as-is, but single-byte strings will be converted (implicitly) to Unicode.

The following is a minimal list of functions that have been changed and duplicated:

Table 76.2:

Name	Description
FileCreate (628)	Create a new file and return a handle to it.
FileOpen (628)	Open an existing file and return a filehandle
FileExists (628)	Check whether a particular file exists in the filesystem.
DirectoryExists (628)	Check whether a directory exists in the file system.
FileSetDate (628)	Set the date of a file.
FileGetAttr (628)	Return attributes of a file.
FileSetAttr (628)	Set the attributes of a file.
DeleteFile (628)	Delete a file from the filesystem.
RenameFile (628)	Rename a file.
FileSearch (628)	Search for a file in a path.
ExeSearch (628)	Search for an executable
FindFirst (628)	Start a file search and return a findhandle
FindNext (628)	Find the next entry in a findhandle.
FindClose (628)	Close a find handle
FileIsReadOnly (628)	Check whether a file is read-only.
GetCurrentDir (628)	Return the current working directory of the application.
SetCurrentDir (628)	Set the current directory of the application.

The following functions do not interact with the OS, but may nevertheless change the codepage of the strings involved in their operation:

Table 76.3:

Name	Description
ChangeFileExt (628)	Change the extension of a filename.
ExtractFilePath (628)	Extract the path from a filename.
ExtractFileDrive (628)	Extract the drive part from a filename.
ExtractFileName (628)	Extract the filename part from a full path filename.
ExtractFileExt (628)	Return the extension from a filename.
ExtractFileDir (628)	Extract the drive and directory part of a filename.
ExtractShortPathName (628)	Returns a 8.3 path name
ExpandFileName (628)	Expand a relative filename to an absolute filename.
ExpandFileNameCase (628)	Expand a filename entered as case insensitive to the full path as stored on the disk.
ExtractRelativepath (628)	Extract a relative path from a filename, given a base directory.
ExpandUNCFileName (628)	Expand a relative filename to an absolute UNC filename.
IncludeTrailingPathDelimiter (628)	Add trailing directory separator to a pathname, if needed.
IncludeTrailingBackslash (628)	Add trailing directory separator to a pathname, if needed.
ExcludeTrailingBackslash (628)	Strip trailing directory separator from a pathname, if needed.
ExcludeTrailingPathDelimiter (628)	Strip trailing directory separator from a pathname, if needed.
IncludeLeadingPathDelimiter (628)	Prepend a path delimiter if there is not already one.
ExcludeLeadingPathDelimiter (628)	Strip the leading path delimiter of a path
IsPathDelimiter (628)	Is the character at the given position a pathdelimiter ?
DoDirSeparators (628)	Convert known directory separators to the current directory separator.
SetDirSeparators (628)	Set the directory separators to the known directory separators.
GetDirs (628)	Return a list of directory names from a path.
ConcatPaths (628)	Concatenate an array of paths to form a single path
GetEnvironmentVariable (628)	Return the value of an environment variable.

76.5 Miscellaneous conversion routines

Functions for various conversions.

Table 76.4:

Name	Description
BCDToInt (628)	Convert BCD number to integer
CompareMem (628)	Compare two memory regions
FloatToStrF (628)	Convert float to formatted string
FloatToStr (628)	Convert float to string
FloatToText (628)	Convert float to string
FormatFloat (628)	Format a floating point value
GetDirs (628)	Split string in list of directories
IntToHex (628)	return hexadecimal representation of integer
IntToStr (628)	return decumal representation of integer
StrToIntDef (628)	Convert string to integer with default value
StrToInt (628)	Convert string to integer
StrToFloat (628)	Convert string to float
TextToFloat (628)	Convert null-terminated string to float

76.6 Date/time routines

Functions for date and time handling.

Table 76.5:

Name	Description
DateTimeToFileDate (628)	Convert DateTime type to file date
DateTimeToStr (628)	Construct string representation of DateTime
DateTimeToString (628)	Construct string representation of DateTime
DateTimeToSystemTime (628)	Convert DateTime to system time
DateTimeToTimeStamp (628)	Convert DateTime to timestamp
DateToStr (628)	Construct string representation of date
Date (628)	Get current date
DayOfWeek (628)	Get day of week
DecodeDate (628)	Decode DateTime to year month and day
DecodeTime (628)	Decode DateTime to hours, minutes and seconds
EncodeDate (628)	Encode year, day and month to DateTime
EncodeTime (628)	Encode hours, minutes and seconds to DateTime
FormatDateTime (628)	Return string representation of DateTime
IncMonth (628)	Add 1 to month
IsLeapYear (628)	Determine if year is leap year
MSecsToTimeStamp (628)	Convert nr of milliseconds to timestamp
Now (628)	Get current date and time
StrToDateTime (628)	Convert string to DateTime
StrToDate (628)	Convert string to date
StrToTime (628)	Convert string to time
SystemTimeToDateTime (628)	Convert system time to datetime
TimeStampToDateTime (628)	Convert time stamp to DateTime
TimeStampToMSecs (628)	Convert Timestamp to number of millicseconds
TimeToStr (628)	return string representation of Time
Time (628)	Get current time

76.7 FileName handling routines

Functions for file manipulation.

Table 76.6:

Name	Description
AnsiCompareFileName (628)	Compare 2 filenames
AnsiLowerCaseFileName (628)	Create lowercase filename
AnsiUpperCaseFileName (628)	Create uppercase filename
AddDisk (628)	Add disk to list of disk drives
ChangeFileExt (628)	Change extension of file name
CreateDir (628)	Create a directory
DeleteFile (628)	Delete a file
DiskFree (628)	Free space on disk
DiskSize (628)	Total size of disk
ExpandFileName (628)	Create full file name
ExpandFileNameCase (628)	Create full file name case insensitively
ExpandUNCFileName (628)	Create full UNC file name
ExtractFileDir (628)	Extract drive and directory part of filename
ExtractFileDrive (628)	Extract drive part of filename
ExtractFileExt (628)	Extract extension part of filename
ExtractFileName (628)	Extract name part of filename
ExtractFilePath (628)	Extrct path part of filename
ExtractRelativePath (628)	Construct relative path between two files
FileAge (628)	Return file age
FileDateToDateTime (628)	Convert file date to system date
FileExists (628)	Determine whether a file exists on disk
FileGetAttr (628)	Get attributes of file
FileGetDate (628)	Get date of last file modification
FileSearch (628)	Search for file in path
FileSetAttr (628)	Get file attributes
FileSetDate (628)	Get file dates
FindFirst (628)	Start finding a file
FindNext (628)	Find next file
GetCurrentDir (628)	Return current working directory
RemoveDir (628)	Remove a directory from disk
RenameFile (628)	Rename a file on disk
SameFileName (628)	Check whether 2 filenames are the same
SetCurrentDir (628)	Set current working directory
SetDirSeparators (628)	Set directory separator characters
FindClose (628)	Stop searching a file
DoDirSeparators (628)	Replace directory separator characters

76.8 File input/output routines

Functions for reading/writing to file.

Table 76.7:

Name	Description
FileCreate (628)	Create a file and return handle
FileOpen (628)	Open file end return handle
FileRead (628)	Read from file
FileSeek (628)	Set file position
FileTruncate (628)	Truncate file length
FileWrite (628)	Write to file
FileClose (628)	Close file handle

76.9 PChar related functions

Most PChar functions are the same as their counterparts in the STRINGS unit. The following functions are the same :

1. StrCat (628) : Concatenates two `PChar` strings.
2. StrComp (628) : Compares two `PChar` strings.
3. StrCopy (628) : Copies a `PChar` string.
4. StrECopy (628) : Copies a `PChar` string and returns a pointer to the terminating null byte.
5. StrEnd (628) : Returns a pointer to the terminating null byte.
6. StrIComp (628) : Case insensitive compare of 2 `PChar` strings.
7. StrLCat (628) : Appends at most L characters from one `PChar` to another `PChar`.
8. StrLComp (628) : Case sensitive compare of at most L characters of 2 `PChar` strings.
9. StrLCopy (628) : Copies at most L characters from one `PChar` to another.
10. StrLen (628) : Returns the length (exclusive terminating null byte) of a `PChar` string.
11. StrLIComp (628) : Case insensitive compare of at most L characters of 2 `PChar` strings.
12. StrLower (628) : Converts a `PChar` to all lowercase letters.
13. StrMove (628) : Moves one `PChar` to another.
14. StrNew (628) : Makes a copy of a `PChar` on the heap, and returns a pointer to this copy.
15. StrPos (628) : Returns the position of one `PChar` string in another?
16. StrRScan (628) : returns a pointer to the last occurrence of on `PChar` string in another one.
17. StrScan (628) : returns a pointer to the first occurrence of on `PChar` string in another one.
18. StrUpper (628) : Converts a `PChar` to all uppercase letters.

The subsequent functions are different from their counterparts in STRINGS, although the same examples can be used.

76.10 Date and time formatting characters

Various date and time formatting routines accept a format string to format the date and or time. The following characters can be used to control the date and time formatting:

c Formats date using `shortdateformat` and formats time using `longtimeformat` if the time is not zero.

f Same as c, but adds the time even if it is zero.

d day of month

dd day of month (leading zero)

ddd day of week (abbreviation)

dddd day of week (full)

ddddd shortdateformat

dddddd longdateformat

m month

mm month (leading zero)

mmm month (abbreviation)

mmmm month (full)

y year (2 digits)

yy year (two digits)

yyyy year (with century)

h hour

hh hour (leading zero)

n minute

nn minute (leading zero)

s second

ss second (leading zero)

t shorttimeformat

tt longtimeformat

am/pm use 12 hour clock and display am and pm accordingly

a/p use 12 hour clock and display a and p accordingly

/ insert date seperator

: insert time seperator

"xx" literal text

'xx' literal text

z milliseconds

zzz milliseconds(leading zero)

The date and time separators are taken from the DefaultFormatSettings (628) record, unless a TFormatSettings (628) record is passed to the FormatDateTime (628) function.

Note that to include any of the above characters literally in the result string, they must be enclosed in double quotes.

See also: DefaultFormatSettings (628), TFormatSettings (628), FormatDateTime (628)

76.11 Formatting strings

Functions for formatting strings.

Table 76.8:

Name	Description
AdjustLineBreaks (628)	Convert line breaks to line breaks for system
FormatBuf (628)	Format a buffer
Format (628)	Format arguments in string
FmtStr (628)	Format buffer
QuotedStr (628)	Quote a string
StrFmt (628)	Format arguments in a string
StrLFmt (628)	Format maximum L characters in a string
TrimLeft (628)	Remove whitespace at the left of a string
TrimRight (628)	Remove whitespace at the right of a string
Trim (628)	Remove whitespace at both ends of a string

76.12 String functions

Functions for handling strings.

Table 76.9:

Name	Description
AnsiCompareStr (628)	Compare two strings
AnsiCompareText (628)	Compare two strings, case insensitive
AnsiExtractQuotedStr (628)	Removes quotes from string
AnsiLastChar (628)	Get last character of string
AnsiLowerCase (628)	Convert string to all-lowercase
AnsiQuotedStr (628)	Qoutes a string
AnsiStrComp (628)	Compare strings case-sensitive
AnsiStrIComp (628)	Compare strings case-insensitive
AnsiStrLComp (628)	Compare L characters of strings case sensitive
AnsiStrLIComp (628)	Compare L characters of strings case insensitive
AnsiStrLastChar (628)	Get last character of string
AnsiStrLower (628)	Convert string to all-lowercase
AnsiStrUpper (628)	Convert string to all-uppercase
AnsiUpperCase (628)	Convert string to all-uppercase
AppendStr (628)	Append 2 strings
AssignStr (628)	Assign value of strings on heap
CompareStr (628)	Compare two strings case sensitive
CompareText (628)	Compare two strings case insensitive
DisposeStr (628)	Remove string from heap
IsValidIdent (628)	Is string a valid pascal identifier
LastDelimiter (628)	Last occurance of character in a string
LeftStr (628)	Get first N characters of a string
LoadStr (628)	Load string from resources
LowerCase (628)	Convert string to all-lowercase
NewStr (628)	Allocate new string on heap
RightStr (628)	Get last N characters of a string
StrAlloc (628)	Allocate memory for string
StrBufSize (628)	Reserve memory for a string
StrDispose (628)	Remove string from heap
StrPas (628)	Convert PChar to pascal string
StrPCopy (628)	Copy pascal string
StrPLCopy (628)	Copy N bytes of pascal string
UpperCase (628)	Convert string to all-uppercase

Chapter 77

Reference for unit 'Types'

77.1 Overview

Starting with D6, types from Windows specific units that were needed in Kylix were extracted to this unit. So it mostly contains type of Windows origin that are needed in the VCL framework.

77.2 Constants, types and variables

77.2.1 Constants

```
E_FAIL = ($80004005)
```

Defined for Delphi compatibility, this should not be used.

```
E_INVALIDARG = ($80070057)
```

Defined for Delphi compatibility, this should not be used.

```
GUID_NULL : TGUID = '{00000000-0000-0000-0000-000000000000}'
```

GUID_NULL is the definition of the NULL (empty) GUID.

```
LOCK_EXCLUSIVE = 2
```

Defined for Delphi compatibility, this should not be used.

```
LOCK_ONLYONCE = 4
```

Defined for Delphi compatibility, this should not be used.

```
LOCK_WRITE = 1
```

Defined for Delphi compatibility, this should not be used.

```
STATFLAG_DEFAULT = 0
```

Defined for Delphi compatibility, this should not be used.

```
STATFLAG_NONAME = 1
```

Defined for Delphi compatibility, this should not be used.

```
STATFLAG_NOOPEN = 2
```

Defined for Delphi compatibility, this should not be used.

```
STGTY_LOCKBYTES = 3
```

Defined for Delphi compatibility, this should not be used.

```
STGTY_PROPERTY = 4
```

Defined for Delphi compatibility, this should not be used.

```
STGTY_STORAGE = 1
```

Defined for Delphi compatibility, this should not be used.

```
STGTY_STREAM = 2
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_ABNORMALAPIEXIT = ($800300FA)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_ACCESSDENIED = ($80030005)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_BADBASEADDRESS = ($80030110)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_CANTSAVE = ($80030103)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_DISKISWRITEPROTECTED = ($80030013)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_DOCFILECORRUPT = ($80030109)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_EXTANTMARSHALLINGS = ($80030108)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_FILEALREADYEXISTS = ($80030050)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_FILENOTFOUND = ($80030002)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_INCOMPLETE = ($80030201)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_INSUFFICIENTMEMORY = ($80030008)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_INUSE = ($80030100)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_INVALIDFLAG = ($800300FF)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_INVALIDFUNCTION = ($80030001)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_INVALIDHANDLE = ($80030006)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_INVALIDHEADER = ($800300FB)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_INVALIDNAME = ($800300FC)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_INVALIDPARAMETER = ($80030057)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_INVALIDPOINTER = ($80030009)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_LOCKVIOLATION = ($80030021)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_MEDIUMFULL = ($80030070)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_NOMOREFILES = ($80030012)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_NOTCURRENT = ($80030101)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_OLDDLL = ($80030105)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_OLDFORMAT = ($80030104)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_PATHNOTFOUND = ($80030003)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_PROPSETMISMATCHED = ($800300F0)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_READFAULT = ($8003001E)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_REVERTED = ($80030102)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_SEEKERROR = ($80030019)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_SHAREREQUIRED = ($80030106)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_SHAREVIOLATION = ($80030020)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_TERMINATED = ($80030202)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_TOOMANYOPENFILES = ($80030004)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_UNIMPLEMENTEDFUNCTION = ($800300FE)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_UNKNOWN = ($800300FD)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_WRITEFAULT = ($8003001D)
```

Defined for Delphi compatibility, this should not be used.

```
STG_S_BLOCK = $00030201
```

Defined for Delphi compatibility, this should not be used.

```
STG_S_CONVERTED = $00030200
```

Defined for Delphi compatibility, this should not be used.

```
STG_S_MONITORING = $00030203
```

Defined for Delphi compatibility, this should not be used.

```
STG_S_RETRYNOW = $00030202
```

Defined for Delphi compatibility, this should not be used.

```
STREAM_SEEK_CUR = 1
```

Defined for Delphi compatibility, this should not be used.

```
STREAM_SEEK_END = 2
```

Defined for Delphi compatibility, this should not be used.

```
STREAM_SEEK_SET = 0
```

Defined for Delphi compatibility, this should not be used.

77.2.2 Types

`ArgList = Pointer`

`ArgList` is defined for Delphi/Kylix compatibility and should not be used.

`DWORD = LongWord`

Alias for cardinal type

`FILETIME = _FILETIME`

Alias for the `_FILETIME` type

`Largeint = Int64`

`Largeint` is an alias for the Int64 type defined in the system unit. This is an alias for Delphi/Kylix compatibility.

`LargeUint = QWord`

`LargeUInt` is an alias for the `QWord` type defined in the system unit. This is an alias for Delphi/Kylix compatibility.

`LARGE_INT = Largeint`

`LARGE_INT` is an alias for the Int64 type defined in the system unit. This is an alias for Delphi/Kylix compatibility.

`LARGE_UINT = LargeUint`

`LARGE_UINT` is an alias for the `QWord` type defined in the system unit. This is an alias for Delphi/Kylix compatibility.

`PByte = System.PByte`

`PByte` is defined in the system unit. This is an alias for Delphi/Kylix compatibility.

`PCLSID = PGUID`

`PCLSID` is a pointer to a `TCLSID` type.

`PDisplay = Pointer`

`PDisplay` is defined for Delphi/Kylix compatibility and should not be used.

`PDouble = System.PDouble`

`PDouble` is defined in the system unit. This is an alias for Delphi/Kylix compatibility.

`PDWord = ^DWORD`

PDWord is equivalent to the PCardinal type.

```
PEvent = Pointer
```

PEvent is defined for Delphi/Kylix compatibility and should not be used.

```
PFileTime = ^TFileTime
```

Pointer to TFileTime type

```
PLargeInt = ^Largeint
```

PLargeInt is an alias for the PInt64 type defined in the system unit. This is an alias for Delphi/Kylix compatibility.

```
PLargeuInt = ^LargeUint
```

PLargeUInt is an alias for the PQWord type defined in the system unit. This is an alias for Delphi/Kylix compatibility.

```
PLongint = System.PLongint
```

PLongint is defined in the system unit. This is an alias for Delphi/Kylix compatibility.

```
POleStr = PWideChar
```

POleStr is a pointer to a (double) null-terminated array of TOleChar characters.

```
PPoint = ^TPoint
```

PPoint is a typed pointer to the TPoint (646) type.

```
PPOleStr = ^POleStr
```

PPOleStr is a typed pointer to a POleStr variable.

```
PRect = ^TRect
```

PRect is a typed pointer to the TRect (647) type.

```
PSize = ^TSize
```

PSize is a typed pointer to the TSize (647) type.

```
PSmallInt = System.PSmallInt
```

PSmallInt is defined in the system unit. This is an alias for Delphi/Kylix compatibility.

```
PSmallPoint = ^TSmallPoint
```

PSmallPoint is a typed pointer to the TSmallPoint (647) record.

```
PStatStg = ^TStatStg
```

Pointer to TStatStg record.

```
PXrmOptionDescRec = ^TXrmOptionDescRec
```

PXrmOptionDescRec is defined for Delphi/Kylix compatibility and should not be used.

```
Region = Pointer
```

Region is defined for Delphi/Kylix compatibility and should not be used.

```
STATSTG = TStatStg
```

Alias for the TStatStg type.

```
tagPOINT = TPoint
```

tagPOINT is a simple alias for TPoint (646)

```
tagSIZE = TSize
```

tagSize is an alias for the TSize (647) type.

```
tagSTATSTG = record
  pwcsName : POleStr;
  dwType : DWORD;
  cbSize : LARGE_UINT;
  mtime : TFileTime;
  ctime : TFileTime;
  atime : TFileTime;
  grfMode : DWORD;
  grfLocksSupported : DWORD;
  clsid : TCLSID;
  grfStateBits : DWORD;
  reserved : DWORD;
end
```

tagSTATSTG is used in the IStream.Stat (655) call. It describes a storage medium (typically a file).

```
TBooleanDynArray = Array of Boolean
```

TBooleanDynArray is a standard definition of a dynamical array of booleans.

```
TByteDynArray = Array of Byte
```

TByteDynArray is a standard definition of a dynamical array of (8-bit, unsigned) bytes.

```
TCardinalDynArray = Array of Cardinal
```

TCardinalDynArray is a standard definition of a dynamical array of (32-bit, unsigned) cardinals.

```
TCLSID = TGUID
```

TCLSID is an alias for the #rtl.system.TGUID (622) type.

```
TDoubleDynArray = Array of Double
```

TSoubleDynArray is a standard definition of a dynamical array of doubles. (regular floating point type)

```
TDuplicates = (dupIgnore,dupAccept,dupError)
```

Table 77.1: Enumeration values for type TDuplicates

Value	Explanation
dupAccept	Accept duplicates, adding them to the list.
dupError	Raise an error when an attempt is made to add a duplicate.
dupIgnore	Ignore the new item, do not add it to the list.

TDuplicates can be used to indicate how a list structure acts on the addition of a duplicate item to the list.

dupIgnore Ignore the new item, do not add it to the list.

dupAccept Accept duplicates, adding them to the list.

dupError Raise an error when an attempt is made to add a duplicate.

```
TFileTime = _FILETIME
```

Alias for the _FILETIME type

```
TInt64DynArray = Array of Int64
```

TInt64DynArray is a standard definition of a dynamical array of (64-bit, signed) int64s.

```
TIntegerDynArray = Array of Integer
```

TIntegerDynArray is a standard definition of a dynamical array of (32-bit, signed) integers.

```
TListCallback = procedure(data: pointer;arg: pointer) of object
```

TListCallback is the prototype for a Foreach operation on a list. It will be called with as Data the pointer in the list, and Arg will contain the extra user data added to the Foreach call. It can be used in methods of objects; for a version that can be used as a global procedure, see TListStaticCallback (646)

```
TListStaticCallback = procedure(data: pointer;arg: pointer)
```

TListStaticCallback is the prototype for a Foreach operation on a list. It will be called with as Data the pointer in the list, and Arg will contain the extra user data added to the Foreach call. It can be used in plain procedures; for a version that can be used as a method, see TListCallback (645)

```
TLongWordDynArray = Array of LongWord
```

TLongWordDynArray is a standard definition of a dynamical array of (32-bit, unsigned) LongWords.

```
TOleChar = WideChar
```

TOleChar is an alias for the WideChar type, defined in the system unit.

```
TPoint = packed record
  X : LongInt;
  Y : LongInt;
end
```

TPoint is a generic definition of a point in a 2-dimensional discrete plane, where X indicates the horizontal position, and Y the vertical position (positions usually measured in pixels), and 0,0 is the origin of the plane.

Usually, the origin is the upper-left corner of the screen, with Y increasing as one moves further down the screen - this is opposite to the mathematical view where Y increases as one moves upwards.

The coordinates are integers, (32-bit, signed) so the coordinate system runs from -MaxInt to MaxInt.

```
TPointerDynArray = Array of Pointer
```

Dynamic array of untyped pointers

```
TQWordDynArray = Array of QWord
```

TQWordDynArray is a standard definition of a dynamical array of (64-bit, unsigned) QWords.

```
TRect = packed record
case Integer of
0: (
  Left : LongInt;
  Top : LongInt;
  Right : LongInt;
  Bottom : LongInt;
);
1: (
  TopLeft : TPoint;
  BottomRight : TPoint;
);
end
```

TRect defines a rectangle in a discrete plane. It is described by the horizontal (left, right) or vertical (top, Bottom) positions (in pixels) of the edges, or, alternatively, by the coordinates of the top left (TopLeft) and bottom right (BottomRight) corners.

```
TShortIntDynArray = Array of ShortInt
```

TShortintDynArray is a standard definition of a dynamical array of (8-bit, signed) shortints.

```
TSingleDynArray = Array of Single
```

TSingleDynArray is a standard definition of a dynamical array of singles. (smallest floating point type)

```
TSize = packed record
  cx : LongInt;
  cy : LongInt;
end
```

TSize is a type to describe the size of a rectangular area, where cx is the width, cy is the height (in pixels) of the rectangle.

```
TSmallIntDynArray = Array of SmallInt
```

TSmallintDynArray is a standard definition of a dynamical array of (16-bit, unsigned) integers.

```
TSmallPoint = packed record
  x : SmallInt;
  y : SmallInt;
end
```

TSmallPoint defines a point in a 2-dimensional plane, just like TPoint (646), but the coordinates have a smaller range: The coordinates are smallints (16-bit, signed) and they run from -MaxSmallInt to maxSmallint.

```
TStatStg = tagSTATSTG
```

TStatStg is a record type describing a storage medium. It is uses in the IStream.Stat (655) function.

```
TStringDynArray = Array of AnsiString
```

TStringDynArray is a standard definition of a dynamical array of Ansistrings.

```
TWideStringDynArray = Array of WideString
```

TWideStringDynArray is a standard definition of a dynamical array of WideStrings.

```
TWordDynArray = Array of Word
```

TWordDynArray is a standard definition of a dynamical array of (16-bit, unsigned) words.

```
TXrmOptionDescRec = record
end
```

TXrmOptionDescRec is defined for Delphi/Kylix compatibility and should not be used.

```
Widget = Pointer
```

Widget is defined for Delphi/Kylix compatibility and should not be used.

```
WidgetClass = Pointer
```

WidgetClass is defined for Delphi/Kylix compatibility and should not be used.

```
XrmOptionDescRec = TXrmOptionDescRec
```

XrmOptionDescRec is defined for Delphi/Kylix compatibility and should not be used.

```
_FILETIME = packed record
  dwLowDateTime : DWORD;
  dwHighDateTime : DWORD;
end
```

_FILETIME describes a file time stamp. It is defined or Delphi/Kylix compatibility and should not be used except when implementing or accessing the IStream interface. The TDateTime type should be used instead.

77.3 Procedures and functions

77.3.1 Bounds

Synopsis: Create a rectangle, given a position and size

Declaration:
```
function Bounds(ALeft: Integer;ATop: Integer;AWidth: Integer;
                AHeight: Integer) : TRect
```

Visibility: default

Description: Bounds returns a TRect structure with the indicated position (Left=ALeft and Top=ATop) and size (Right=ALeft+AWidth and Bottom=ATop+AHeight)

See also: Rect (650), PtInRect (650), IntersectRect (649), UnionRect (651)

77.3.2 CenterPoint

Synopsis: Return the center point of a rectangle

Declaration:
```
function CenterPoint(const Rect: TRect) : TPoint
```

Visibility: default

Description: CenterPoint returns the center point of the rectangle Rect.

See also: PtinRect (650), IntersectRect (649), IsRectEmpty (649), OffsetRect (650), InflateRect (649), Size (651), IsRectEmpty (649)

77.3.3 EqualRect

Synopsis: Check if two rectangles are equal.

Declaration:
```
function EqualRect(const r1: TRect;const r2: TRect) : Boolean
```

Visibility: default

Description: `EqualRect` returns `True` if the rectangles `R1` and `R2` are equal (i.e. have the position and size). If the rectangles differ, the function returns `False`

See also: Rect (650), Bounds (648), PtInRect (650), IntersectRect (649), UnionRect (651), IsRectEmpty (649), OffsetRect (650), InflateRect (649), Size (651)

77.3.4 InflateRect

Synopsis: Increase the rectangle in size, keeping it centered

Declaration:
```
function InflateRect(var Rect: TRect;dx: Integer;dy: Integer) : Boolean
```

Visibility: default

Description: `InflateRect` inflates the rectangle horizontally with `dx` pixels on each side, and vertically with `dy` pixels, thus keeping its center point on the same location. It returns `true` if the operation was succesfully, `False` if it was not (only possible if the address of `Rect` is `Nil`).

See also: PtinRect (650), IntersectRect (649), IsRectEmpty (649), OffsetRect (650), CenterPoint (648), Size (651), IsRectEmpty (649)

77.3.5 IntersectRect

Synopsis: Return the intersection of 2 rectangles

Declaration:
```
function IntersectRect(var Rect: TRect;const R1: TRect;const R2: TRect)
                       : Boolean
```

Visibility: default

Description: `IntersectRect` returns the intersection of the 2 rectangles `R1` and `R2` in `Rect`. It returns `True` if the 2 rectangles have an intersection, otherwise `False` is returned, and `Rect` is filled with zero.

See also: PtinRect (650), UnionRect (651), IsRectEmpty (649), OffsetRect (650), InflateRect (649), Size (651)

77.3.6 IsRectEmpty

Synopsis: Check whether a rectangle is empty

Declaration:
```
function IsRectEmpty(const Rect: TRect) : Boolean
```

Visibility: default

Description: `IsRectEmpty` returns true if the rectangle is empty, i.e. has a zero or negative width or height.

See also: PtinRect (650), IntersectRect (649), IsRectEmpty (649), OffsetRect (650), InflateRect (649), Size (651)

77.3.7 OffsetRect

Synopsis: Offset the rectangle

Declaration:
```
function OffsetRect(var Rect: TRect;DX: Integer;DY: Integer) : Boolean
```

Visibility: default

Description: `OffsetRect` offsets the rectangle `Rect` by a horizontal distance `DX` and a vertical distance `DY`. The operation returns `True` if the operation was successfull, `false` if it was not (only possible if the address of `Rect` is `Nil`).

See also: PtinRect (650), IntersectRect (649), IsRectEmpty (649), OffsetRect (650), InflateRect (649), Size (651), IsRectEmpty (649)

77.3.8 Point

Synopsis: Create a point

Declaration:
```
function Point(x: Integer;y: Integer) : TPoint
```

Visibility: default

Description: `Point` returns a `TPoint` structure with the given position (`X, Y`).

See also: Rect (650), PtInRect (650)

77.3.9 PtInRect

Synopsis: Check whether a point is inside a rectangle.

Declaration:
```
function PtInRect(const Rect: TRect;const p: TPoint) : Boolean
```

Visibility: default

Description: `PtInRect` returns `True` if `p` is located inside `Rect`, and `False` if it is located outside the rectangle.

Remark: Note that the bottom, right edges are not considered part of the rectangle, therefor a point located on one of these edges will not be considered part of the rectangle, meaning that for a record (10,10,100,100) the point (90,100) will not be considered part of the record, but `90,10` will be.

See also: IntersectRect (649), UnionRect (651), IsRectEmpty (649), OffsetRect (650), InflateRect (649), Size (651)

77.3.10 Rect

Synopsis: Create a rectangle record

Declaration:
```
function Rect(Left: Integer;Top: Integer;Right: Integer;Bottom: Integer)
                  : TRect
```

Visibility: default

Description: `Rect` returns a rectangle structure with the 4 members `Left`, `Top`, `Right` and `Bottom` as passed in the arguments.

See also: Bounds (648), PtInRect (650), IntersectRect (649), UnionRect (651), IsRectEmpty (649), OffsetRect (650), InflateRect (649), Size (651)

77.3.11 Size

Synopsis: Return the size of the rectangle

Declaration:
```
function Size(AWidth: Integer;AHeight: Integer) : TSize
function Size(const ARect: TRect) : TSize
```

Visibility: default

Description: `Size` returns a `TSize` record with the indicated `AWidth, AHeight`. In the case `ARect` is passed, the width and height are calculated (taking into account that the right, bottom are not considered part of the rectangle).

See also: PtinRect (650), IntersectRect (649), IsRectEmpty (649), OffsetRect (650), InflateRect (649), CenterPoint (648), IsRectEmpty (649)

77.3.12 UnionRect

Synopsis: Return the union of 2 rectangles.

Declaration:
```
function UnionRect(var Rect: TRect;const R1: TRect;const R2: TRect)
                        : Boolean
```

Visibility: default

Description: `UnionRect` retuns the rectangle that encompasses both `R1` and `R2` in `Rect`. It returns `True` if the resulting rectangle is not empty, `False` if the result is an empty rectangle (in which case the result is filled with zeroes)

See also: PtinRect (650), IntersectRect (649), IsRectEmpty (649), OffsetRect (650), InflateRect (649), Size (651)

77.4 IClassFactory

77.4.1 Description

`IClassFactory` is defined for Delphi/Kylix compatibility and should not be used.

77.4.2 Method overview

Page	Method	Description
651	CreateInstance	Create a new instance of an interface.
652	LockServer	Lock ActiveX server object.

77.4.3 IClassFactory.CreateInstance

Synopsis: Create a new instance of an interface.

Declaration:
```
function CreateInstance(const unkOuter: IUnknown;const riid: TGUID;
                              out vObject) : HResult
```

Visibility: default

Description: `IClassFactory.CreateInstance` is defined for Delphi/Kylix compatibility and should not be used.

77.4.4 IClassFactory.LockServer

Synopsis: Lock ActiveX server object.

Declaration: `function LockServer(fLock: LongBool) : HResult`

Visibility: default

Description: `IClassFactory.LocksServer` is defined for Delphi/Kylix compatibility and should not be used.

77.5 ISequentialStream

77.5.1 Description

`ISequentialStream` is the interface for streams which only support sequential reading of chunks of data. It is defined for Delphi/Kylix compatibility and should not be used.

See also: IStream (652)

77.5.2 Method overview

Page	Method	Description
652	Read	Read data from the stream
652	Write	Write data to the stream

77.5.3 ISequentialStream.Read

Synopsis: Read data from the stream

Declaration: `function Read(pv: Pointer;cb: DWORD;pcbRead: PDWord) : HRESULT`

Visibility: default

Description: `Read` reads `cbCount` bytes from the stream into the memory pointed to by `pv` and returns the number of bytes read in `pcbread`. The result is zero for success or an error code.

See also: ISequentialStream.Write (652)

77.5.4 ISequentialStream.Write

Synopsis: Write data to the stream

Declaration: `function Write(pv: Pointer;cb: DWORD;pcbWritten: PDWord) : HRESULT`

Visibility: default

Description: `Write` writes `cbCount` bytes from the memory pointed to by `pv` to the stream and returns the number of bytes written in `pcbwritten`. The result is zero for success or an error code.

See also: ISequentialStream.Read (652)

77.6 IStream

77.6.1 Description

An abstract interface for an external (non pascal) stream, as defined in Microsoft COM interfaces

77.6.2 Method overview

Page	Method	Description
655	Clone	Clone the stream instance
654	Commit	Commit data to the stream
653	CopyTo	Copy data from one stream to another
654	LockRegion	Lock a region of bytes in the stream
654	Revert	Revert changes
653	Seek	Set the stream position
653	SetSize	Set the stream size
655	Stat	return information about the stream.
654	UnlockRegion	Unlocks a previously locked region of bytes in the stream

77.6.3 IStream.Seek

Synopsis: Set the stream position

Declaration:
```
function Seek(dlibMove: Largeint;dwOrigin: LongInt;
                  out libNewPosition: Largeint) : HResult
```

Visibility: default

Description: `Seek` sets the stream position at `dlibMove` bytes from `dwOrigin` (one of the `SEEK_*` constants) and returns the new absolute position in `libNewPosition`. The function returns zero on success, or an error code.

Errors: On error, a nonzero exit code is returned.

77.6.4 IStream.SetSize

Synopsis: Set the stream size

Declaration:
```
function SetSize(libNewSize: Largeint) : HRESULT
```

Visibility: default

Description: `SetSize` sets the size of the stream to `libNewSize` bytes, if the stream allows it. On sucess, zero is returned.

Errors: On error, a nonzero exit code is returned.

77.6.5 IStream.CopyTo

Synopsis: Copy data from one stream to another

Declaration:
```
function CopyTo(stm: IStream;cb: Largeint;out cbRead: Largeint;
                    out cbWritten: Largeint) : HRESULT
```

Visibility: default

Description: `CopyTo` copies `cb` bytes from the stream to target stream `stm`. `cbRead` returns how many bytes were read from the stream, `cbwrite` returns how many bytes were actually written to the destination stream. The function returns zero on success.

Errors: On error, a nonzero exit code is returned.

77.6.6 IStream.Commit

Synopsis: Commit data to the stream

Declaration:
```
function Commit(grfCommitFlags: LongInt) : HRESULT
```

Visibility: default

Description: `Commit` commits the data in the stream to the underlying medium. Flags is a set of options to control the commit operation (see MSDN for the possible flags).

Errors: On error, a nonzero exit code is returned.

77.6.7 IStream.Revert

Synopsis: Revert changes

Declaration:
```
function Revert : HRESULT
```

Visibility: default

Description: `Revert` reverts all changes that were done to a transacted stream, i.e. all changes since the last commit. The function returns zero on success.

Errors: On error, a nonzero exit code is returned.

77.6.8 IStream.LockRegion

Synopsis: Lock a region of bytes in the stream

Declaration:
```
function LockRegion(libOffset: Largeint;cb: Largeint;
                    dwLockType: LongInt) : HRESULT
```

Visibility: default

Description: `LockRegion` locks a region of the storage, starting at `libOffset`, for `cbCount` bytes. The applied lock is of type `dwLockType`. The function returns zero if the lock was succesfully applied.

Errors: On error, a nonzero exit code is returned.

77.6.9 IStream.UnlockRegion

Synopsis: Unlocks a previously locked region of bytes in the stream

Declaration:
```
function UnlockRegion(libOffset: Largeint;cb: Largeint;
                      dwLockType: LongInt) : HRESULT
```

Visibility: default

Description: `UnlockRegion` removes the lock on a region of the storage, starting at `libOffset`, for `cbCount` bytes. The lock must be of type `dwLockType`. The function returns zero if the lock was succesfully removed.

Errors: On error, a nonzero exit code is returned.

77.6.10 IStream.Stat

Synopsis: return information about the stream.

Declaration: `function Stat(out statstg: TStatStg;grfStatFlag: LongInt) : HRESULT`

Visibility: default

Description: `Stat` returns information about the stream in `statstg`, taking into account the flags in `grfStatFlag` (one of the `STATFLAG_` constants). The function returns zero if the call was successful.

Errors: On error, a nonzero exit code is returned.

77.6.11 IStream.Clone

Synopsis: Clone the stream instance

Declaration: `function Clone(out stm: IStream) : HRESULT`

Visibility: default

Description: `Clone` returns an independent but initially equal copy of the stream in `stm`. The function returns zero if the call was successful.

Errors: On error, a nonzero exit code is returned.

Chapter 78

Reference for unit 'typinfo'

78.1 Used units

Table 78.1: Used units by unit 'typinfo'

Name	Page
System	622
sysutils	628

78.2 Overview

The TypeInfo unit contains many routines which can be used for the querying of the Run-Time Type Information (RTTI) which is generated by the compiler for classes that are compiled under the `{$M+}` switch. This information can be used to retrieve or set property values for published properties for totally unknown classes. In particular, it can be used to stream classes. The `TPersistent` class in the Classes unit is compiled in the `{$M+}` state and serves as the base class for all classes that need to be streamed.

The unit should be compatible to the Delphi 5 unit with the same name. The only calls that are still missing are the Variant calls, since Free Pascal does not support the variant type yet.

The examples in this chapter use a rttiobj auxiliary unit, which contains an object that has a published property for all supported types. It also contains some auxiliary routines and definitions. This unit is included in the documentation sources, in the directory typinfex.

78.3 Auxiliary functions

Other typinfo related functions.

Table 78.2:

Name	Description
GetEnumName (674)	Get an enumerated type element name
GetEnumValue (676)	Get ordinal number of an enumerated type, based on the name.
GetEnumNameCount (675)	Get number of elements in an enumerated type.
GetTypeData (688)	Skip type name and return a pointer to the type data
SetToString (697)	Convert a set to its string representation
StringToSet (699)	Convert a string representation of a set to a set

78.4 Getting or setting property values

Functions to set or set a property's value.

Table 78.3:

Name	Description
GetEnumProp (675)	Return the value of an enumerated type property
GetFloatProp (676)	Return the value of a float property
GetInt64Prop (677)	Return the value of an Int64 property
GetMethodProp (678)	Return the value of a procedural type property
GetObjectProp (680)	Return the value of an object property
GetOrdProp (682)	Return the value of an ordinal type property
GetPropValue (685)	Return the value of a property as a variant
GetSetProp (686)	Return the value of a set property
GetStrProp (687)	Return the value of a string property
GetWideStrProp (689)	Return the value of a widestring property
GetVariantProp (689)	Return the value of a variant property
SetEnumProp (693)	Set the value of an enumerated type property
SetFloatProp (693)	Set the value of a float property
SetInt64Prop (694)	Set the value of an Int64 property
SetMethodProp (695)	Set the value of a procedural type property
SetObjectProp (695)	Set the value of an object property
SetOrdProp (695)	Set the value of an ordinal type property
SetPropValue (696)	Set the value of a property trhough a variant
SetSetProp (697)	Set the value of a set property
SetStrProp (697)	Set the value of a string property
SetWideStrProp (699)	Set the value of a widestring property
SetVariantProp (699)	Set the value of a variant property

78.5 Examining published property information

Functions for retrieving or examining property information

Table 78.4:

Name	Description
FindPropInfo (673)	Getting property type information, With error checking.
GetPropInfo (683)	Getting property type information, No error checking.
GetPropInfos (683)	Find property information of a certain kind
GetObjectPropClass (681)	Return the declared class of an object property
GetPropList (684)	Get a list of all published properties
IsPublishedProp (690)	Is a property published
IsStoredProp (690)	Is a property stored
PropIsType (691)	Is a property of a certain kind
PropType (692)	Return the type of a property

78.6 Constants, types and variables

78.6.1 Constants

```
BooleanIdents : Array[Boolean] of string = ('False', 'True')
```

Names for boolean values

```
DotSep : string = '.'
```

Name separator character

```
OnGetPropValue : TGetPropValue = Nil
```

This callback is set by the variants unit to enable reading of properties as a variant. If set, it is called by the GetPropValue (685) function.

```
OnGetVariantprop : TGetVariantProp = Nil
```

This callback is set by the variants unit to enable reading of variant properties If set, it is called by the GetVariantProp (689) function.

```
OnSetPropValue : TSetPropValue = Nil
```

This callback is set by the variants unit to enable writing of properties as a variant. If set, it is called by the SetPropValue (696) function.

```
OnSetVariantprop : TSetVariantProp = Nil
```

This callback is set by the variants unit to enable writing of variant properties. If set, it is called by the GetVariantProp (689) function.

```
ptConst = 3
```

Constant used in acces method

```
ptField = 0
```

Property acces directly from field

```
ptStatic = 1
```

Property acces via static method

```
ptVirtual = 2
```

Property acces via virtual method

```
tkAny = [(TTypeKind)..(TTypeKind)]
```

Any property type

```
tkMethods = [tkMethod]
```

Only method properties. (event handlers)

```
tkProcedure = tkProcVar
```

Procedure kind

```
tkProperties = tkAny - tkMethods - [tkUnknown]
```

Real properties. (not methods)

```
tkString = tkSString
```

Alias for the `tsSString` enumeration value

78.6.2 Types

```
PManagedField = ^TManagedField
```

`PManagedField` is a pointer to TManagedField (662). It is used to describe automatically managed fields in records when the type kind is `tkRecord`.

```
PProcedureParam = ^TProcedureParam
```

`PProcedureParam` is a pointer to `TProcedureParam`. It is used in TProcedureSignature (700).

```
PPropInfo = ^TPropInfo
```

Pointer to TPropInfo (664) record

```
PPropList = ^TPropList
```

Pointer to TPropList (664)

```
PPTypeInfo = ^PTypeInfo
```

Pointer to PTypeInfo (660) pointer

```
PTypeData = ^TTypeData
```

Pointer to TTypeData (671) record.

```
PTypeInfo = ^TTypeInfo
```

Pointer to TTypeInfo (671) record

```
PVmtFieldEntry = ^TVmtFieldEntry
```

Pointer to #rtl.typinfo.TVmtFieldEntry (672) type.

```
PVmtFieldTable = ^TVmtFieldTable
```

Pointer to #rtl.typinfo.TVmtFieldTable (673) type.

```
ShortStringBase = string
```

`ShortStringBase` is the base definition of a short string.

```
TArrayTypeData = packed record
  Size : SizeInt;
  ElCount : SizeInt;
  ElType : PTypeInfo;
  DimCount : Byte;
  Dims : Array[0..255] of PTypeInfo;
end
```

`TArrayTypeData` is used to describe arrays in RTTI. It can be encountered when the type kind is `tkArray`, and is used for both static and dynamic arrays and single or multi-dimensional arrays. The type of the array elements is described in `elType`, and the ranges for each of the dimensions (specified in `DimCount` in `Dims`.

```
TCallConv = (ccReg,ccCdecl,ccPascal,ccStdCall,ccSafeCall,ccCppdecl,
             ccFar16,ccOldFPCCall,ccInternProc,ccSysCall,ccSoftFloat,
             ccMWPascal)
```

Table 78.5: Enumeration values for type TCallConv

Value	Explanation
ccCdecl	Cdecl calling convention.
ccCppdecl	Cppdecl calling convention
ccFar16	Far16 calling convention (Delphi compatibility)
ccInternProc	InternProc calling convention (compiler internal)
ccMWPascal	MWPascal (MetroWerks Pascal) calling convention.
ccOldFPCCall	OldFPCCall calling convention (deprecated)
ccPascal	Pascal calling convention.
ccReg	Register calling convention
ccSafeCall	SafeCall calling convention.
ccSoftFloat	Softfloat calling convention.
ccStdCall	stdcall calling convention.
ccSysCall	SysCall calling convention.

TCallConv is a type describing the calling convention used by a method. It contains an element for all supported calling conventions.

```
TFloatType = (ftSingle,ftDouble,ftExtended,ftComp,ftCurr)
```

Table 78.6: Enumeration values for type TFloatType

Value	Explanation
ftComp	Comp-type float
ftCurr	Currency-type float
ftDouble	Double-sized float
ftExtended	Extended-size float
ftSingle	Single-sized float

The size of a float type.

```
TGetPropValue = function(Instance: TObject;const PropName: string;
                         PreferStrings: Boolean) : Variant
```

The callback function must return the property with name PropName of instance Instance. If PreferStrings is true, it should favour converting the property to a string value. The function needs to return the variant with the property value.

```
TGetVariantProp = function(Instance: TObject;PropInfo: PPropInfo)
                           : Variant
```

The callback function must return the variant property with name PropName of instance Instance.

```
TIntfFlag = (ifHasGuid,ifDispInterface,ifDispatch,ifHasStrGUID)
```

Table 78.7: Enumeration values for type TIntfFlag

Value	Explanation
ifDispatch	Interface is a dispatch interface
ifDispInterface	Interface is a dual dispatch interface
ifHasGuid	Interface has GUID identifier
ifHasStrGUID	Interface has a string GUID identifier

Type of interface.

```
TIntfFlags = Set of TIntfFlag
```

Set of TIntfFlag (661).

```
TIntfFlagsBase = Set of TIntfFlag
```

Set of TIntfFlag (661).

```
TManagedField = packed record
  TypeRef : PTypeInfo;
  FldOffset : SizeInt;
end
```

`TManagedField` describes 1 managed field in a record. It consists of type information (`TypeRef`) and an offset in the record's memory layout (`FldOffset`). Size can be determined from the type information.

```
TMethodKind = (mkProcedure,mkFunction,mkConstructor,mkDestructor,
               mkClassProcedure,mkClassFunction,mkClassConstructor,
               mkClassDestructor,mkOperatorOverload)
```

Table 78.8: Enumeration values for type TMethodKind

Value	Explanation
mkClassConstructor	Class constructor method.
mkClassDestructor	Class destructor method.
mkClassFunction	Class function
mkClassProcedure	Class procedure
mkConstructor	Class constructor
mkDestructor	Class Desctructor
mkFunction	Function method
mkOperatorOverload	Operator overloader
mkProcedure	Procedure method.

Method type description

```
TOrdType = (otSByte,otUByte,otSWord,otUWord,otSLong,otULong)
```

Table 78.9: Enumeration values for type TOrdType

Value	Explanation
otSByte	Signed byte
otSLong	Signed longint
otSWord	Signed word
otUByte	Unsigned byte
otULong	Unsigned longing (Cardinal)
otUWord	Unsigned word

If the property is and ordinal type, then `TOrdType` determines the size and sign of the ordinal type:

```
TParamFlag = (pfVar,pfConst,pfArray,pfAddress,pfReference,pfOut)
```

Table 78.10: Enumeration values for type TParamFlag

Value	Explanation
pfAddress	Parameter is passed by address
pfArray	Parameter is an array parameter
pfConst	Parameter is a const parameter (i.e. cannot be modified)
pfOut	Parameter is a string parameter
pfReference	Parameter is passed by reference
pfVar	Parameter is a var parameter (passed by reference)

`TParamFlag` describes a parameter.

```
TParamFlags = Set of TParamFlag
```

The kind of parameter for a method

```
TProcedureParam = packed record
  Flags : Byte;
  ParamType : PTypeInfo;
  Name : ShortString;
end
```

`TProcedureParam` describes a single parameter to a procedure (or function).

```
TProcInfoProc = procedure(PropInfo: PPropInfo) of object
```

Property info callback method

```
TPropData = packed record
  PropCount : Word;
  PropList : record
    _alignmentdummy : ptrint;
  end;
end
```

The `TPropData` record is not used, but is provided for completeness and compatibility with Delphi.

```
TPropInfo = packed record
  PropType : PTypeInfo;
  GetProc : CodePointer;
  SetProc : CodePointer;
  StoredProc : CodePointer;
  Index : Integer;
  Default : LongInt;
  NameIndex : SmallInt;
  PropProcs : Byte;
  Name : ShortString;
end
```

The TPropInfo record describes one published property of a class. The property information of a class are stored as an array of TPropInfo records.

The Name field is stored not with 255 characters, but with just as many characters as required to store the name.

```
TPropList = Array[0..65535] of PPropInfo
```

Array of property information pointers

```
TSetPropValue = procedure(Instance: TObject;const PropName: string;
                          const Value: Variant)
```

The callback function must set the property with name PropName of instance Instance to Value.

```
TSetVariantProp = procedure(Instance: TObject;PropInfo: PPropInfo;
                            const Value: Variant)
```

The callback function must set the variant property with name PropName of instance to Value.

```
TTypeData = packed record
case TTypeKind of
tkUnKnown: (
);, tkLString: (
);, tkWString: (
);, tkVariant: (
);, tkUString: (
);
tkAString: (
  CodePage : Word;
);
tkInteger: (
  OrdType : TOrdType;
case TTypeKind of
tkInteger: (
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration: (
  BaseType : PTypeInfo;
  NameList : ShortString;
);
);, tkChar: (
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration: (
  BaseType : PTypeInfo;
  NameList : ShortString;
);
);, tkEnumeration: (
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
```

```
tkEnumeration: (
  BaseType : PTypeInfo;
  NameList : ShortString;
);
);, tkBool: (
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration: (
  BaseType : PTypeInfo;
  NameList : ShortString;
);
);, tkWChar: (
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration: (
  BaseType : PTypeInfo;
  NameList : ShortString;
);
);
tkSet: (
  CompType : PTypeInfo;
);
);, tkChar: (
  OrdType : TOrdType;
case TTypeKind of
tkInteger: (
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration: (
  BaseType : PTypeInfo;
  NameList : ShortString;
);
);, tkChar: (
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration: (
  BaseType : PTypeInfo;
  NameList : ShortString;
);
);, tkEnumeration: (
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration: (
  BaseType : PTypeInfo;
  NameList : ShortString;
);
);, tkBool: (
  MinValue : LongInt;
  MaxValue : LongInt;
```

```
case TTypeKind of
tkEnumeration: (
  BaseType : PTypeInfo;
  NameList : ShortString;
);
);, tkWChar: (
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration: (
  BaseType : PTypeInfo;
  NameList : ShortString;
);
);
tkSet: (
  CompType : PTypeInfo;
);
);, tkEnumeration: (
  OrdType : TOrdType;
case TTypeKind of
tkInteger: (
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration: (
  BaseType : PTypeInfo;
  NameList : ShortString;
);
);, tkChar: (
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration: (
  BaseType : PTypeInfo;
  NameList : ShortString;
);
);, tkEnumeration: (
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration: (
  BaseType : PTypeInfo;
  NameList : ShortString;
);
);, tkBool: (
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration: (
  BaseType : PTypeInfo;
  NameList : ShortString;
);
);, tkWChar: (
  MinValue : LongInt;
```

```
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration: (
  BaseType : PTypeInfo;
  NameList : ShortString;
);
);
tkSet: (
  CompType : PTypeInfo;
);
);, tkBool: (
  OrdType : TOrdType;
case TTypeKind of
tkInteger: (
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration: (
  BaseType : PTypeInfo;
  NameList : ShortString;
);
);, tkChar: (
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration: (
  BaseType : PTypeInfo;
  NameList : ShortString;
);
);, tkEnumeration: (
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration: (
  BaseType : PTypeInfo;
  NameList : ShortString;
);
);, tkBool: (
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration: (
  BaseType : PTypeInfo;
  NameList : ShortString;
);
);, tkWChar: (
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration: (
  BaseType : PTypeInfo;
  NameList : ShortString;
);
);
```

```
tkSet: (
  CompType : PTypeInfo;
);
);, tkWChar: (
  OrdType : TOrdType;
case TTypeKind of
tkInteger: (
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration: (
  BaseType : PTypeInfo;
  NameList : ShortString;
);
);, tkChar: (
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration: (
  BaseType : PTypeInfo;
  NameList : ShortString;
);
);, tkEnumeration: (
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration: (
  BaseType : PTypeInfo;
  NameList : ShortString;
);
);, tkBool: (
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration: (
  BaseType : PTypeInfo;
  NameList : ShortString;
);
);, tkWChar: (
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration: (
  BaseType : PTypeInfo;
  NameList : ShortString;
);
);
tkSet: (
  CompType : PTypeInfo;
);
);, tkSet: (
  OrdType : TOrdType;
case TTypeKind of
tkInteger: (
```

```
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration: (
  BaseType : PTypeInfo;
  NameList : ShortString;
);
);, tkChar: (
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration: (
  BaseType : PTypeInfo;
  NameList : ShortString;
);
);, tkEnumeration: (
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration: (
  BaseType : PTypeInfo;
  NameList : ShortString;
);
);, tkBool: (
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration: (
  BaseType : PTypeInfo;
  NameList : ShortString;
);
);, tkWChar: (
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration: (
  BaseType : PTypeInfo;
  NameList : ShortString;
);
);
tkSet: (
  CompType : PTypeInfo;
);
);
tkFloat: (
  FloatType : TFloatType;
);
tkSString: (
  MaxLength : Byte;
);
tkClass: (
  ClassType : TClass;
  ParentInfo : PTypeInfo;
  PropCount : SmallInt;
```

```
  UnitName : ShortString;
);
tkRecord: (
  RecSize : Integer;
  ManagedFldCount : Integer;
);
tkHelper: (
  HelperParent : PTypeInfo;
  ExtendedInfo : PTypeInfo;
  HelperProps : SmallInt;
  HelperUnit : ShortString;
);
tkMethod: (
  MethodKind : TMethodKind;
  ParamCount : Byte;
  ParamList : Array[0..1023] of Char;
);
tkProcVar: (
  ProcSig : TProcedureSignature;
);
tkInt64: (
  MinInt64Value : Int64;
  MaxInt64Value : Int64;
);
tkQWord: (
  MinQWordValue : QWord;
  MaxQWordValue : QWord;
);
tkInterface: (
  IntfParent : PTypeInfo;
  IntfFlags : TIntfFlagsBase;
  GUID : TGUID;
  IntfUnit : ShortString;
);
tkInterfaceRaw: (
  RawIntfParent : PTypeInfo;
  RawIntfFlags : TIntfFlagsBase;
  IID : TGUID;
  RawIntfUnit : ShortString;
  IIDStr : ShortString;
);
tkArray: (
  ArrayData : TArrayTypeData;
);
tkDynArray: (
  elSize : PtrUInt;
  elType2 : PTypeInfo;
  varType : LongInt;
  elType : PTypeInfo;
  DynUnitName : ShortStringBase;
);
tkClassRef: (
  InstanceType : PTypeInfo;
);
```

```
tkPointer: (
  RefType : PTypeInfo;
);
end
```

If the typeinfo kind is `tkClass`, then the property information follows the `UnitName` string, as an array of TPropInfo (664) records.

```
TTypeInfo = record
  Kind : TTypeKind;
  Name : ShortString;
end
```

The `TypeInfo` function returns a pointer to a `TTypeInfo` record.

Note that the Name field is stored with as much bytes as needed to store the name, it is not padded to 255 characters. The type data immediately follows the `TTypeInfo` record as a TTypeData (671) record.

```
TTypeKind = (tkUnknown,tkInteger,tkChar,tkEnumeration,tkFloat,tkSet,
             tkMethod,tkSString,tkLString,tkAString,tkWString,tkVariant,
             tkArray,tkRecord,tkInterface,tkClass,tkObject,tkWChar,
             tkBool,tkInt64,tkQWord,tkDynArray,tkInterfaceRaw,tkProcVar,
             tkUString,tkUChar,tkHelper,tkFile,tkClassRef,tkPointer)
```

Table 78.11: Enumeration values for type TTypeKind

Value	Explanation
tkArray	Array property.
tkAString	Ansistring property.
tkBool	Boolean property.
tkChar	Char property.
tkClass	Class property.
tkClassRef	Class of type
tkDynArray	Dynamical array property.
tkEnumeration	Enumeration type property.
tkFile	File type (both text and binary)
tkFloat	Float property.
tkHelper	Helper class type.
tkInt64	Int64 property.
tkInteger	Integer property.
tkInterface	Interface property.
tkInterfaceRaw	Raw interface property.
tkLString	Longstring property.
tkMethod	Method property.
tkObject	Object property.
tkPointer	Pointer type
tkProcVar	Procedural variable
tkQWord	QWord property.
tkRecord	Record property.
tkSet	Set property.
tkSString	Shortstring property.
tkUChar	Unicode character
tkUnknown	Unknown property type.
tkUString	Unicode string
tkVariant	Variant property.
tkWChar	Widechar property.
tkWString	Widestring property.

Type of a property.

```
TTypeKinds = Set of TTypeKind
```

Set of TTypeKind (671) enumeration.

```
TVmtFieldEntry = packed record
  FieldOffset : PtrUInt;
  TypeIndex : Word;
  Name : ShortString;
end
```

`TVmtFieldEntry` records are generated by the compiler for all fields of a record or class that have RTTI associated with them. They describe the field as known to the compiler.

```
TVmtFieldTable = packed record
```

```
  Count : Word;
  ClassTab : Pointer;
  Fields : Array[0..0] of TVmtFieldEntry;
end
```

`TVmtFieldTable` describes the fields for which RTTI was generated. A `TVmtFieldTable` entry is generated by the compiler in the RTI information, it is not something one creates manually. Basically it contains a list of TVmtFieldEntry (672) values.

78.7 Procedures and functions

78.7.1 FindPropInfo

Synopsis: Return property information by property name.

Declaration:
```
function FindPropInfo(Instance: TObject;const PropName: string)
                        : PPropInfo
function FindPropInfo(Instance: TObject;const PropName: string;
                      AKinds: TTypeKinds) : PPropInfo
function FindPropInfo(AClass: TClass;const PropName: string) : PPropInfo
function FindPropInfo(AClass: TClass;const PropName: string;
                      AKinds: TTypeKinds) : PPropInfo
```

Visibility: default

Description: `FindPropInfo` examines the published property information of a class and returns a pointer to the property information for property `PropName`. The class to be examined can be specified in one of two ways:

AClassa class pointer.

Instancean instance of the class to be investigated.

If the property does not exist, a `EPropertyError` exception will be raised. The GetPropInfo (683) function has the same function as the `FindPropInfo` function, but returns `Nil` if the property does not exist.

Errors: Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: GetPropInfo (683), GetPropList (684), GetPropInfos (683)

Listing: ./typinfex/ex14.pp

```
Program example13;

{ This program demonstrates the FindPropInfo function }

{$mode objfpc}

uses
  rttiobj,typinfo,sysutils;


Var
  O : TMyTestObject;
  PT : PTypeData;
```

```
  PI : PPropInfo;
  I,J : Longint;
  PP : PPropList;
  prI : PPropInfo;

begin
  O:=TMyTestObject.Create;
  PI:=FindPropInfo(O,'BooleanField');
  Writeln('FindPropInfo(Instance,BooleanField) : ',PI^.Name);
  PI:=FindPropInfo(O.ClassType,'ByteField');
  Writeln('FindPropInfo(Class,ByteField)       : ',PI^.Name);
  Write  ('FindPropInfo(Class,NonExistingProp) : ');
  Try
    PI:=FindPropInfo(O,'NonExistingProp');
  except
    On E: Exception do
      Writeln('Caught exception "',E.ClassName,'" with message : ',E.Message);
  end;
  O.Free;
end.
```

78.7.2 GetEnumName

Synopsis: Return name of enumeration constant.

Declaration: `function GetEnumName(TypeInfo: PTypeInfo;Value: Integer) : string`

Visibility: default

Description: `GetEnumName` scans the type information for the enumeration type described by `TypeInfo` and returns the name of the enumeration constant for the element with ordinal value equal to `Value`.

If `Value` is out of range, the first element of the enumeration type is returned. The result is lowercased, but this may change in the future.

This can be used in combination with `GetOrdProp` to stream a property of an enumerated type.

Errors: No check is done to determine whether `TypeInfo` really points to the type information for an enumerated type.

See also: GetOrdProp (682), GetEnumValue (676)

Listing: ./typinfex/ex9.pp

```
program example9;

{ This program demonstrates the GetEnumName, GetEnumValue functions }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;
  TI : PTypeInfo;

begin
  O:=TMyTestObject.Create;
  TI:=GetPropInfo(O,'MyEnumField')^.PropType;
```

```
  WriteLn('GetEnumName                : ',GetEnumName(TI,Ord(O.MyEnumField)));
  WriteLn('GetEnumValue(mefirst) : ',GetEnumName(TI,GetEnumValue(TI,'mefirst')));
  O.Free;
end.
```

78.7.3 GetEnumNameCount

Synopsis: Return number of names in an enumerated type

Declaration: `function GetEnumNameCount(enum1: PTypeInfo) : SizeInt`

Visibility: default

Description: `GetEnumNameCount` returns the number of values (names) in the enumerated type, described by `enum1`

Errors: No checking is done to see whether `Enum1` is really type information of an enumerated type.

See also: GetEnumValue (676), GetEnumName (674)

78.7.4 GetEnumProp

Synopsis: Return the value of an enumeration type property.

Declaration:
```
function GetEnumProp(Instance: TObject;const PropName: string) : string
function GetEnumProp(Instance: TObject;const PropInfo: PPropInfo)
                     : string
```

Visibility: default

Description: `GetEnumProp` returns the value of an property of an enumerated type and returns the name of the enumerated value for the objetc `Instance`. The property whose value must be returned can be specified by its property info in `PropInfo` or by its name in `PropName`

Errors: No check is done to determine whether `PropInfo` really points to the property information for an enumerated type. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: SetEnumProp (693), GetOrdProp (682), GetStrProp (687), GetInt64Prop (677), GetMethodProp (678), GetSetProp (686), GetObjectProp (680), GetEnumProp (675)

Listing: ./typinfex/ex2.pp

```
program example2;

{ This program demonstrates the GetEnumProp function }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;
  TI : PTypeInfo;

begin
```

```
  O:=TMyTestObject.Create;
  PI:=GetPropInfo(O,'MyEnumField');
  TI:=PI^.PropType;
  Writeln('Enum property     : ');
  Writeln('Value                    : ',GetEnumName(TI,Ord(O.MyEnumField)));
  Writeln('Get (name)               : ',GetEnumProp(O,'MyEnumField'));
  Writeln('Get (propinfo)           : ',GetEnumProp(O,PI));
  SetEnumProp(O,'MyEnumField','meFirst');
  Writeln('Set (name,meFirst)       : ',GetEnumName(TI,Ord(O.MyEnumField)));
  SetEnumProp(O,PI,'meSecond');
  Writeln('Set (propinfo,meSecond)  : ',GetEnumName(TI,Ord(O.MyEnumField)));
  O.Free;
end.
```

78.7.5 GetEnumValue

Synopsis: Get ordinal value for enumerated type by name

Declaration: `function GetEnumValue(TypeInfo: PTypeInfo;const Name: string) : Integer`

Visibility: default

Description: `GetEnumValue` scans the type information for the enumeration type described by `TypeInfo` and returns the ordinal value for the element in the enumerated type that has identifier `Name`. The identifier is searched in a case-insensitive manner.

This can be used to set the value of enumerated properties from a stream.

For an example, see GetEnumName (674).

Errors: If `Name` is not found in the list of enumerated values, then -1 is returned. No check is done whether `TypeInfo` points to the type information for an enumerated type.

See also: GetEnumName (674), SetOrdProp (695)

78.7.6 GetFloatProp

Synopsis: Return value of floating point property

Declaration:
```
function GetFloatProp(Instance: TObject;PropInfo: PPropInfo) : Extended
function GetFloatProp(Instance: TObject;const PropName: string)
                      : Extended
```

Visibility: default

Description: `GetFloatProp` returns the value of the float property described by `PropInfo` or with name `Propname` for the object `Instance`. All float types are converted to extended.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid float property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: SetFloatProp (693), GetOrdProp (682), GetStrProp (687), GetInt64Prop (677), GetMethodProp (678), GetSetProp (686), GetObjectProp (680), GetEnumProp (675)

Listing: ./typinfex/ex4.pp

```
program example4;

{ This program demonstrates the GetFloatProp function }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;

begin
  O:=TMyTestObject.Create;
  Writeln('Real property : ');
  PI:=GetPropInfo(O,'RealField');
  Writeln('Value              : ',O.RealField);
  Writeln('Get (name)         : ',GetFloatProp(O,'RealField'));
  Writeln('Get (propinfo)     : ',GetFloatProp(O,PI));
  SetFloatProp(O,'RealField',system.Pi);
  Writeln('Set (name,pi)      : ',O.RealField);
  SetFloatProp(O,PI,exp(1));
  Writeln('Set (propinfo,e)   : ',O.RealField);
  Writeln('Extended property : ');
  PI:=GetPropInfo(O,'ExtendedField');
  Writeln('Value              : ',O.ExtendedField);
  Writeln('Get (name)         : ',GetFloatProp(O,'ExtendedField'));
  Writeln('Get (propinfo)     : ',GetFloatProp(O,PI));
  SetFloatProp(O,'ExtendedField',system.Pi);
  Writeln('Set (name,pi)      : ',O.ExtendedField);
  SetFloatProp(O,PI,exp(1));
  Writeln('Set (propinfo,e)   : ',O.ExtendedField);
  O.Free;
end.
```

78.7.7 GetInt64Prop

Synopsis: return value of an Int64 property

Declaration:
```
function GetInt64Prop(Instance: TObject;PropInfo: PPropInfo) : Int64
function GetInt64Prop(Instance: TObject;const PropName: string) : Int64
```

Visibility: default

Description: Publishing of Int64 properties is not yet supported by Free Pascal. This function is provided for Delphi compatibility only at the moment.

`GetInt64Prop` returns the value of the property of type `Int64` that is described by `PropInfo` or with name `Propname` for the object `Instance`.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid `Int64` property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception

See also: SetInt64Prop (694), GetOrdProp (682), GetStrProp (687), GetFloatProp (676), GetMethodProp (678), GetSetProp (686), GetObjectProp (680), GetEnumProp (675)

Listing: ./typinfex/ex15.pp

```
program example15;

{ This program demonstrates the GetInt64Prop function }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;

begin
  O:=TMyTestObject.Create;
  Writeln('Int64 property : ');
  PI:=GetPropInfo(O,'Int64Field');
  Writeln('Value              : ',O.Int64Field);
  Writeln('Get (name)         : ',GetInt64Prop(O,'Int64Field'));
  Writeln('Get (propinfo)     : ',GetInt64Prop(O,PI));
  SetInt64Prop(O,'Int64Field',12345);
  Writeln('Set (name,12345)     : ',O.Int64Field);
  SetInt64Prop(O,PI,54321);
  Writeln('Set (propinfo,54321) : ',O.Int64Field);
  O.Free;
end.
```

78.7.8 GetInterfaceProp

Synopsis: Return interface-typed property

Declaration:
```
function GetInterfaceProp(Instance: TObject;const PropName: string)
                          : IInterface
function GetInterfaceProp(Instance: TObject;PropInfo: PPropInfo)
                          : IInterface
```

Visibility: default

Description: `GetInterfaceProp` returns the interface which the property described by `PropInfo` or with name `Propname` points to for object `Instance`.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid method property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: SetInterfaceProp (694), GetOrdProp (682), GetStrProp (687), GetFloatProp (676), GetInt64Prop (677), GetSetProp (686), GetObjectProp (680), GetEnumProp (675)

78.7.9 GetMethodProp

Synopsis: Return value of a method property

Declaration:
```
function GetMethodProp(Instance: TObject;PropInfo: PPropInfo) : TMethod
function GetMethodProp(Instance: TObject;const PropName: string)
                       : TMethod
```

Visibility: default

Description: `GetMethodProp` returns the method the property described by `PropInfo` or with name `Propname` for object `Instance`. The return type `TMethod` is defined in the SysUtils unit as:

```
TMethod = packed record
  Code, Data: Pointer;
end;
```

`Data` points to the instance of the class with the method `Code`.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid method property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: SetMethodProp (695), GetOrdProp (682), GetStrProp (687), GetFloatProp (676), GetInt64Prop (677), GetSetProp (686), GetObjectProp (680), GetEnumProp (675)

Listing: ./typinfex/ex6.pp

```
program example6;

{ This program demonstrates the GetMethodProp function }

{$mode objfpc}

uses rttiobj,typinfo,sysutils;

Type
  TNotifyObject = Class(TObject)
    Procedure Notification1(Sender : TObject);
    Procedure Notification2(Sender : TObject);
  end;

Procedure TNotifyObject.Notification1(Sender : TObject);

begin
  Write('Received notification 1 of object with class: ');
  Writeln(Sender.ClassName);
end;

Procedure TNotifyObject.Notification2(Sender : TObject);

begin
  Write('Received notification 2 of object with class: ');
  Writeln(Sender.ClassName);
end;

Var
  O : TMyTestObject;
  PI : PPropInfo;
  NO : TNotifyObject;
  M : TMethod;

Procedure PrintMethod (Const M : TMethod);

begin
  If (M.Data=Pointer(NO)) Then
```

```
    If (M.Code=Pointer(@TNotifyObject.Notification1)) then
      WriteIn('Notification1')
    else If (M.Code=Pointer(@TNotifyObject.Notification2)) then
      WriteIn('Notification2')
    else
      begin
      Write('Unknown method adress (data:');
      Write(hexStr(Longint(M.data),8));
      WriteIn(',code:',hexstr(Longint(M.Code),8),')');
      end;
end;


begin
  O:=TMyTestObject.Create;
  NO:=TNotifyObject.Create;
  O.NotifyEvent:=@NO.Notification1;
  PI:=GetPropInfo(O,'NotifyEvent');
  WriteIn('Method property : ');
  Write('Notifying                    : ');
  O.Notify;
  Write('Get (name)                   : ');
  M:=GetMethodProp(O,'NotifyEvent');
  PrintMethod(M);
  Write('Notifying                    : ');
  O.Notify;
  Write('Get (propinfo)               : ');
  M:=GetMethodProp(O,PI);
  PrintMethod(M);
  M:=TMethod(@NO.Notification2);
  SetMethodProp(O,'NotifyEvent',M);
  Write('Set (name,Notification2)     : ');
  M:=GetMethodProp(O,PI);
  PrintMethod(M);
  Write('Notifying                    : ');
  O.Notify;
  Write('Set (propinfo,Notification1) : ');
  M:=TMethod(@NO.Notification1);
  SetMethodProp(O,PI,M);
  M:=GetMethodProp(O,PI);
  PrintMethod(M);
  Write('Notifying                    : ');
  O.Notify;
  O.Free;
end.
```

78.7.10 GetObjectProp

Synopsis: Return value of an object-type property.

Declaration:
```
function GetObjectProp(Instance: TObject;const PropName: string)
                       : TObject
function GetObjectProp(Instance: TObject;const PropName: string;
                       MinClass: TClass) : TObject
function GetObjectProp(Instance: TObject;PropInfo: PPropInfo) : TObject
function GetObjectProp(Instance: TObject;PropInfo: PPropInfo;
                       MinClass: TClass) : TObject
```

Visibility: default

Description: `GetObjectProp` returns the object which the property described by `PropInfo` with name `Propname` points to for object `Instance`.

If `MinClass` is specified, then if the object is not descendent of class `MinClass`, then `Nil` is returned.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid method property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: SetMethodProp (695), GetOrdProp (682), GetStrProp (687), GetFloatProp (676), GetInt64Prop (677), GetSetProp (686), GetObjectProp (680), GetEnumProp (675)

Listing: ./typinfex/ex5.pp

```
program example5;

{ This program demonstrates the GetObjectProp function }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;
  NO1,NO2 : TNamedObject;

begin
  O:=TMyTestObject.Create;
  NO1:=TNamedObject.Create;
  NO1.ObjectName:='First named object';
  NO2:=TNamedObject.Create;
  NO2.ObjectName:='Second named object';
  O.ObjField:=NO1;
  Writeln('Object property : ');
  PI:=GetPropInfo(O,'ObjField');
  Write('Property class      : ');
  Writeln(GetObjectPropClass(O,'ObjField').ClassName);
  Write('Value               : ');
  Writeln((O.ObjField as TNamedObject).ObjectName);
  Write('Get (name)          : ');
  Writeln((GetObjectProp(O,'ObjField') As TNamedObject).ObjectName);
  Write('Get (propinfo)      : ');
  Writeln((GetObjectProp(O,PI,TObject) as TNamedObject).ObjectName);
  SetObjectProp(O,'ObjField',NO2);
  Write('Set (name,NO2)      : ');
  Writeln((O.ObjField as TNamedObject).ObjectName);
  SetObjectProp(O,PI,NO1);
  Write('Set (propinfo,NO1)  : ');
  Writeln((O.ObjField as TNamedObject).ObjectName);
  O.Free;
end.
```

78.7.11 GetObjectPropClass

Synopsis: Return class of property.

Declaration:
```
function GetObjectPropClass(Instance: TObject;const PropName: string)
                                 : TClass
function GetObjectPropClass(AClass: TClass;const PropName: string)
                                 : TClass
```

Visibility: default

Description: `GetObjectPropClass` returns the declared class of the property with name `PropName`. This may not be the actual class of the property value.

For an example, see GetObjectProp (680).

Errors: No checking is done whether `Instance` is non-nil. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: SetMethodProp (695), GetOrdProp (682), GetStrProp (687), GetFloatProp (676), GetInt64Prop (677)

78.7.12 GetOrdProp

Synopsis: Get the value of an ordinal property

Declaration:
```
function GetOrdProp(Instance: TObject;PropInfo: PPropInfo) : Int64
function GetOrdProp(Instance: TObject;const PropName: string) : Int64
```

Visibility: default

Description: `GetOrdProp` returns the value of the ordinal property described by `PropInfo` or with name `PropName` for the object `Instance`. The value is returned as a longint, which should be typecasted to the needed type.

Ordinal properties that can be retrieved include:

Integers and subranges of integers The value of the integer will be returned.

Enumerated types and subranges of enumerated types The ordinal value of the enumerated type will be returned.

Sets If the base type of the set has less than 31 possible values. If a bit is set in the return value, then the corresponding element of the base ordinal class of the set type must be included in the set.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid ordinal property of `Instance` Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: SetOrdProp (695), GetStrProp (687), GetFloatProp (676), GetInt64Prop (677), GetMethodProp (678), GetSetProp (686), GetObjectProp (680), GetEnumProp (675)

Listing: ./typinfex/ex1.pp

```
program example1;

{ This program demonstrates the GetOrdProp function }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;
```

```
  PI : PPropInfo;

begin
  O:=TMyTestObject.Create;
  Writeln('Boolean property     : ');
  Writeln('Value                : ',O.BooleanField);
  Writeln('Ord(Value)           : ',Ord(O.BooleanField));
  Writeln('Get (name)           : ',GetOrdProp(O,'BooleanField'));
  PI:=GetPropInfo(O,'BooleanField');
  Writeln('Get (propinfo)       : ',GetOrdProp(O,PI));
  SetOrdProp(O,'BooleanField',Ord(False));
  Writeln('Set (name,false)     : ',O.BooleanField);
  SetOrdProp(O,PI,Ord(True));
  Writeln('Set (propinfo,true) : ',O.BooleanField);
  O.Free;
end.
```

78.7.13 GetPropInfo

Synopsis: Return property type information, by property name.

Declaration:
```
function GetPropInfo(TypeInfo: PTypeInfo;const PropName: string)
                     : PPropInfo
function GetPropInfo(TypeInfo: PTypeInfo;const PropName: string;
                     AKinds: TTypeKinds) : PPropInfo
function GetPropInfo(Instance: TObject;const PropName: string)
                     : PPropInfo
function GetPropInfo(Instance: TObject;const PropName: string;
                     AKinds: TTypeKinds) : PPropInfo
function GetPropInfo(AClass: TClass;const PropName: string) : PPropInfo
function GetPropInfo(AClass: TClass;const PropName: string;
                     AKinds: TTypeKinds) : PPropInfo
```

Visibility: default

Description: `GetPropInfo` returns a pointer to the `TPropInfo` record for the `PropName` property of a class. The class to examine can be specified in one of three ways:

InstanceAn instance of the class.

AClassA class pointer to the class.

TypeInfoA pointer to the type information of the class.

In each of these three ways, if `AKinds` is specified, if the property has `TypeKind` which is not included in `Akinds`, `Nil` will be returned.

For an example, see most of the other functions.

Errors: If the property `PropName` does not exist, `Nil` is returned.

See also: GetPropInfos (683), GetPropList (684)

78.7.14 GetPropInfos

Synopsis: Return a list of published properties.

Declaration:
```
procedure GetPropInfos(TypeInfo: PTypeInfo;PropList: PPropList)
```

Visibility: default

Description: `GetPropInfos` stores pointers to the property information of all published properties of a class with class info `TypeInfo` in the list pointed to by `Proplist`. The `PropList` pointer must point to a memory location that contains enough space to hold all properties of the class and its parent classes.

Errors: No checks are done to see whether `PropList` points to a memory area that is big enough to hold all pointers.

See also: GetPropInfo (683), GetPropList (684)

Listing: ./typinfex/ex12.pp

```
Program example12;

{ This program demonstrates the GetPropInfos function }

uses
  rttiobj,typinfo;


Var
  O : TMyTestObject;
  PT : PTypeData;
  PI : PTypeInfo;
  I,J : Longint;
  PP : PPropList;
  prI : PPropInfo;

begin
  O:=TMyTestObject.Create;
  PI:=O.ClassInfo;
  PT:=GetTypeData(PI);
  Writeln('Property Count : ',PT^.PropCount);
  GetMem (PP,PT^.PropCount*SizeOf(Pointer));
  GetPropInfos(PI,PP);
  For I:=0 to PT^.PropCount-1 do
    begin
    With PP^[i]^ do
      begin
      Write('Property ',i+1:3,': ',name:30);
      writeln('  Type: ',TypeNames[typinfo.PropType(O,Name)]);
      end;
    end;
  FreeMem(PP);
  O.Free;
end.
```

78.7.15 GetPropList

Synopsis: Return a list of a certain type of published properties.

Declaration:
```
function GetPropList(TypeInfo: PTypeInfo;TypeKinds: TTypeKinds;
                     PropList: PPropList;Sorted: Boolean) : LongInt
function GetPropList(TypeInfo: PTypeInfo;out PropList: PPropList)
                     : SizeInt
```

```
function GetPropList(AClass: TClass;out PropList: PPropList) : Integer
function GetPropList(Instance: TObject;out PropList: PPropList)
                     : Integer
```

Visibility: default

Description: `GetPropList` stores pointers to property information of the class with class info `TypeInfo` for properties of kind `TypeKinds` in the list pointed to by `Proplist`. `PropList` must contain enough space to hold all properties.

The function returns the number of pointers that matched the criteria and were stored in `PropList`.

Errors: No checks are done to see whether `PropList` points to a memory area that is big enough to hold all pointers.

See also: GetPropInfos (683), GetPropInfo (683)

Listing: ./typinfex/ex13.pp

```
Program example13;

{ This program demonstrates the GetPropList function }

uses
  rttiobj,typinfo;


Var
  O : TMyTestObject;
  PT : PTypeData;
  PI : PTypeInfo;
  I,J : Longint;
  PP : PPropList;
  prI : PPropInfo;

begin
  O:=TMyTestObject.Create;
  PI:=O.ClassInfo;
  PT:=GetTypeData(PI);
  Writeln('Total property Count : ',PT^.PropCount);
  GetMem (PP,PT^.PropCount*SizeOf(Pointer));
  J:=GetPropList(PI,OrdinalTypes,PP);
  Writeln('Ordinal property Count : ',J);
  For I:=0 to J-1 do
    begin
    With PP^[i]^ do
      begin
      Write('Property ',i+1:3,': ',name:30);
      writeln('  Type: ',TypeNames[typinfo.PropType(O,Name)]);
      end;
    end;
  FreeMem(PP);
  O.Free;
end.
```

78.7.16 GetPropValue

Synopsis: Get property value as a string.

Declaration:
```
function GetPropValue(Instance: TObject;const PropName: string)
                          : Variant
function GetPropValue(Instance: TObject;const PropName: string;
                      PreferStrings: Boolean) : Variant
```

Visibility: default

Description: Due to missing `Variant` support, `GetPropValue` is not yet implemented. The declaration is provided for compatibility with Delphi.

78.7.17 GetRawInterfaceProp

Synopsis: Get a raw (CORBA) interface property.

Declaration:
```
function GetRawInterfaceProp(Instance: TObject;const PropName: string)
                                : Pointer
function GetRawInterfaceProp(Instance: TObject;PropInfo: PPropInfo)
                                : Pointer
```

Visibility: default

Description: `GetRawInterfaceProp` can be used to retrieve the value of a published CORBA interface property with name `PropName` from object `Instance`. Alternatively, the required property information can be specified by `PropInfo` instead of the property name. In difference with the GetInterfaceProp (678) function, no reference counting is done.

Errors: If the property `PropName` does not exist, an `EPropertyError` exception is raised.

See also: GetInterfaceProp (678), SetRawInterfaceProp (696)

78.7.18 GetSetProp

Synopsis: Return the value of a set property.

Declaration:
```
function GetSetProp(Instance: TObject;const PropName: string) : string
function GetSetProp(Instance: TObject;const PropName: string;
                    Brackets: Boolean) : string
function GetSetProp(Instance: TObject;const PropInfo: PPropInfo;
                    Brackets: Boolean) : string
```

Visibility: default

Description: `GetSetProp` returns the contents of a set property as a string. The property to be returned can be specified by it's name in `PropName` or by its property information in `PropInfo`.

The returned set is a string representation of the elements in the set as returned by SetToString (697). The `Brackets` option can be used to enclose the string representation in square brackets.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid ordinal property of `Instance` Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: SetSetProp (697), GetStrProp (687), GetFloatProp (676), GetInt64Prop (677), GetMethodProp (678)

Listing: ./typinfex/ex7.pp

```
program example7;

{ This program demonstrates the GetSetProp function }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;

Function SetAsString (ASet : TMyEnums) : String;

Var
  i : TmyEnum;

begin
  result:='';
  For i:=mefirst to methird do
    If i in ASet then
      begin
      If (Result<>'') then
        Result:=Result+',';
      Result:=Result+MyEnumNames[i];
      end;
end;

Var
  S : TMyEnums;

begin
  O:=TMyTestObject.Create;
  O.SetField:=[mefirst,meSecond,meThird];
  Writeln('Set property     : ');
  Writeln('Value                          : ',SetAsString(O.SetField));
  Writeln('Ord(Value)                     : ',Longint(O.SetField));
  Writeln('Get (name)                     : ',GetSetProp(O,'SetField'));
  PI:=GetPropInfo(O,'SetField');
  Writeln('Get (propinfo)                 : ',GetSetProp(O,PI,false));
  S:=[meFirst,meThird];
  SetOrdProp(O,'SetField',Integer(S));
  Write('Set (name,[mefirst,methird]) : ');
  Writeln(SetAsString(O.SetField));
  S:=[meSecond];
  SetOrdProp(O,PI,Integer(S));
  Write('Set (propinfo,[meSecond])    : ');
  Writeln(SetAsString(O.SetField));
  O.Free;
end.
```

78.7.19 GetStrProp

Synopsis: Return the value of a string property.

Declaration: `function GetStrProp(Instance: TObject;PropInfo: PPropInfo) : Ansistring`

```
function GetStrProp(Instance: TObject;const PropName: string) : string
```

Visibility: default

Description: `GetStrProp` returns the value of the string property described by `PropInfo` or with name `PropName` for object `Instance`.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid string property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: SetStrProp (697), SetWideStrProp (699), GetOrdProp (682), GetFloatProp (676), GetInt64Prop (677), GetMethodProp (678)

Listing: ./typinfex/ex3.pp

```
program example3;

{ This program demonstrates the GetStrProp function }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;

begin
  O:=TMyTestObject.Create;
  PI:=GetPropInfo(O,'AnsiStringField');
  Writeln('String property : ');
  Writeln('Value                    : ',O.AnsiStringField);
  Writeln('Get (name)               : ',GetStrProp(O,'AnsiStringField'));
  Writeln('Get (propinfo)           : ',GetStrProp(O,PI));
  SetStrProp(O,'AnsiStringField','First');
  Writeln('Set (name,''First'')       : ',O.AnsiStringField);
  SetStrProp(O,PI,'Second');
  Writeln('Set (propinfo,''Second'') : ',O.AnsiStringField);
  O.Free;
end.
```

78.7.20 GetTypeData

Synopsis: Return a pointer to type data, based on type information.

Declaration: `function GetTypeData(TypeInfo: PTypeInfo) : PTypeData`

Visibility: default

Description: `GetTypeData` returns a pointer to the `TTypeData` record that follows after the `TTypeInfo` record pointed to by `TypeInfo`. It essentially skips the `Kind` and `Name` fields in the `TTypeInfo` record.

Errors: None.

78.7.21 GetUnicodeStrProp

Synopsis: Get UnicodeString-valued property

Declaration:
```
function GetUnicodeStrProp(Instance: TObject;PropInfo: PPropInfo)
                           : UnicodeString
function GetUnicodeStrProp(Instance: TObject;const PropName: string)
                           : UnicodeString
```

Visibility: default

Description: `GetUnicodeStrProp` returns the UnicodeString property from `Instance`, where the property is identified by the `PropInfo` pointer or the `PropertyName`.

Errors: If no property of the indicated name exists, or the value is not a unicode string, an exception will occur.

See also: GetStrProp (687), SetUnicodeStrProp (698)

78.7.22 GetVariantProp

Synopsis: Return the value of a variant property.

Declaration:
```
function GetVariantProp(Instance: TObject;PropInfo: PPropInfo) : Variant
function GetVariantProp(Instance: TObject;const PropName: string)
                        : Variant
```

Visibility: default

Description: Due to missing Variant support, the `GetVariantProp` function is not yet implemented. Provided for Delphi compatibility only.

See also: SetVariantProp (699)

78.7.23 GetWideStrProp

Synopsis: Read a widestring property

Declaration:
```
function GetWideStrProp(Instance: TObject;PropInfo: PPropInfo)
                        : WideString
function GetWideStrProp(Instance: TObject;const PropName: string)
                        : WideString
```

Visibility: default

Description: `GetWideStrProp` returns the value of the widestring property described by `PropInfo` or with name `PropName` for object `Instance`.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid widestring property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: GetStrProp (687), SetWideStrProp (699), GetOrdProp (682), GetFloatProp (676), GetInt64Prop (677), GetMethodProp (678)

78.7.24 IsPublishedProp

Synopsis: Check whether a published property exists.

Declaration:
```
function IsPublishedProp(Instance: TObject;const PropName: string)
                         : Boolean
function IsPublishedProp(AClass: TClass;const PropName: string)
                         : Boolean
```

Visibility: default

Description: `IsPublishedProp` returns true if a class has a published property with name `PropName`. The class can be specfied in one of two ways:

AClass A class pointer to the class.

Instance An instance of the class.

Errors: No checks are done to ensure `Instance` or `AClass` are valid pointers. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: IsStoredProp (690), PropIsType (691)

Listing: ./typinfex/ex10.pp

```
program example10;

{ This program demonstrates the IsPublishedProp function }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;

begin
  O:=TMyTestObject.Create;
  Writeln('Property tests     : ');
  Write('IsPublishedProp(O,BooleanField)      : ');
  Writeln(IsPublishedProp(O,'BooleanField'));
  Write('IsPublishedProp(Class,BooleanField) : ');
  Writeln(IsPublishedProp(O.ClassType,'BooleanField'));
  Write('IsPublishedProp(O,SomeField)         : ');
  Writeln(IsPublishedProp(O,'SomeField'));
  Write('IsPublishedProp(Class,SomeField)     : ');
  Writeln(IsPublishedProp(O.ClassType,'SomeField'));
  O.Free;
end.
```

78.7.25 IsStoredProp

Synopsis: Check whether a property is stored.

Declaration:
```
function IsStoredProp(Instance: TObject;PropInfo: PPropInfo) : Boolean
function IsStoredProp(Instance: TObject;const PropName: string)
                      : Boolean
```

Visibility: default

Description: `IsStoredProp` returns `True` if the `Stored` modifier evaluates to `True` for the property described by `PropInfo` or with name `PropName` for object `Instance`. It returns `False` otherwise. If the function returns `True`, this indicates that the property should be written when streaming the object `Instance`.

If there was no `stored` modifier in the declaration of the property, `True` will be returned.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: IsPublishedProp (690), PropIsType (691)

Listing: ./typinfex/ex11.pp

```
program example11;

{ This program demonstrates the IsStoredProp function }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;

begin
  O:=TMyTestObject.Create;
  Writeln('Stored tests     : ');
  Write('IsStoredProp(O,StoredIntegerConstFalse)     : ');
  Writeln(IsStoredProp(O,'StoredIntegerConstFalse'));
  Write('IsStoredProp(O,StoredIntegerConstTrue)      : ');
  Writeln(IsStoredProp(O,'StoredIntegerConstTrue'));
  Write('IsStoredProp(O,StoredIntegerMethod)         : ');
  Writeln(IsStoredProp(O,'StoredIntegerMethod'));
  Write('IsStoredProp(O,StoredIntegerVirtualMethod) : ');
  Writeln(IsStoredProp(O,'StoredIntegerVirtualMethod'));
  O.Free;
end.
```

78.7.26 PropIsType

Synopsis: Check the type of a published property.

Declaration:
```
function PropIsType(Instance: TObject;const PropName: string;
                    TypeKind: TTypeKind) : Boolean
function PropIsType(AClass: TClass;const PropName: string;
                    TypeKind: TTypeKind) : Boolean
```

Visibility: default

Description: `PropIsType` returns `True` if the property with name `PropName` has type `TypeKind`. It returns `False` otherwise. The class to be examined can be specified in one of two ways:

AClassA class pointer.

InstanceAn instance of the class.

Errors: No checks are done to ensure `Instance` or `AClass` are valid pointers.Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: IsPublishedProp (690), IsStoredProp (690), PropType (692)

Listing: ./typinfex/ex16.pp

```
program example16;

{ This program demonstrates the PropIsType function }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;

begin
  O:=TMyTestObject.Create;
  Writeln('Property tests     : ');
  Write('PropIsType(O,BooleanField,tkBool)      : ');
  Writeln(PropIsType(O,'BooleanField',tkBool));
  Write('PropIsType(Class,BooleanField,tkBool) : ');
  Writeln(PropIsType(O.ClassType,'BooleanField',tkBool));
  Write('PropIsType(O,ByteField,tkString)       : ');
  Writeln(PropisType(O,'ByteField',tkString));
  Write('PropIsType(Class,ByteField,tkString)  : ');
  Writeln(PropIsType(O.ClassType,'ByteField',tkString));
  O.Free;
end.
```

78.7.27 PropType

Synopsis: Return the type of a property

Declaration:
```
function PropType(Instance: TObject;const PropName: string) : TTypeKind
function PropType(AClass: TClass;const PropName: string) : TTypeKind
```

Visibility: default

Description: `Proptype` returns the type of the property `PropName` for a class. The class to be examined can be specified in one of 2 ways:

AClassA class pointer.

InstanceAn instance of the class.

Errors: No checks are done to ensure `Instance` or `AClass` are valid pointers. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: IsPublishedProp (690), IsStoredProp (690), PropIsType (691)

Listing: ./typinfex/ex17.pp

```
program example17;

{ This program demonstrates the PropType function }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;

begin
  O:=TMyTestObject.Create;
  Writeln('Property tests     : ');
  Write('PropType(O,BooleanField)      : ');
  Writeln(TypeNames[PropType(O,'BooleanField')]);
  Write('PropType(Class,BooleanField) : ');
  Writeln(TypeNames[PropType(O.ClassType,'BooleanField')]);
  Write('PropType(O,ByteField)         : ');
  Writeln(TypeNames[PropType(O,'ByteField')]);
  Write('PropType(Class,ByteField)     : ');
  Writeln(TypeNames[PropType(O.ClassType,'ByteField')]);
  O.Free;
end.
```

78.7.28 SetEnumProp

Synopsis: Set value of an enumerated-type property

Declaration:
```
procedure SetEnumProp(Instance: TObject;const PropName: string;
                      const Value: string)
procedure SetEnumProp(Instance: TObject;const PropInfo: PPropInfo;
                      const Value: string)
```

Visibility: default

Description: `SetEnumProp` sets the property described by `PropInfo` or with name `PropName` to `Value`. `Value` must be a string with the name of the enumerate value, i.e. it can be used as an argument to GetEnumValue (676).

For an example, see GetEnumProp (675).

Errors: No checks are done to ensure `Instance` or `PropInfo` are valid pointers. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: GetEnumProp (675), SetStrProp (697), SetFloatProp (693), SetInt64Prop (694), SetMethodProp (695)

78.7.29 SetFloatProp

Synopsis: Set value of a float property.

Declaration:
```
procedure SetFloatProp(Instance: TObject;const PropName: string;
                       Value: Extended)
procedure SetFloatProp(Instance: TObject;PropInfo: PPropInfo;
                       Value: Extended)
```

Visibility: default

Description: `SetFloatProp` assigns `Value` to the property described by `PropInfo` or with name `Propname` for the object `Instance`.

For an example, see GetFloatProp (676).

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid float property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: GetFloatProp (676), SetOrdProp (695), SetStrProp (697), SetInt64Prop (694), SetMethodProp (695)

78.7.30 SetInt64Prop

Synopsis: Set value of a Int64 property

Declaration:
```
procedure SetInt64Prop(Instance: TObject;PropInfo: PPropInfo;
                       const Value: Int64)
procedure SetInt64Prop(Instance: TObject;const PropName: string;
                       const Value: Int64)
```

Visibility: default

Description: `SetInt64Prop` assigns `Value` to the property of type `Int64` that is described by `PropInfo` or with name `Propname` for the object `Instance`.

For an example, see GetInt64Prop (677).

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid `Int64` property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: GetInt64Prop (677), GetMethodProp (678), SetOrdProp (695), SetStrProp (697), SetFloatProp (693)

78.7.31 SetInterfaceProp

Synopsis: Set interface-valued property

Declaration:
```
procedure SetInterfaceProp(Instance: TObject;const PropName: string;
                           const Value: IInterface)
procedure SetInterfaceProp(Instance: TObject;PropInfo: PPropInfo;
                           const Value: IInterface)
```

Visibility: default

Description: `SetInterfaceProp` assigns `Value` to the object property described by `PropInfo` or with name `Propname` for the object `Instance`.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid interface property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: GetInterfaceProp (678), SetObjectProp (695), SetOrdProp (695), SetStrProp (697), SetFloatProp (693), SetInt64Prop (694), SetMethodProp (695)

78.7.32 SetMethodProp

Synopsis: Set the value of a method property

Declaration:
```
procedure SetMethodProp(Instance: TObject;PropInfo: PPropInfo;
                        const Value: TMethod)
procedure SetMethodProp(Instance: TObject;const PropName: string;
                        const Value: TMethod)
```

Visibility: default

Description: `SetMethodProp` assigns `Value` to the method the property described by `PropInfo` or with name `Propname` for object `Instance`.

The type `TMethod` of the `Value` parameter is defined in the SysUtils unit as:

```
TMethod = packed record
  Code, Data: Pointer;
end;
```

`Data` should point to the instance of the class with the method `Code`.

For an example, see GetMethodProp (678).

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid method property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: GetMethodProp (678), SetOrdProp (695), SetStrProp (697), SetFloatProp (693), SetInt64Prop (694)

78.7.33 SetObjectProp

Synopsis: Set the value of an object-type property.

Declaration:
```
procedure SetObjectProp(Instance: TObject;const PropName: string;
                        Value: TObject)
procedure SetObjectProp(Instance: TObject;PropInfo: PPropInfo;
                        Value: TObject)
```

Visibility: default

Description: `SetObjectProp` assigns `Value` to the object property described by `PropInfo` or with name `Propname` for the object `Instance`.

For an example, see GetObjectProp (680).

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid object property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: GetObjectProp (680), SetOrdProp (695), SetStrProp (697), SetFloatProp (693), SetInt64Prop (694), SetMethodProp (695)

78.7.34 SetOrdProp

Synopsis: Set value of an ordinal property

Declaration:
```
procedure SetOrdProp(Instance: TObject;PropInfo: PPropInfo;Value: Int64)
procedure SetOrdProp(Instance: TObject;const PropName: string;
                     Value: Int64)
```

Visibility: default

Description: `SetOrdProp` assigns `Value` to the ordinal property described by `PropInfo` or with name `Propname` for the object `Instance`.

Ordinal properties that can be set include:

Integers and subranges of integers The actual value of the integer must be passed.

Enumerated types and subranges of enumerated types The ordinal value of the enumerated type must be passed.

Subrange typesof integers or enumerated types. Here the ordinal value must be passed.

SetsIf the base type of the set has less than 31 possible values. For each possible value; the corresponding bit of `Value` must be set.

For an example, see GetOrdProp (682).

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid ordinal property of `Instance`. No range checking is performed. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: GetOrdProp (682), SetStrProp (697), SetFloatProp (693), SetInt64Prop (694), SetMethodProp (695)

78.7.35 SetPropValue

Synopsis: Set property value as variant

Declaration:
```
procedure SetPropValue(Instance: TObject;const PropName: string;
                       const Value: Variant)
```

Visibility: default

Description: Due to missing Variant support, this function is not yet implemented; it is provided for Delphi compatibility only.

78.7.36 SetRawInterfaceProp

Synopsis: Set a raw (CORBA) interface property.

Declaration:
```
procedure SetRawInterfaceProp(Instance: TObject;const PropName: string;
                              const Value: Pointer)
procedure SetRawInterfaceProp(Instance: TObject;PropInfo: PPropInfo;
                              const Value: Pointer)
```

Visibility: default

Description: `SetRawInterfaceProp` can be used to set the value of a published CORBA interface with name `PropName` from object `Instance` to `Value`. Alternatively, the required property information can be specified by `PropInfo` instead of the property name. In difference with the SetInterfaceProp (694) procedure, no reference counting is done.

Errors: If the property `PropName` does not exist, an `EPropertyError` exception is raised.

See also: SetInterfaceProp (694), GetRawInterfaceProp (686)

78.7.37 SetSetProp

Synopsis: Set value of set-typed property.

Declaration:
```
procedure SetSetProp(Instance: TObject;const PropName: string;
                     const Value: string)
procedure SetSetProp(Instance: TObject;const PropInfo: PPropInfo;
                     const Value: string)
```

Visibility: default

Description: `SetSetProp` sets the property specified by `PropInfo` or `PropName` for object `Instance` to `Value`. `Value` is a string which contains a comma-separated list of values, each value being a string-representation of the enumerated value that should be included in the set. The value should be accepted by the StringToSet (699) function.

The value can be formed using the SetToString (697) function.

For an example, see GetSetProp (686).

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid ordinal property of `Instance`. No range checking is performed. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: GetSetProp (686), SetOrdProp (695), SetStrProp (697), SetFloatProp (693), SetInt64Prop (694), SetMethodProp (695), SetToString (697), StringToSet (699)

78.7.38 SetStrProp

Synopsis: Set value of a string property

Declaration:
```
procedure SetStrProp(Instance: TObject;const PropName: string;
                     const Value: AnsiString)
procedure SetStrProp(Instance: TObject;PropInfo: PPropInfo;
                     const Value: Ansistring)
```

Visibility: default

Description: `SetStrProp` assigns `Value` to the string property described by `PropInfo` or with name `Propname` for object `Instance`.

For an example, see GetStrProp (687)

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid string property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: GetStrProp (687), SetWideStrProp (699), SetOrdProp (695), SetFloatProp (693), SetInt64Prop (694), SetMethodProp (695)

78.7.39 SetToString

Synopsis: Convert set to a string description

Declaration:
```
function SetToString(TypeInfo: PTypeInfo;Value: Integer;
                     Brackets: Boolean) : string
function SetToString(PropInfo: PPropInfo;Value: Integer;
                     Brackets: Boolean) : string
function SetToString(PropInfo: PPropInfo;Value: Integer) : string
```

Visibility: default

Description: `SetToString` takes an integer representation of a set (as received e.g. by `GetOrdProp`) and turns it into a string representing the elements in the set, based on the type information found in the `PropInfo` property information. By default, the string representation is not surrounded by square brackets. Setting the `Brackets` parameter to `True` will surround the string representation with brackets.

The function returns the string representation of the set.

Errors: No checking is done to see whether `PropInfo` points to valid property information.

See also: GetEnumName (674), GetEnumValue (676), StringToSet (699)

Listing: ./typinfex/ex18.pp

```
program example18;

{ This program demonstrates the SetToString function }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;
  I : longint;

begin
  O:=TMyTestObject.Create;
  PI:=GetPropInfo(O,'SetField');
  O.SetField:=[mefirst,meSecond,meThird];
  I:=GetOrdProp(O,PI);
  Writeln('Set property to string : ');
  Writeln('Value  : ',SetToString(PI,I,False));
  O.SetField:=[mefirst,meSecond];
  I:=GetOrdProp(O,PI);
  Writeln('Value  : ',SetToString(PI,I,True));
  I:=StringToSet(PI,'mefirst');
  SetOrdProp(O,PI,I);
  I:=GetOrdProp(O,PI);
  Writeln('Value  : ',SetToString(PI,I,False));
  I:=StringToSet(PI,'[mesecond,methird]');
  SetOrdProp(O,PI,I);
  I:=GetOrdProp(O,PI);
  Writeln('Value  : ',SetToString(PI,I,True));
  O.Free;
end.
```

78.7.40 SetUnicodeStrProp

Synopsis: Set UnicodeString-valued property

Declaration:
```
procedure SetUnicodeStrProp(Instance: TObject;const PropName: string;
                            const Value: UnicodeString)
procedure SetUnicodeStrProp(Instance: TObject;PropInfo: PPropInfo;
                            const Value: UnicodeString)
```

Visibility: default

Description: `SetUnicodeStrProp` sets the UnicodeString property from `Instance` to `Value`, where the property is identified by the `PropInfo` pointer or the `PropertyName`.

Errors: If no property of the indicated name exists, or it is not of type unicodestring, an exception will occur.

See also: SetStrProp (697), GetUnicodeStrProp (689)

78.7.41 SetVariantProp

Synopsis: Set value of a variant property

Declaration:
```
procedure SetVariantProp(Instance: TObject;const PropName: string;
                         const Value: Variant)
procedure SetVariantProp(Instance: TObject;PropInfo: PPropInfo;
                         const Value: Variant)
```

Visibility: default

Description: Due to missing Variant support, this function is not yet implemented. Provided for Delphi compatibility only.

78.7.42 SetWideStrProp

Synopsis: Set a widestring property

Declaration:
```
procedure SetWideStrProp(Instance: TObject;const PropName: string;
                         const Value: WideString)
procedure SetWideStrProp(Instance: TObject;PropInfo: PPropInfo;
                         const Value: WideString)
```

Visibility: default

Description: `SetWideStrProp` assigns `Value` to the widestring property described by `PropInfo` or with name `Propname` for object `Instance`.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid widestring property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: GetWideStrProp (689), SetStrProp (697), SetOrdProp (695), SetFloatProp (693), SetInt64Prop (694), SetMethodProp (695)

78.7.43 StringToSet

Synopsis: Convert string description to a set.

Declaration:
```
function StringToSet(PropInfo: PPropInfo;const Value: string) : Integer
function StringToSet(TypeInfo: PTypeInfo;const Value: string) : Integer
```

Visibility: default

Description: `StringToSet` converts the string representation of a set in `Value` to a integer representation of the set, using the property information found in `PropInfo`. This property information should point to the property information of a set property. The function returns the integer representation of the set. (i.e, the set value, typecast to an integer)

The string representation can be surrounded with square brackets, and must consist of the names of the elements of the base type of the set. The base type of the set should be an enumerated type. The elements should be separated by commas, and may be surrounded by spaces. each of the names will be fed to the GetEnumValue (676) function.

For an example, see SetToString (697).

Errors: No checking is done to see whether `PropInfo` points to valid property information. If a wrong name is given for an enumerated value, then an `EPropertyError` will be raised.

See also: GetEnumName (674), GetEnumValue (676), SetToString (697)

78.8 TProcedureSignature

```
TProcedureSignature = packed record
  Flags : Byte;
  CC : TCallConv;
  ResultType : PTypeInfo;
  ParamCount : Byte;
  function GetParam(ParamIndex: Integer) : PProcedureParam;
end
```

`TProcedureSignature` describes a procedure/method call signature. It consists of some flags (`Flags`), a calling convention (`CC`), the result type (`ResultType`) if any, and a list of `ParamCount` parameters (of type TProcedureParam (663)).

78.8.1 Method overview

Page	Method	Description
700	GetParam	Get parameter signature

78.8.2 TProcedureSignature.GetParam

Synopsis: Get parameter signature

Declaration: `function GetParam(ParamIndex: Integer) : PProcedureParam`

Visibility: default

Description: `GetParam` can be used to retrieve a pointer to the description of a parameter. The index `ParamIndex` is zero-based.

Errors: In case of an invalid parameter index, `Nil` is returned.

See also: TProcedureParam (663)

78.9 EPropertyConvertError

78.9.1 Description

`EPropertyConvertError` is not used in the Free Pascal implementation of the typinfo unit, but is declared for Delphi compatibility.

78.10 EPropertyError

78.10.1 Description

Exception raised in case of an error in one of the functions.

Chapter 79

Reference for unit 'unicodedata'

79.1 Overview

The fpwidestring (702) unit relies on having relevant unicode collation data linked in the binary. The unicode data is managed using the routines in the unicodedata unit. The FPC project distributes some unicode collation data in `.bco` files which can be loaded using the LoadCollation (711) routines. The `LoadCollation` is the main routine of this unit.

All collation data requires at least the Default Unicode Collation Element Table to be registered (called `DUCET`). The DUCET encoding is provided by the unicodeducet unit, part of the rtl-unicode package.

There are two ways to register collations :

1. **at compile time**: by including the desired collation unit, for example for Russian and Japanese languages to be available you will have to include collation_ru and collation_ja from package "rtl-unicode".
2. **at runtime** using the abovementioned `LoadCollation` function.

The two ways can co-exist: some collations may be compile time included (for example for most used collations) and others can be loaded at runtime in the same application.

The binary collation files are endian sensitive:

- there are files for little endian systems named collation__le.bco (such as collation_ru_le.bco and collation_ja_le.bco)
- there are files for big endian systems named collation__be.bco (such as collation_ru_be.bco and collation_ja_be.bco).

Note that the compile time units collation units (collation_**lang**.pas) include already the unicodeducet.pas (DUCET) unit so it is not necessary to include it manually, contrary to the binary files. So an application that only uses the binary collation files should at least include the unicodeducet unit or manually load the binary collation collation_ducet_le.bco or collation_ducet_be.bco, depending on the endianness of the platform. The LoadCollation (711) call using a directory and the language `ducet` automatically select the correct file.

79.2 Constants, types and variables

79.2.1 Resource strings

```
SCollationNotFound = 'Collation not found : "%s".'
```

Error message when a collation is not found.

79.2.2 Constants

```
ENDIAN_SUFFIX : Array[TEndianKind] of string = ('le', 'be')
```

ENDIAN_SUFFIX contains the suffixes used in LoadCollation (711) when constructing a collation filename for a language.

```
ERROR_INVALID_CODEPOINT_SEQUENCE = 1
```

Error exit code for UnicodeToLower/UnicodeToUpper

```
HIGH_SURROGATE_BEGIN = ($D800)
```

First value for high surrogate values.

```
HIGH_SURROGATE_COUNT = HIGH_SURROGATE_END - HIGH_SURROGATE_BEGIN + 1
```

Number of high surrogate values.

```
HIGH_SURROGATE_END = ($DBFF)
```

Last value for high surrogate values.

```
LOW_SURROGATE_BEGIN = ($DC00)
```

First value for low surrogate values.

```
LOW_SURROGATE_END = ($DFFF)
```

Last value for low surrogate values.

```
MAX_LEGAL_UTF32 = $10FFFF
```

Maximum value of a legally encoded UTF32 value (currently unused)

```
MAX_WORD = (Word)
```

MAX_WORD is the maximum value of a WORD typed value.

```
ROOT_COLLATION_NAME = 'DUCET'
```

Name of the root collation (used if no collation name is given)

```
UCS4_HALF_BASE = ($10000)
```

Offset for UCS4 character encoding

```
UCS4_HALF_MASK = ($3FF)
```

Unused currently

```
UGC_ClosePunctuation = 14
```

Unicode token category: Close punctuation

```
UGC_CombiningMark = 6
```

Unicode token category: Uppercase letter

```
UGC_ConnectPunctuation = 11
```

Unicode token category: Connect punctuation

```
UGC_Control = 25
```

Unicode token category: control token

```
UGC_CurrencySymbol = 19
```

Unicode token category: Currency symbol

```
UGC_DashPunctuation = 12
```

Unicode token category: Dash punctuation

```
UGC_DecimalNumber = 8
```

Unicode token category: Uppercase letter

```
UGC_EnclosingMark = 7
```

Unicode token category: Uppercase letter

```
UGC_FinalPunctuation = 16
```

Unicode token category: Final punctuation

```
UGC_Format = 26
```

Unicode token category: Formatting token

```
UGC_InitialPunctuation = 15
```

Unicode token category: Initial punctuation

```
UGC_LetterNumber = 9
```

Unicode token category: Letter-number

```
UGC_LineSeparator = 23
```

Unicode token category: Line separator

```
UGC_LowercaseLetter = 1
```

Unicode general category: Lowercase letter

```
UGC_MathSymbol = 18
```

Unicode token category: Math symbol

```
UGC_ModifierLetter = 3
```

Unicode general category: modifier letter

```
UGC_ModifierSymbol = 20
```

Unicode token category: modifier symbol

```
UGC_NonSpacingMark = 5
```

Unicode general category: Non-spacing mark

```
UGC_OpenPunctuation = 13
```

Unicode token category: Open puntctuation

```
UGC_OtherLetter = 4
```

Unicode general category: Other letter

```
UGC_OtherNumber = 10
```

Unicode token category: Other number

```
UGC_OtherPunctuation = 17
```

Unicode token category: Other punctuation

```
UGC_OtherSymbol = 21
```

Unicode token category: Other symbol

```
UGC_ParagraphSeparator = 24
```

Unicode token category: Paragraph separator

`UGC_PrivateUse = 28`

Unicode token category: For private use

`UGC_SpaceSeparator = 22`

Unicode token category: Space separator

`UGC_Surrogate = 27`

Unicode token category: Surrogate token

`UGC_TitlecaseLetter = 2`

Unicode general category: Titlecase letter

`UGC_Unassigned = 29`

Unicode token category: As yet unassigned

`UGC_UppercaseLetter = 0`

Unicode general category: Uppercase letter

`ZERO_UINT24 : UInt24 = (byte2: 0; byte1: 0; byte0: 0)`

`TUInt24Rec` value with all bytes zero.

79.2.3 Types

`PUCA_DataBook = ^TUCA_DataBook`

Pointer to `TUCA_DataBook` type.

`PUCA_PropItemContextRec = ^TUCA_PropItemContextRec`

Pointer to `TUCA_PropItemContextRec`

`PUCA_PropItemContextTreeNodeRec = ^TUCA_PropItemContextTreeNodeRec`

Pointer to `TUCA_PropItemContextTreeNodeRec`

`PUCA_PropItemContextTreeRec = ^TUCA_PropItemContextTreeRec`

Pointer to `TUCA_PropItemContextTreeRec`

`PUCA_PropItemRec = ^TUCA_PropItemRec`

Pointer to `TUCA_PropItemRec`

```
PUCA_PropWeights = ^TUCA_PropWeights
```

Pointer to `TUCA_PropWeights`

```
PUC_Prop = ^TUC_Prop
```

Pointer to `TUC_Prop` record

```
PUInt24 = ^UInt24
```

Pointer to `TUInt24Rec`

```
TCollationField = (BackWard,VariableLowLimit,VariableHighLimit)
```

Table 79.1: Enumeration values for type TCollationField

Value	Explanation
BackWard	Backwards encoded
VariableHighLimit	Has upper bound on variable weights
VariableLowLimit	Has lower bound on variable weights

`TCollationField` describes some properties of the collation data items.

```
TCollationFields = Set of TCollationField
```

Set of `TCollationField`

```
TCollationName = string
```

Collation name string type (fixed lenght)

```
TEndianKind = (Little,Big)
```

Table 79.2: Enumeration values for type TEndianKind

Value	Explanation
Big	Big-endian platform
Little	Little-endian platform

`TEndianKind` is an auxiliary enumerated type to enumerate the endianness of platforms.

```
TUCASortKey = Array of TUCASortKeyItem
```

Array of `TUCASortKeyItem`

```
TUCASortKeyItem = Word
```

Alias for WORD

```
TUCA_PropWeights = packed record
  Weights : Array[0..2] of Word;
end
```

`TUC_PropWeights` describes the weights of collation characteristics of a unicode character. It is an internal structure which should not be used directly, the actual structure is subject to change.

```
TUCA_VariableKind = (ucaShifted,ucaNonIgnorable,ucaBlanked,
                     ucaShiftedTrimmed,ucaIgnoreSP)
```

Table 79.3: Enumeration values for type TUCA_VariableKind

Value	Explanation
ucaBlanked	Variable collation elements and any subsequent ignorable collation elements are reset so that all weigh
ucaIgnoreSP	Not implemented (variant of Shifted that reduces the set of variable collation elements to include only
ucaNonIgnorable	Variable collation elements are not reset to be quaternary collation elements
ucaShifted	Variable collation elements are reset to zero at levels one through three
ucaShiftedTrimmed	This option is the same as Shifted, except that all trailing FFFFs are trimmed from the sort key.

Options for weighting data

```
UInt24 = TUInt24Rec
```

Alias for `TUInt24Rec`

79.3 Procedures and functions

79.3.1 CanonicalOrder

Synopsis: Put unicode string in canonical order.

Declaration:
```
procedure CanonicalOrder(var AString: UnicodeString);  Overload
procedure CanonicalOrder(AStr: PUnicodeChar;const ALength: SizeInt)
                        ;  Overload
```

Visibility: default

Description: `CanonicalOrder` transforms a unicode string `AString` (or the alternate form using a null-terminated `AStr` with length `Alength`) so it is in canonical order (as defined by the unicode specification). A string needs to be in canonical order to be able to compare strings. This function is called as part of NormalizeNFD (712).

See also: NormalizeNFD (712)

79.3.2 CompareSortKey

Synopsis: Compare two sort keys.

Declaration:
```
function CompareSortKey(const A: TUCASortKey;const B: TUCASortKey)
                         : Integer;  Overload
function CompareSortKey(const A: TUCASortKey;const B: Array of Word)
                         : Integer;  Overload
```

Visibility: default

Description: `CompareSortKey` compares 2 sort keys `A` and `B`. It returns

- a negative number if `A` comes alphabetically ordered before `B`.
- Zero if `A` is alphabetically identical to `B`.
- A positive number iff `A` is alphabetically ordered after `B`.

Sort keys can be constructed from unicode strings using ComputeSortKey (709).

See also: ComputeSortKey (709)

79.3.3 ComputeSortKey

Synopsis: Compute the sort key for a string

Declaration:
```
function ComputeSortKey(const AString: UnicodeString;
                        const ACollation: PUCA_DataBook) : TUCASortKey
                        ;  Overload
function ComputeSortKey(const AStr: PUnicodeChar;const ALength: SizeInt;
                        const ACollation: PUCA_DataBook) : TUCASortKey
                        ;  Overload
```

Visibility: default

Description: `ComputeSortKey` computes the sort key for a unicode string `AString` (or the alternate form using a null-terminated `AStr` with length `Alength`) using the Unicode Collation Algorithm dat in `ACollation`. This key can then be used in CompareSortKey (709) to compare unicode strings.

See also: CompareSortKey (709)

79.3.4 FindCollation

Synopsis: Find a collation by name

Declaration:
```
function FindCollation(const AName: ansistring) : PUCA_DataBook
                       ;  Overload
function FindCollation(const AIndex: Integer) : PUCA_DataBook;  Overload
```

Visibility: default

Description: `FindCollection` searches a collation with name `AName` or index `AIndex` in the list of known collations and returns a pointer to the collation data. If the requested collation is not known, i.e. the name is not found, or the index is out of range, then `Nil` is returned. The valid index range is 0 to `GetCollationCount-1`.

See also: GetCollationCount (710)

79.3.5 FreeCollation

Synopsis: Free collation data.

Declaration:
```
procedure FreeCollation(AItem: PUCA_DataBook)
```

Visibility: default

Description: `FreeCollation` removes all structures in the collation data from memory. It will not do anything when the header field `Dynamic` is `false`. (collations loaded and registered using LoadCollation (711) are always dynamic and must be freed).

See also: LoadCollation (711)

79.3.6 FromUCS4

Synopsis: Convert UCS4 to UNicodeChar

Declaration:
```
procedure FromUCS4(const AValue: UCS4Char;var AHighS: UnicodeChar;
                   var ALowS: UnicodeChar)
```

Visibility: default

Description: `FromUCS4` converts the UCS4 encoded unicode character `AValue` to a set of unicode (surrogate pair) characters encoded in UTF16: `AHighS`, `ALowS`.

See also: ToUCS4 (713), UnicodeIsHighSurrogate (713), UnicodeIsLowSurrogate (713), UnicodeIsSurrogatePair (714)

79.3.7 GetCollationCount

Synopsis: Return the number of known collations.

Declaration:
```
function GetCollationCount : Integer
```

Visibility: default

Description: `GetCollationCount` returns the number of registered collations. The collation data can be retrieved by index using the FindCollation (709). The maximum index is `GetCollationCount-1`.

See also: FindCollation (709)

79.3.8 GetProps

Synopsis: Get unicode character data

Declaration:
```
function GetProps(const ACodePoint: Word) : PUC_Prop;  Overload
function GetProps(const AHighS: UnicodeChar;const ALowS: UnicodeChar)
                  : PUC_Prop;  Overload
function GetProps(const ACodePoint: Cardinal) : PUC_Prop;  Overload
```

Visibility: default

Description: `GetProps` returns a pointer to a general unicode character property data structure. The character can be specified using a word or cardinal sized codepoint (`ACodePoint`), or using a UTF16 encoded surrogate pair (`AHighS`,`ALowS`).

The returned pointer must not be freed, it points to part of a static structure.

See also: GetPropUCA (711)

79.3.9 GetPropUCA

Synopsis: Get unicode collation algorithm properties for a unicode character

Declaration:
```
function GetPropUCA(const AHighS: UnicodeChar;const ALowS: UnicodeChar;
                    const ABook: PUCA_DataBook) : PUCA_PropItemRec
                    ;  Overload
function GetPropUCA(const AChar: UnicodeChar;const ABook: PUCA_DataBook)
                     : PUCA_PropItemRec;  Overload
```

Visibility: default

Description: `GetPropUCA` returns UCA data for the unicode character (`AChar`) or surrogate pair `AHighS`, `ALowS`) in the collation data book `ABook`. If no data is available, `Nil` is returned.

Errors: if an invalid `ABook` is specified, an access violation may occur.

See also: GetProps (710)

79.3.10 IncrementalCompareString

Synopsis: Compare 2 strings using the specified collation

Declaration:
```
function IncrementalCompareString(const AStrA: PUnicodeChar;
                                  const ALengthA: SizeInt;
                                  const AStrB: PUnicodeChar;
                                  const ALengthB: SizeInt;
                                  const ACollation: PUCA_DataBook)
                                   : Integer;  Overload
function IncrementalCompareString(const AStrA: UnicodeString;
                                  const AStrB: UnicodeString;
                                  const ACollation: PUCA_DataBook)
                                   : Integer;  Overload
```

Visibility: default

Description: `IncrementalCompareString` creates 2 compare keys from the strings `AStrA` and `AStrB` using collation data in `ACollation`. The keys are computed only to the point where the two strings differ. This means the keys cannot be reused for other comparisons if the strings differ. The two strings can be specified as a unicode string or as a pointer to a null-terminated character array with a length (`ALengthA` and `ALengthB`). It returns then the result of CompareSortKey (709).

Errors: None.

See also: ComputeSortKey (709), CompareSortKey (709)

79.3.11 LoadCollation

Synopsis: Load a binary collation data file from file

Declaration:
```
function LoadCollation(const AData: Pointer;const ADataLength: Integer)
                        : PUCA_DataBook;  Overload
function LoadCollation(const AFileName: string) : PUCA_DataBook
                       ;  Overload
function LoadCollation(const ADirectory: string;const ALanguage: string)
                        : PUCA_DataBook;  Overload
```

Visibility: default

Description: `LoadCollation` loads collation data from file `AFileName`, or from a memory block `AData` with length `ADataLength`. If successful, it returns a pointer which can be used to register the collation using RegisterCollation (712). if there is a problem with the data, `Nil` is returned.

The filename can also be specified as a `Directory` and language name `ALanguage`. The latter is prepended with `collation_` and appended with the native endianness of the current platform, and has extension `.bco`

Errors: If the file containing data does not exist or has a size which is less than the encoded header size, `Nil` is returned.

See also: RegisterCollation (712)

79.3.12 NormalizeNFD

Synopsis: Perform unicode normalization D on a string

Declaration:
```
function NormalizeNFD(const AString: UnicodeString) : UnicodeString
                         ;  Overload
function NormalizeNFD(const AStr: PUnicodeChar;ALength: SizeInt)
                          : UnicodeString;  Overload
```

Visibility: default

Description: `NormalizeNFD` normalizes the string `AString` (or the alternate form using a null-terminated `AStr` with length `Alength`) to Unicode Normalization Form D. The resulting string can be used to determine equivalence of unicode strings.

See also: CanonicalOrder (708)

79.3.13 PrepareCollation

Synopsis: Prepare a collation for use in the list

Declaration:
```
procedure PrepareCollation(ACollation: PUCA_DataBook;
                           const ABaseName: ansistring;
                           const AChangedFields: TCollationFields)
```

Visibility: default

Description: `PrepareCollation` will link collation definition `ACollation` to the base collection with name `ABaseName` (if empty, it defaults to the root collation). It will also initialize some fields in the definition, copying them from the base collation, but excludes the fields enumerated in `AChangedFields`.

It should normally not be needed to call this function, it is called as part of LoadCollation (711).

See also: LoadCollation (711)

79.3.14 RegisterCollation

Synopsis: Register a new collation

Declaration:
```
function RegisterCollation(const ACollation: PUCA_DataBook) : Boolean
                           ;  Overload
function RegisterCollation(const ADirectory: string;
                           const ALanguage: string) : Boolean;  Overload
```

Visibility: default

Description: `RegisterCollation` registers a new collation `ACollation` in the list of known collations. The collation data can be specified directly (`ACollation`) or a name of a language (`ALanguage`) in a directory (`ADirectory`). The latter option will load the binary encodeded collation from file using LoadCollation (711).

If the collation is loaded correctly, `True` is returned, otherwise `False` is returned (for instance when a collation with the same name is already loaded).

See also: UnRegisterCollation (715), UnRegisterCollations (715), LoadCollation (711)

79.3.15 ToUCS4

Synopsis: Encode unicode UTF16 surrogate pair to UCS4 character

Declaration:
```
function ToUCS4(const AHighS: UnicodeChar;const ALowS: UnicodeChar)
                      : UCS4Char
```

Visibility: default

Description: `ToUCS4` converts set of unicode (surrogate pair) characters encoded in UTF16: `AHighS`, `ALowS` to a UCS4 encoded unicode character.

See also: FromUCS4 (710), UnicodeIsHighSurrogate (713), UnicodeIsLowSurrogate (713), UnicodeIsSurrogatePair (714)

79.3.16 UnicodeIsHighSurrogate

Synopsis: Check if a UTF16 character is the high character in a surrogate pair.

Declaration:
```
function UnicodeIsHighSurrogate(const AValue: UnicodeChar) : Boolean
```

Visibility: default

Description: `UnicodeIsHighSurrogate` checks whether `AValue` is a valid high character of a surrogate pair, and returns `True` if this is the case. It does this by checking whether the values are within the bounds for high characters in surrogate pairs.

See also: FromUCS4 (710), ToUCS4 (713), UnicodeIsHighSurrogate (713), UnicodeIsLowSurrogate (713), UnicodeIsSurrogatePair (714)

79.3.17 UnicodeIsLowSurrogate

Synopsis: Check if a UTF16 character is the low character in a surrogate pair.

Declaration:
```
function UnicodeIsLowSurrogate(const AValue: UnicodeChar) : Boolean
```

Visibility: default

Description: `UnicodeIsHighSurrogate` checks whether `AValue` is a valid high character of a surrogate pair, and returns `True` if this is the case. It does this by checking whether the values are within the bounds for high characters in surrogate pairs.

See also: FromUCS4 (710), ToUCS4 (713), UnicodeIsHighSurrogate (713), UnicodeIsSurrogatePair (714)

79.3.18 UnicodeIsSurrogatePair

Synopsis: Check if a pair of UTF16 encoded characters is a valid surrogate pair.

Declaration:
```
function UnicodeIsSurrogatePair(const AHighSurrogate: UnicodeChar;
                                const ALowSurrogate: UnicodeChar)
                                 : Boolean
```

Visibility: default

Description: `UnicodeIsSurrogatePair` checks whether `AHighSurrogate`,`ALowSurrogate` constitute a valid surrogate pair, and returns `True` if this is the case. It does this by checking whether the values are within the bounds for high and low surrogate pairs.

See also: FromUCS4 (710), ToUCS4 (713), UnicodeIsHighSurrogate (713), UnicodeIsLowSurrogate (713)

79.3.19 UnicodeToLower

Synopsis: Transform unicode string to lowercase

Declaration:
```
function UnicodeToLower(const AString: UnicodeString;
                        const AIgnoreInvalidSequence: Boolean;
                        out AResultString: UnicodeString) : Integer
```

Visibility: default

Description: `UnicodeToLower` transforms a UTF16 unicode string `AString` to its lowercase equivalent and returns this in `AResultString`. If the transformation was succesful, then the function returns `0`. A nonzero return value means an error occurred. `AResultString` will remain untouched in that case.

If a character in `AString` cannot be found in the unicode data tables, an error will be reported, unless `AIgnoreInvalidSequence` is set to `True`, in which case the character will be copied as-is to the output.

Unicode collation data can be loaded using RegisterCollation (712) or LoadCollation (711)

Errors: On error, a nonzero value will be returned.

See also: UnicodeToUpper (714), RegisterCollation (712), LoadCollation (711)

79.3.20 UnicodeToUpper

Synopsis: Transform unicode string to uppercase

Declaration:
```
function UnicodeToUpper(const AString: UnicodeString;
                        const AIgnoreInvalidSequence: Boolean;
                        out AResultString: UnicodeString) : Integer
```

Visibility: default

Description: `UnicodeToUpper` transforms a UTF16 unicode string `AString` to its uppercase equivalent and returns this in `AResultString`. If the transformation was succesful, then the function returns `0`. A nonzero return value means an error occurred. `AResultString` will remain untouched in that case.

If a character in `AString` cannot be found in the unicode data tables, an error will be reported, unless `AIgnoreInvalidSequence` is set to `True`, in which case the character will be copied as-is to the output.

Unicode collation data can be loaded using RegisterCollation (712) or LoadCollation (711)

Errors: On error, a nonzero value will be returned.

See also: UnicodeToLower (714), RegisterCollation (712), LoadCollation (711)

79.3.21 UnregisterCollation

Synopsis: Remove a collation from the list of known collections.

Declaration: `function UnregisterCollation(const AName: ansistring) : Boolean`

Visibility: default

Description: `UnRegisterCollation` removes a collation `AName` from the list of known collations. It returns `True` if the collation was found and succesfully removed.

Errors: If the collation was not found, `False` is returned.

See also: RegisterCollation (712), UnRegisterCollations (715), LoadCollation (711)

79.3.22 UnregisterCollations

Synopsis: Unregister all collations.

Declaration: `procedure UnregisterCollations(const AFreeDynamicCollations: Boolean)`

Visibility: default

Description: `UnregisterCollations` unregisters all known collations. If `AFreeDynamicCollations` is `True`, then dynamic collations will be removed from memory using FreeCollation (710). This must normally be set to `true`.

See also: RegisterCollation (712), UnRegisterCollation (715), LoadCollation (711), FreeCollation (710)

79.4 TUCA_DataBook

```
TUCA_DataBook = record
public
  Base : PUCA_DataBook;
  Version : TCollationName;
  CollationName : TCollationName;
  VariableWeight : TUCA_VariableKind;
  Backwards : Array[0..3] of Boolean;
  BMP_Table1 : PByte;
  BMP_Table2 : PUInt24;
  OBMP_Table1 : PWord;
  OBMP_Table2 : PUInt24;
  PropCount : Integer;
  Props : PUCA_PropItemRec;
  VariableLowLimit : Word;
  VariableHighLimit : Word;
  Dynamic : Boolean;
  function IsVariable(const AWeight: PUCA_PropWeights) : Boolean;
end
```

TUCA_DataBook describes a Unicode Collation Alghorithm data set. data sets can be registered using the RegisterCollation (712) function or loaded from file using LoadCollation (711). A collation data book must be specified when comparing unicode strings.

79.4.1 Method overview

Page	Method	Description
716	IsVariable	Check if a weight is a variable weight

79.4.2 TUCA_DataBook.IsVariable

Synopsis: Check if a weight is a variable weight

Declaration: `function IsVariable(const AWeight: PUCA_PropWeights) : Boolean`

Visibility: public

Description: IsVariable checks whether AWeight is between the VariableLowLimit and VariableHighLimit limits.

79.5 TUCA_PropItemContextRec

```
TUCA_PropItemContextRec = packed record
public
  CodePointCount : Byte;
  WeightCount : Byte;
  function GetCodePoints : PUInt24;
  function GetWeights : PUCA_PropWeights;
end
```

This is an internal structure which should not be used directly, the actual structure is subject to change.

79.5.1 Method overview

Page	Method	Description
716	GetCodePoints	get the address of actual code points.
716	GetWeights	Get the Address of actual weights.

79.5.2 TUCA_PropItemContextRec.GetCodePoints

Synopsis: get the address of actual code points.

Declaration: `function GetCodePoints : PUInt24`

Visibility: public

79.5.3 TUCA_PropItemContextRec.GetWeights

Synopsis: Get the Address of actual weights.

Declaration: `function GetWeights : PUCA_PropWeights`

Visibility: public

79.6 TUCA_PropItemContextTreeNodeRec

```
TUCA_PropItemContextTreeNodeRec = packed record
public
  Left : Word;
  Right : Word;
  Data : TUCA_PropItemContextRec;
  function GetLeftNode : PUCA_PropItemContextTreeNodeRec;
  function GetRightNode : PUCA_PropItemContextTreeNodeRec;
end
```

This is an internal structure for the tree which should not be used directly, the actual structure is subject to change.

79.6.1 Method overview

Page	Method	Description
717	GetLeftNode	Access to left tree node data
717	GetRightNode	Access to right tree node data

79.6.2 TUCA_PropItemContextTreeNodeRec.GetLeftNode

Synopsis: Access to left tree node data

Declaration: `function GetLeftNode : PUCA_PropItemContextTreeNodeRec`

Visibility: public

79.6.3 TUCA_PropItemContextTreeNodeRec.GetRightNode

Synopsis: Access to right tree node data

Declaration: `function GetRightNode : PUCA_PropItemContextTreeNodeRec`

Visibility: public

79.7 TUCA_PropItemContextTreeRec

```
TUCA_PropItemContextTreeRec = packed record
public
  Size : UInt24;
  function GetData : PUCA_PropItemContextTreeNodeRec;
  property Data : PUCA_PropItemContextTreeNodeRec;
  function Find(const AChars: PUInt24;const ACharCount: Integer;
                out ANode: PUCA_PropItemContextTreeNodeRec) : Boolean;
end
```

This is an internal tree structure for storing unicode collation data which should not be used directly, the actual structure is subject to change.

79.7.1 Method overview

Page	Method	Description
718	Find	Find data for encoded character
718	GetData	Access to tree data (getter for Data property)

79.7.2 Property overview

Page	Properties	Access	Description
718	Data	r	Read-only access to tree data

79.7.3 TUCA_PropItemContextTreeRec.GetData

Synopsis: Access to tree data (getter for Data property)

Declaration:
```
function GetData : PUCA_PropItemContextTreeNodeRec
```

Visibility: public

79.7.4 TUCA_PropItemContextTreeRec.Find

Synopsis: Find data for encoded character

Declaration:
```
function Find(const AChars: PUInt24;const ACharCount: Integer;
                  out ANode: PUCA_PropItemContextTreeNodeRec) : Boolean
```

Visibility: public

Description: `Find` searches the tree for the collation data for the character encoded in `AChars` (`ACharCount`). It returns true if the data was found, false if not. A pointer to the collation data is returned in `ANode`.

79.7.5 TUCA_PropItemContextTreeRec.Data

Synopsis: Read-only access to tree data

Declaration:
```
Property Data : PUCA_PropItemContextTreeNodeRec
```

Visibility: public

Access: Read

79.8 TUCA_PropItemRec

```
TUCA_PropItemRec = packed record
private
  FLAG_VALID = 0;
  FLAG_CODEPOINT = 1;
  FLAG_CONTEXTUAL = 2;
  FLAG_DELETION = 3;
  FLAG_COMPRESS_WEIGHT_1 = 6;
```

```
  FLAG_COMPRESS_WEIGHT_2 = 7;
  function GetCodePoint : UInt24;
public
  WeightLength : Byte;
  ChildCount : Byte;
  Size : Word;
  Flags : Byte;
  function HasCodePoint : Boolean;
  property CodePoint : UInt24;
  function IsValid : Boolean;
  procedure GetWeightArray(ADest: PUCA_PropWeights);
  function GetSelfOnlySize : Cardinal;
  function GetContextual : Boolean;
  property Contextual : Boolean;
  function GetContext : PUCA_PropItemContextTreeRec;
  function IsDeleted : Boolean;
  function IsWeightCompress_1 : Boolean;
  function IsWeightCompress_2 : Boolean;
end
```

`TUCA_PropItemRec` encodes 1 entry from the Unicode Collation data in an encoded form.

79.8.1 Method overview

Page	Method	Description
720	GetContext	Access to context data
720	GetContextual	Check if the contextual bit is set in the flags (getter for Contextual)
720	GetSelfOnlySize	Size of this item data (in bytes).
720	GetWeightArray	Return an array of weights
719	HasCodePoint	Check flags whether a codepoint is present.
720	IsDeleted	Check flags if deleted bit is set
720	IsValid	Check flags for validity
721	IsWeightCompress_1	Check whether weight compression flag 1 is set
721	IsWeightCompress_2	Check whether weight compression flag 2 is set

79.8.2 Property overview

Page	Properties	Access	Description
721	CodePoint	r	Get the codepoint
721	Contextual	r	Check if the contextual bit is set in the flags

79.8.3 TUCA_PropItemRec.HasCodePoint

Synopsis: Check flags whether a codepoint is present.

Declaration: `function HasCodePoint : Boolean`

Visibility: public

79.8.4 TUCA_PropItemRec.IsValid

Synopsis: Check flags for validity

Declaration: `function IsValid : Boolean`

Visibility: public

79.8.5 TUCA_PropItemRec.GetWeightArray

Synopsis: Return an array of weights

Declaration: `procedure GetWeightArray(ADest: PUCA_PropWeights)`

Visibility: public

Description: `GetWeightArray` returns an array with the weights in `ADest`. `ADest` must point to enough room for `WeightLength` weights.

79.8.6 TUCA_PropItemRec.GetSelfOnlySize

Synopsis: Size of this item data (in bytes).

Declaration: `function GetSelfOnlySize : Cardinal`

Visibility: public

Description: Return the size of the item's data and properties.

79.8.7 TUCA_PropItemRec.GetContextual

Synopsis: Check if the contextual bit is set in the flags (getter for Contextual)

Declaration: `function GetContextual : Boolean`

Visibility: public

79.8.8 TUCA_PropItemRec.GetContext

Synopsis: Access to context data

Declaration: `function GetContext : PUCA_PropItemContextTreeRec`

Visibility: public

Description: `GetContext` returns a pointer to the context data. It is `Nil` if the context flag is not set.

79.8.9 TUCA_PropItemRec.IsDeleted

Synopsis: Check flags if deleted bit is set

Declaration: `function IsDeleted : Boolean`

Visibility: public

79.8.10 TUCA_PropItemRec.IsWeightCompress_1

Synopsis: Check whether weight compression flag 1 is set

Declaration: `function IsWeightCompress_1 : Boolean`

Visibility: public

79.8.11 TUCA_PropItemRec.IsWeightCompress_2

Synopsis: Check whether weight compression flag 2 is set

Declaration: `function IsWeightCompress_2 : Boolean`

Visibility: public

79.8.12 TUCA_PropItemRec.CodePoint

Synopsis: Get the codepoint

Declaration: `Property CodePoint : UInt24`

Visibility: public

Access: Read

Description: Access to codepoint if `HasCodePoint` returns true. If `HasCodePoint` returns false, an exception will be raised.

79.8.13 TUCA_PropItemRec.Contextual

Synopsis: Check if the contextual bit is set in the flags

Declaration: `Property Contextual : Boolean`

Visibility: public

Access: Read

79.9 TUC_Prop

```
TUC_Prop = packed record
private
  function GetCategory : Byte;
  procedure SetCategory(AValue: Byte);
  function GetWhiteSpace : Boolean;
  procedure SetWhiteSpace(AValue: Boolean);
  function GetHangulSyllable : Boolean;
  procedure SetHangulSyllable(AValue: Boolean);
  function GetNumericValue : Double;
public
  CategoryData : Byte;
  CCC : Byte;
  NumericIndex : Byte;
  SimpleUpperCase : UInt24;
```

```
  SimpleLowerCase : UInt24;
  DecompositionID : SmallInt;
  property Category : Byte;
  property WhiteSpace : Boolean;
  property HangulSyllable : Boolean;
  property NumericValue : Double;
end
```

`TUC_Prop` describes the collation characteristics of a unicode character. It is an internal structure which should not be used directly, the actual structure is subject to change.

79.9.1 Property overview

Page	Properties	Access	Description
722	Category	rw	Get the category
722	HangulSyllable	rw	Is the character a hangul syllable
723	NumericValue	r	Numeric value
722	WhiteSpace	rw	Is the character considered whitespace

79.9.2 TUC_Prop.Category

Synopsis: Get the category

Declaration: `Property Category : Byte`

Visibility: public

Access: Read,Write

Description: `CategoryData` provides access to the category part of CategoryData (encoded).

79.9.3 TUC_Prop.WhiteSpace

Synopsis: Is the character considered whitespace

Declaration: `Property WhiteSpace : Boolean`

Visibility: public

Access: Read,Write

Description: `Whitespace` provides easy access to the Whitespace part of CategoryData (encoded).

79.9.4 TUC_Prop.HangulSyllable

Synopsis: Is the character a hangul syllable

Declaration: `Property HangulSyllable : Boolean`

Visibility: public

Access: Read,Write

Description: `Whitespace` provides easy access to the HangulSyllable part of CategoryData (encoded).

79.9.5 TUC_Prop.NumericValue

Synopsis: Numeric value

Declaration: `Property NumericValue : Double`

Visibility: public

Access: Read

Description: `NumericValue` uses numericalindex to get the numerical value.

79.10 TUInt24Rec

```
TUInt24Rec = packed record
public
  byte2 : Byte;
  byte1 : Byte;
  byte0 : Byte;
  TUInt24Rec.class operator implicit(a: TUInt24Rec) : Cardinal;
  TUInt24Rec.class operator implicit(a: TUInt24Rec) : LongInt;
  TUInt24Rec.class operator implicit(a: TUInt24Rec) : Word;
  TUInt24Rec.class operator implicit(a: TUInt24Rec) : Byte;
  TUInt24Rec.class operator implicit(a: Cardinal) : TUInt24Rec;
  TUInt24Rec.class operator equal(a: TUInt24Rec;b: TUInt24Rec) : Boolean;
  TUInt24Rec.class operator equal(a: TUInt24Rec;b: Cardinal) : Boolean;
  TUInt24Rec.class operator equal(a: Cardinal;b: TUInt24Rec) : Boolean;
  TUInt24Rec.class operator equal(a: TUInt24Rec;b: LongInt) : Boolean;
  TUInt24Rec.class operator equal(a: LongInt;b: TUInt24Rec) : Boolean;
  TUInt24Rec.class operator equal(a: TUInt24Rec;b: Word) : Boolean;
  TUInt24Rec.class operator equal(a: Word;b: TUInt24Rec) : Boolean;
  TUInt24Rec.class operator equal(a: TUInt24Rec;b: Byte) : Boolean;
  TUInt24Rec.class operator equal(a: Byte;b: TUInt24Rec) : Boolean;
  TUInt24Rec.class operator notequal(a: TUInt24Rec;b: TUInt24Rec)
                                    : Boolean;
  TUInt24Rec.class operator notequal(a: TUInt24Rec;b: Cardinal) : Boolean;
  TUInt24Rec.class operator notequal(a: Cardinal;b: TUInt24Rec) : Boolean;
  TUInt24Rec.class operator greaterthan(a: TUInt24Rec;b: TUInt24Rec)
                                       : Boolean;
  TUInt24Rec.class operator greaterthan(a: TUInt24Rec;b: Cardinal)
                                       : Boolean;
  TUInt24Rec.class operator greaterthan(a: Cardinal;b: TUInt24Rec)
                                       : Boolean;
  TUInt24Rec.class operator greaterthanorequal(a: TUInt24Rec;
                                          b: TUInt24Rec) : Boolean;
  TUInt24Rec.class operator greaterthanorequal(a: TUInt24Rec;b: Cardinal)
                                           : Boolean;
  TUInt24Rec.class operator greaterthanorequal(a: Cardinal;b: TUInt24Rec)
                                           : Boolean;
  TUInt24Rec.class operator lessthan(a: TUInt24Rec;b: TUInt24Rec)
                                    : Boolean;
  TUInt24Rec.class operator lessthan(a: TUInt24Rec;b: Cardinal) : Boolean;
  TUInt24Rec.class operator lessthan(a: Cardinal;b: TUInt24Rec) : Boolean;
  TUInt24Rec.class operator lessthanorequal(a: TUInt24Rec;b: TUInt24Rec)
```

```
                                                : Boolean;
  TUInt24Rec.class operator lessthanorequal(a: TUInt24Rec;b: Cardinal)
                                                : Boolean;
  TUInt24Rec.class operator lessthanorequal(a: Cardinal;b: TUInt24Rec)
                                                : Boolean;
end
```

Unicode data exists mostly of 24-bit data (3 bytes). This type is meant to deal efficiently with this data. it has members to split out the data in bytes, and functions to query the various properties stored in the data.

79.10.1 Method overview

Page	Method	Description
729	equal(Byte,TUInt24Rec):Boolean	Check whether a TUInt24Rec value equals a byte value.
728	equal(Cardinal,TUInt24Rec):Boolean	Check whether a TUInt24Rec value equals a cardinal value.
729	equal(LongInt,TUInt24Rec):Boolean	Check whether a TUInt24Rec value equals a longint value.
729	equal(TUInt24Rec,Byte):Boolean	Check whether a TUInt24Rec value equals a byte value.
728	equal(TUInt24Rec,Cardinal):Boolean	Check if cardinal and TUInt24Rec are equal
728	equal(TUInt24Rec,LongInt):Boolean	Check whether a TUInt24Rec value equals a longint value.
728	equal(TUInt24Rec,TUInt24Rec):Boolean	Determine equality of 2 TUInt24Rec records
729	equal(TUInt24Rec,Word):Boolean	Check whether a TUInt24Rec value equals a word value.
729	equal(Word,TUInt24Rec):Boolean	Check whether a TUInt24Rec value equals a word value.
730	greaterthan(Cardinal,TUInt24Rec):Boolean	Check whether a cardinal value is greater than a TUInt24Rec value
730	greaterthan(TUInt24Rec,Cardinal):Boolean	Check whether a TUInt24Rec value is greater than a cardinal value
730	greaterthan(TUInt24Rec,TUInt24Rec):Boolean	Check whether a TUInt24Rec value is greater than another TUInt24Rec value
731	greaterthanorequal(Cardinal,TUInt24Rec):Boolean	Check whether a cardinal value is greater than or equal to a TUInt24Rec value
731	greaterthanorequal(TUInt24Rec,Cardinal):Boolean	Check whether a TUInt24Rec value is greater than or equal to a cardinal value
731	greaterthanorequal(TUInt24Rec,TUInt24Rec):Boolean	Check whether a TUInt24Rec value is greater than or equal to a cardinal value
728	implicit(Cardinal):TUInt24Rec	Assign TUInt24Recfrom Cardinal
727	implicit(TUInt24Rec):Byte	Assign TUInt24Rec to byte
727	implicit(TUInt24Rec):Cardinal	Assign TUInt24Rec to cardinal
727	implicit(TUInt24Rec):LongInt	Assign TUInt24Rec to longint
727	implicit(TUInt24Rec):Word	Assign TUInt24Rec to word
732	lessthan(Cardinal,TUInt24Rec):Boolean	Check whether a cardinal value is less than a TUInt24Rec value
731	lessthan(TUInt24Rec,Cardinal):Boolean	Check whether a TUInt24Rec value is

79.10.2 TUInt24Rec.implicit(TUInt24Rec):Cardinal

Synopsis: Assign `TUInt24Rec` to cardinal

Declaration: `TUInt24Rec.class operator implicit(a: TUInt24Rec) : Cardinal`

Visibility: public

Description: Assign to cardinal, byte0 to MSB and so on.

See also: TUInt24Rec.implicit(TUInt24Rec):LongInt (727), TUInt24Rec.implicit(TUInt24Rec):Word (727), TUInt24Rec.implicit(TUInt24Rec):Byte (727), TUInt24Rec.implicit(Cardinal):TUInt24Rec (728), TUInt24Rec.implicit(Longint):TUInt24Rec (724), TUInt24Rec.implicit(Word):TUInt24Rec (724), TUInt24Rec.implicit(Byte):TUInt24Rec (724)

79.10.3 TUInt24Rec.implicit(TUInt24Rec):LongInt

Synopsis: Assign `TUInt24Rec` to longint

Declaration: `TUInt24Rec.class operator implicit(a: TUInt24Rec) : LongInt`

Visibility: public

Description: Assign to cardinal, byte0 to MSB and so on.

See also: TUInt24Rec.implicit(TUInt24Rec):Cardinal (727), TUInt24Rec.implicit(TUInt24Rec):Word (727), TUInt24Rec.implicit(TUInt24Rec):Byte (727)

79.10.4 TUInt24Rec.implicit(TUInt24Rec):Word

Synopsis: Assign `TUInt24Rec` to word

Declaration: `TUInt24Rec.class operator implicit(a: TUInt24Rec) : Word`

Visibility: public

Description: Assign to word, byte0 to MSB, byte1 to MSB.

Errors: If the value is too big (>$FFFF) to be assigned, an overflow error occurs.

See also: TUInt24Rec.implicit(TUInt24Rec):Cardinal (727), TUInt24Rec.implicit(TUInt24Rec):Longint (727), TUInt24Rec.implicit(TUInt24Rec):Byte (727), TUInt24Rec.implicit(Cardinal):TUInt24Rec (728), TUInt24Rec.implicit(Longint):TUInt24Rec (724), TUInt24Rec.implicit(Word):TUInt24Rec (724), TUInt24Rec.implicit(Byte):TUInt24Rec (724)

79.10.5 TUInt24Rec.implicit(TUInt24Rec):Byte

Synopsis: Assign `TUInt24Rec` to byte

Declaration: `TUInt24Rec.class operator implicit(a: TUInt24Rec) : Byte`

Visibility: public

Description: Assign to byte, byte0 is assigned.

Errors: If the value is too big (>$FF) to be assigned, an overflow error occurs.

See also: TUInt24Rec.implicit(TUInt24Rec):Cardinal (727), TUInt24Rec.implicit(TUInt24Rec):Longint (727), TUInt24Rec.implicit(TUInt24Rec):Word (727), TUInt24Rec.implicit(Cardinal):TUInt24Rec (728), TUInt24Rec.implicit(Longint):TUInt24Rec (724), TUInt24Rec.implicit(Word):TUInt24Rec (724), TUInt24Rec.implicit(Byte):TUInt24Rec (724)

79.10.6 TUInt24Rec.implicit(Cardinal):TUInt24Rec

Synopsis: Assign `TUInt24Rec`from Cardinal

Declaration: `TUInt24Rec.class operator implicit(a: Cardinal) : TUInt24Rec`

Visibility: public

Description: Assign from cardinal, byte0 to MSB.

Errors: If the value is too big (>$FFFFFF) to be assigned, an overflow error occurs.

See also: TUInt24Rec.implicit(TUInt24Rec):Longint (727), TUInt24Rec.implicit(TUInt24Rec):Word (727), TUInt24Rec.implicit(TUInt24Rec):Byte (727), TUInt24Rec.implicit(Longint):TUInt24Rec (724), TUInt24Rec.implicit(Word):TUInt24Rec (724), TUInt24Rec.implicit(Byte):TUInt24Rec (724)

79.10.7 TUInt24Rec.equal(TUInt24Rec,TUInt24Rec):Boolean

Synopsis: Determine equality of 2 `TUInt24Rec` records

Declaration: `TUInt24Rec.class operator equal(a: TUInt24Rec;b: TUInt24Rec) : Boolean`

Visibility: public

Description: The 2 records are considered equal if the 3 bytes are equal.

79.10.8 TUInt24Rec.equal(TUInt24Rec,Cardinal):Boolean

Synopsis: Check if cardinal and `TUInt24Rec` are equal

Declaration: `TUInt24Rec.class operator equal(a: TUInt24Rec;b: Cardinal) : Boolean`

Visibility: public

Description: The cardinal `b` is considered equal to `a` if the fourth byte (LSB) is zero, and the first three bytes equal `byte0`, `byte1`, and `byte2`.

79.10.9 TUInt24Rec.equal(Cardinal,TUInt24Rec):Boolean

Synopsis: Check whether a `TUInt24Rec` value equals a cardinal value.

Declaration: `TUInt24Rec.class operator equal(a: Cardinal;b: TUInt24Rec) : Boolean`

Visibility: public

Description: The cardinal `a` is considered equal to `b` if the fourth byte (LSB) is zero, and the first three bytes equal `byte0`, `byte1`, and `byte2`.

79.10.10 TUInt24Rec.equal(TUInt24Rec,LongInt):Boolean

Synopsis: Check whether a `TUInt24Rec` value equals a longint value.

Declaration: `TUInt24Rec.class operator equal(a: TUInt24Rec;b: LongInt) : Boolean`

Visibility: public

Description: The longint `b` is considered equal to `a` if the fourth byte (LSB) is zero, and the first three bytes equal `byte0`, `byte1`, and `byte2`.

79.10.11 TUInt24Rec.equal(LongInt,TUInt24Rec):Boolean

Synopsis: Check whether a `TUInt24Rec` value equals a longint value.

Declaration:
```
TUInt24Rec.class operator equal(a: LongInt;b: TUInt24Rec) : Boolean
```

Visibility: public

Description: The longint `a` is considered equal to `b` if the fourth byte (LSB) is zero, and the first three bytes equal `byte0`, `byte1`, and `byte2`.

79.10.12 TUInt24Rec.equal(TUInt24Rec,Word):Boolean

Synopsis: Check whether a `TUInt24Rec` value equals a word value.

Declaration:
```
TUInt24Rec.class operator equal(a: TUInt24Rec;b: Word) : Boolean
```

Visibility: public

Description: The word `b` is considered equal to `a` if its 2 bytes equal `byte0`, `byte1` and `byte2` is zero.

79.10.13 TUInt24Rec.equal(Word,TUInt24Rec):Boolean

Synopsis: Check whether a `TUInt24Rec` value equals a word value.

Declaration:
```
TUInt24Rec.class operator equal(a: Word;b: TUInt24Rec) : Boolean
```

Visibility: public

Description: The word `a` is considered equal to `b` if its 2 bytes equal `byte0`, `byte1` and `byte2` is zero.

79.10.14 TUInt24Rec.equal(TUInt24Rec,Byte):Boolean

Synopsis: Check whether a `TUInt24Rec` value equals a byte value.

Declaration:
```
TUInt24Rec.class operator equal(a: TUInt24Rec;b: Byte) : Boolean
```

Visibility: public

Description: The byte `b` is considered equal to `a` if it equals `byte0` and `byte1` and `byte1` are zero.

79.10.15 TUInt24Rec.equal(Byte,TUInt24Rec):Boolean

Synopsis: Check whether a `TUInt24Rec` value equals a byte value.

Declaration:
```
TUInt24Rec.class operator equal(a: Byte;b: TUInt24Rec) : Boolean
```

Visibility: public

Description: The byte `b` is considered equal to `a` if it equals `byte0` and `byte1` and `byte1` are zero.

79.10.16 TUInt24Rec.notequal(TUInt24Rec,TUInt24Rec):Boolean

Synopsis: Check whether 2 `TUInt24Rec` values differ.

Declaration:
```
TUInt24Rec.class operator notequal(a: TUInt24Rec;b: TUInt24Rec)
                                        : Boolean
```

Visibility: public

Description: The comparison is done by comparing `byte0`, `byte1` and `byte2`.

79.10.17 TUInt24Rec.notequal(TUInt24Rec,Cardinal):Boolean

Synopsis: Check whether a `TUInt24Rec` value differs from a cardinal value.

Declaration:
```
TUInt24Rec.class operator notequal(a: TUInt24Rec;b: Cardinal) : Boolean
```

Visibility: public

Description: The cardinal `a` is considered not equal to `b` if the fourth byte (LSB) is nonzero, or one of the first three bytes differ from `byte0`, `byte1`, and `byte2`.

79.10.18 TUInt24Rec.notequal(Cardinal,TUInt24Rec):Boolean

Synopsis: Check whether a `TUInt24Rec` value differs from a cardinal value.

Declaration:
```
TUInt24Rec.class operator notequal(a: Cardinal;b: TUInt24Rec) : Boolean
```

Visibility: public

Description: The cardinal `a` is considered not equal to `b` if the fourth byte (LSB) is nonzero, or one of the first three bytes differ from `byte0`, `byte1`, and `byte2`.

79.10.19 TUInt24Rec.greaterthan(TUInt24Rec,TUInt24Rec):Boolean

Synopsis: Check whether a `TUInt24Rec` value is greater than another `TUInt24Rec` value

Declaration:
```
TUInt24Rec.class operator greaterthan(a: TUInt24Rec;b: TUInt24Rec)
                                      : Boolean
```

Visibility: public

Description: The comparison is done by comparing 3 bytes `byte0`, `byte1` and `byte2`

79.10.20 TUInt24Rec.greaterthan(TUInt24Rec,Cardinal):Boolean

Synopsis: Check whether a `TUInt24Rec` value is greater than a cardinal value

Declaration:
```
TUInt24Rec.class operator greaterthan(a: TUInt24Rec;b: Cardinal)
                                      : Boolean
```

Visibility: public

Description: The comparison is done by comparing 3 first bytes of the cardinal value with bytes `byte0`, `byte1` and `byte2`.

79.10.21 TUInt24Rec.greaterthan(Cardinal,TUInt24Rec):Boolean

Synopsis: Check whether a cardinal value is greater than a `TUInt24Rec` value

Declaration:
```
TUInt24Rec.class operator greaterthan(a: Cardinal;b: TUInt24Rec)
                                      : Boolean
```

Visibility: public

Description: The comparison is done by comparing 3 first bytes of the cardinal value with bytes `byte0`, `byte1` and `byte2`.

79.10.22 TUInt24Rec.greaterthanorequal(TUInt24Rec,TUInt24Rec):Boolean

Synopsis: Check whether a `TUInt24Rec` value is greater than or equal to a cardinal value

Declaration:
```
TUInt24Rec.class operator greaterthanorequal(a: TUInt24Rec;
                                             b: TUInt24Rec) : Boolean
```

Visibility: public

Description: The comparison is done by comparing 3 first bytes of the cardinal value with bytes `byte0`, `byte1` and `byte2`.

79.10.23 TUInt24Rec.greaterthanorequal(TUInt24Rec,Cardinal):Boolean

Synopsis: Check whether a `TUInt24Rec` value is greater than or equal to a cardinal value

Declaration:
```
TUInt24Rec.class operator greaterthanorequal(a: TUInt24Rec;b: Cardinal)
                                             : Boolean
```

Visibility: public

Description: The comparison is done by comparing 3 first bytes of the cardinal value with bytes `byte0`, `byte1` and `byte2`.

79.10.24 TUInt24Rec.greaterthanorequal(Cardinal,TUInt24Rec):Boolean

Synopsis: Check whether a cardinal value is greater than or equal to a `TUInt24Rec` value

Declaration:
```
TUInt24Rec.class operator greaterthanorequal(a: Cardinal;b: TUInt24Rec)
                                             : Boolean
```

Visibility: public

Description: The comparison is done by comparing 3 first bytes of the cardinal value with bytes `byte0`, `byte1` and `byte2`.

79.10.25 TUInt24Rec.lessthan(TUInt24Rec,TUInt24Rec):Boolean

Synopsis: Check whether a `TUInt24Rec` value is less than another `TUInt24Rec` value

Declaration:
```
TUInt24Rec.class operator lessthan(a: TUInt24Rec;b: TUInt24Rec)
                                   : Boolean
```

Visibility: public

Description: The comparison is done by comparing the bytes `byte0`, `byte1` and `byte2`.

79.10.26 TUInt24Rec.lessthan(TUInt24Rec,Cardinal):Boolean

Synopsis: Check whether a `TUInt24Rec` value is less than a cardinal value

Declaration:
```
TUInt24Rec.class operator lessthan(a: TUInt24Rec;b: Cardinal) : Boolean
```

Visibility: public

Description: The comparison is done by comparing 3 first bytes of the cardinal value with bytes `byte0`, `byte1` and `byte2`.

79.10.27 TUInt24Rec.lessthan(Cardinal,TUInt24Rec):Boolean

Synopsis: Check whether a cardinal value is less than a `TUInt24Rec` value

Declaration:
```
TUInt24Rec.class operator lessthan(a: Cardinal;b: TUInt24Rec) : Boolean
```

Visibility: public

Description: Check whether a cardinal value is less than a `TUInt24Rec` value

79.10.28 TUInt24Rec.lessthanorequal(TUInt24Rec,TUInt24Rec):Boolean

Synopsis: Check whether a `TUInt24Rec` value is less than or equal to another `TUInt24Rec` value

Declaration:
```
TUInt24Rec.class operator lessthanorequal(a: TUInt24Rec;b: TUInt24Rec)
                                                    : Boolean
```

Visibility: public

Description: The comparison is done by comparing the bytes `byte0`, `byte1` and `byte2`.

79.10.29 TUInt24Rec.lessthanorequal(TUInt24Rec,Cardinal):Boolean

Synopsis: Check whether a `TUInt24Rec` value is less than or equal to a cardinal value

Declaration:
```
TUInt24Rec.class operator lessthanorequal(a: TUInt24Rec;b: Cardinal)
                                                    : Boolean
```

Visibility: public

Description: The comparison is done by comparing 3 first bytes of the cardinal value with bytes `byte0`, `byte1` and `byte2`.

79.10.30 TUInt24Rec.lessthanorequal(Cardinal,TUInt24Rec):Boolean

Synopsis: Check whether a cardinal value is less than or equal to a `TUInt24Rec` value

Declaration:
```
TUInt24Rec.class operator lessthanorequal(a: Cardinal;b: TUInt24Rec)
                                                    : Boolean
```

Visibility: public

Description: The comparison is done by comparing 3 first bytes of the cardinal value with bytes `byte0`, `byte1` and `byte2`.

Chapter 80

Reference for unit 'unicodeducet'

80.1 Overview

The unicodeducet unit registers the root Unicode collation (DUCET). This collation is needed by all other collations, so any collation unit will include this file.

This unit does not contain any routines. It simply registers the collation in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 81

Reference for unit 'Unix'

81.1 Used units

Table 81.1: Used units by unit 'Unix'

Name	Page
BaseUnix	51
System	622
unixtype	747

81.2 Constants, types and variables

81.2.1 Constants

`ARG_MAX = UnixType . ARG_MAX`

Maximum number of arguments to a program.

`NAME_MAX = UnixType . NAME_MAX`

Maximum filename length.

`PATH_MAX = UnixType . PATH_MAX`

Maximum pathname length.

`PRIO_PGRP = UnixType . PRIO_PGRP`

#rtl.baseunix.fpGetPriority (51) option: Get process group priority.

`PRIO_PROCESS = UnixType . PRIO_PROCESS`

#rtl.baseunix.fpGetPriority (51) option: Get process priority.

`PRIO_USER = UnixType . PRIO_USER`

#rtl.baseunix.fpGetPriority (51) option: Get user priority.

```
SIG_MAXSIG = UnixType . SIG_MAXSIG
```

Maximum system signal number.

```
SYS_NMLN = UnixType . SYS_NMLN
```

Max system name length.

81.2.2 Types

```
cbool = UnixType.cbool
```

Boolean type

```
cchar = UnixType.cchar
```

Alias for #rtl.UnixType.cchar (747)

```
cdouble = UnixType.cdouble
```

Double precision real format.

```
cfloat = UnixType.cfloat
```

Floating-point real format

```
cint = UnixType.cint
```

C type: integer (natural size)

```
cint16 = UnixType.cint16
```

C type: 16 bits sized, signed integer.

```
cint32 = UnixType.cint32
```

C type: 32 bits sized, signed integer.

```
cint64 = UnixType.cint64
```

C type: 64 bits sized, signed integer.

```
cint8 = UnixType.cint8
```

C type: 8 bits sized, signed integer.

```
clock_t = UnixType.clock_t
```

Clock ticks type

`clong = UnixType.clong`

C type: long signed integer (double sized)

`clonglong = UnixType.clonglong`

C type: 64-bit (double long) signed integer.

`coff_t = UnixType.TOff`

character offset type.

`cschar = UnixType.cschar`

Signed character type

`cshort = UnixType.cshort`

C type: short signed integer (half sized)

`csigned = UnixType.csigned`

`csigned` is an alias for cint (735).

`csint = UnixType.csint`

Signed integer

`csize_t = UnixType.size_t`

Character size type.

`cslong = UnixType.cslong`

The size is CPU dependent.

`cslonglong = UnixType.cslonglong`

`cslonglong` is an alias for clonglong (736).

`csshort = UnixType.csshort`

Short signed integer type

`cuchar = UnixType.cuchar`

Alias for #rtl.UnixType.cuchar (747)

`cuint = UnixType.cuint`

C type: unsigned integer (natural size)

```
cuint16 = UnixType.cuint16
```

C type: 16 bits sized, unsigned integer.

```
cuint32 = UnixType.cuint32
```

C type: 32 bits sized, unsigned integer.

```
cuint64 = UnixType.cuint64
```

C type: 64 bits sized, unsigned integer.

```
cuint8 = UnixType.cuint8
```

C type: 8 bits sized, unsigned integer.

```
culong = UnixType.culong
```

C type: long unsigned integer (double sized)

```
culonglong = UnixType.culonglong
```

C type: 64-bit (double long) unsigned integer.

```
cunsigned = UnixType.cunsigned
```

Alias for #rtl.unixtype.cunsigned (747)

```
cushort = UnixType.cushort
```

C type: short unsigned integer (half sized)

```
dev_t = UnixType.dev_t
```

Device descriptor type.

```
gid_t = UnixType.gid_t
```

Group ID type.

```
ino_t = UnixType.ino_t
```

Inode type.

```
mode_t = UnixType.mode_t
```

Inode mode type.

```
nlink_t = UnixType.nlink_t
```

Number of links type.

```
off_t = UnixType.off_t
```

Offset type.

```
pcbool = UnixType.pcbool
```

Pointer to boolean type cbool (735)

```
pcchar = UnixType.pcchar
```

Alias for #rtl.UnixType.pcchar (747)

```
pcdouble = UnixType.pcdouble
```

Pointer to cdouble (51) type.

```
pcfloat = UnixType.pcfloat
```

Pointer to cfloat (51) type.

```
pcint = UnixType.pcint
```

Pointer to cInt (735) type.

```
pcint16 = UnixType.pcint16
```

Pointer to 16-bit signed integer type

```
pcint32 = UnixType.pcint32
```

Pointer to signed 32-bit integer type

```
pcint64 = UnixType.pcint64
```

Pointer to signed 64-bit integer type

```
pcint8 = UnixType.pcint8
```

Pointer to 8-bits signed integer type

```
pClock = UnixType.pClock
```

Pointer to TClock (742) type.

```
pclong = UnixType.pclong
```

Pointer to cLong (736) type.

```
pclonglong = UnixType.pclonglong
```

Pointer to longlong type.

```
pcschar = UnixType.pcschar
```

Pointer to character type cschar (736).

```
pcshort = UnixType.pcshort
```

Pointer to cShort (736) type.

```
pcsigned = UnixType.pcsigned
```

Pointer to signed integer type csigned (736).

```
pcsint = UnixType.pcsint
```

Pointer to signed integer type csint (736)

```
pcsize_t = UnixType.psize_t
```

Pointer to character size type `pcsize_t`.

```
pcslong = UnixType.pcslong
```

Pointer ot the signed long cslong (736)

```
pcslonglong = UnixType.pcslonglong
```

Pointer to Signed longlong type cslonglong (736)

```
pcsshort = UnixType.pcsshort
```

Pointer to short signed integer type csshort (736)

```
pcuchar = UnixType.pcuchar
```

Alias for #rtl.UnixType.pcuchar (747)

```
pcuint = UnixType.pcuint
```

Pointer to cUInt (737) type.

```
pcuint16 = UnixType.pcuint16
```

Pointer to 16-bit unsigned integer type

```
pcuint32 = UnixType.pcuint32
```

Pointer to unsigned 32-bit integer type

```
pcuint64 = UnixType.pcuint64
```

Pointer to unsigned 64-bit integer type

```
pcuint8 = UnixType.pcuint8
```

Pointer to 8-bits unsigned integer type

```
pculong = UnixType.pculong
```

Pointer to cuLong (737) type.

```
pculonglong = UnixType.pculonglong
```

Unsigned longlong type

```
pcunsigned = UnixType.pcunsigned
```

Alias for #rtl.unixtype.pcunsigned (747)

```
pcushort = UnixType.pcushort
```

Pointer to cuShort (737) type.

```
pDev = UnixType.pDev
```

Pointer to TDev (742) type.

```
pGid = UnixType.pGid
```

Pointer to TGid (742) type.

```
pid_t = UnixType.pid_t
```

Process ID type.

```
pIno = UnixType.pIno
```

Pointer to TIno (742) type.

```
pMode = UnixType.pMode
```

Pointer to TMode (742) type.

```
pnLink = UnixType.pnLink
```

Pointer to TnLink (743) type.

```
pOff = UnixType.pOff
```

Pointer to TOff (743) type.

```
pPid = UnixType.pPid
```

Pointer to TPid (743) type.

```
pSize = UnixType.pSize
```

Pointer to TSize (743) type.

```
pSize_t = UnixType.pSize_t
```

Pointer to type `Size_t`.

```
pSocklen = UnixType.pSocklen
```

Pointer to TSockLen (743) type.

```
psSize = UnixType.psSize
```

Pointer to TsSize (743) type

```
pstatfs = UnixType.PStatFs
```

Pointer to `statfs` type

```
pthread_cond_t = UnixType.pthread_cond_t
```

Thread conditional variable type.

```
pthread_mutex_t = UnixType.pthread_mutex_t
```

Thread mutex type.

```
pthread_t = UnixType.pthread_t
```

Posix thread type.

```
pTime = UnixType.pTime
```

Pointer to TTime (743) type.

```
ptimespec = UnixType.ptimespec
```

Pointer to timespec (742) type.

```
ptimeval = UnixType.ptimeval
```

Pointer to timeval (742) type.

```
ptime_t = UnixType.ptime_t
```

Pointer to time_t (742) type.

```
pUid = UnixType.pUid
```

Pointer to TUid (743) type.

```
size_t = UnixType.size_t
```

Size specification type.

```
socklen_t = UnixType.socklen_t
```

Socket address length type.

```
ssize_t = UnixType.ssize_t
```

Small size type.

```
TClock = UnixType.TClock
```

Alias for clock_t (736) type.

```
TDev = UnixType.TDev
```

Alias for dev_t (737) type.

```
TGid = UnixType.TGid
```

Alias for gid_t (737) type.

```
timespec = UnixType.timespec
```

Short time specification type.

```
timeval = UnixType.timeval
```

Time specification type.

```
time_t = UnixType.time_t
```

Time span type

```
TIno = UnixType.TIno
```

Alias for ino_t (737) type.

```
TIOCtlRequest = UnixType.TIOCtlRequest
```

Alias for the TIOCtlRequest (747) type in unixtypes

```
TMode = UnixType.TMode
```

Alias for mode_t (737) type.

```
TnLink = UnixType.TnLink
```

Alias for nlink_t (738) type.

```
TOff = UnixType.TOff
```

Alias for off_t (738) type.

```
TPid = UnixType.TPid
```

Alias for pid_t (740) type.

```
TSize = UnixType.TSize
```

Alias for size_t (742) type

```
TSocklen = UnixType.TSocklen
```

Alias for socklen_t (742) type.

```
TsSize = UnixType.TsSize
```

Alias for ssize_t (742) type

```
tstatfs = UnixType.TStatFs
```

`StatFS` returns in `Info` information about the filesystem on which the file `Path` resides. `Info` is of type TStatFS (747).
The function returns zero if the call was succesful, a nonzero value is returned if the call failed.

```
TTime = UnixType.TTime
```

Alias for TTime (743) type.

```
Ttimespec = UnixType.Ttimespec
```

Alias for TimeSpec (742) type.

```
TTimeVal = UnixType.TTimeVal
```

Alias for timeval (742) type.

```
TUid = UnixType.TUid
```

Alias for uid_t (743) type.

```
uid_t = UnixType.uid_t
```

User ID type

Chapter 82

Reference for unit 'unixcp'

82.1 Used units

Table 82.1: Used units by unit 'unixcp'

Name	Page
BaseUnix	51
System	622

82.2 Overview

The unixcp unit provides routines to handle mapping of code page names to numerical values as used in libiconv>. The GetCodepageByName (745) function is the main function for this. The GetCodepageData (745) can be used to map a code page number to a name. These function can be used for instance to map code page information in environment variables to code page numbers used in string encodings. The supported code page names are the ones commonly in use in libiconv.

This unit is used for example in unit cwstring (115).

82.3 Constants, types and variables

82.3.1 Constants

```
UnixCpMap : Array[-1..UnixCpMapLimit] of TUnixCpData = ((cp: 0; name: 'UTF-8'), (cp:
```

`UnixCpMap` is a fixed structure with codepage number/codepage name pairs. It is used in GetCodepageData (745), GetSystemCodepage (745) and GetCodepageByName (745) to map code page names to numbers and vice versa.

The map is ordered on code page number, and for equal code page numbers, the names are ordered so the most common one is used first.

```
UnixCpMapLimit = 406 - 83
```

Number of code pages in map `UnixCpMap`.

82.3.2 Types

```
TUnixCpData = record
  cp : Word;
  name : ansistring;
end
```

TUnixCpData contains 2 fields necessary to construct a map between code page number (cp) and name (name).

82.4 Procedures and functions

82.4.1 GetCodepageByName

Synopsis: Find code page by name

Declaration: `function GetCodepageByName(cpname: rawbytestring) : TSystemCodePage`

Visibility: default

Description: GetCodepageByName returns the code page number matching cpname. The supported code page names are the ones commonly in use in libiconv. Names are searched case-sentively, with the exception that 'cpN' is converted to 'CPN', where N is a digit.

Errors: If no matching code page name is found, CP_NONE is returned.

See also: UnixCpMap (744), GetSystemCodepage (745), GetCodepageData (745)

82.4.2 GetCodepageData

Synopsis: Return index of codepage.

Declaration: `function GetCodepageData(cp: TSystemCodePage) : LongInt`

Visibility: default

Description: GetCodepageData returns the index of the first entry in UnixCpMap (744) which matches cp. Since the entries are ordered by code page number, this means the entries can be scanned for alternate code names starting at this index.

Errors: If no matching code page is found, -1 is returned.

See also: UnixCpMap (744), GetSystemCodepage (745), GetCodepageByName (745)

82.4.3 GetSystemCodepage

Synopsis: Return the system code page based on the program environment.

Declaration: `function GetSystemCodepage : TSystemCodePage`

Visibility: default

Description: GetSystemCodepage returns the system code page, based on one of the environment variables LC_ALL, LC_CTYPE or LANG. The first non-empty variable (in the order mentioned here) is used.

Errors: If none is found, then a system default is used: Linux and Darwin use `CP_UTF8`, others use `CP_ASCII`.

See also: UnixCpMap (744), GetSystemCodepage (745), GetCodepageByName (745)

Chapter 83

Reference for unit 'unixtype'

83.1 Overview

The unixtype unit contains the definitions of basic unix types. It was initially implemented by Marco van de Voort.

When porting to a new unix platform, this unit should be adapted to the sizes and conventions of the platform to which the compiler is ported.

Chapter 84

Reference for unit 'unixutil'

84.1 Overview

The UnixUtil unit contains some of the routines that were present in the old Linux unit, but which do not really belong in the unix (734) or baseunix (51) units.

Most of the functions described here have cross-platform counterparts in the SysUtils (628) unit. It is therefore recommended to use that unit.

84.2 Constants, types and variables

84.2.1 Types

```
ComStr = string deprecated
```

Command-line string type.

```
DirStr = string deprecated
```

Filename directory part string type.

```
ExtStr = string deprecated
```

Filename extension part string type.

```
NameStr = string deprecated
```

Filename name part string type.

```
PathStr = string deprecated
```

Filename full path string type.

84.2.2 Variables

```
Tzseconds : LongInt
```

Seconds west of GMT

84.3 Procedures and functions

84.3.1 ArrayStringToPPchar

Synopsis: Concert an array of string to an array of null-terminated strings

Declaration:
```
function ArrayStringToPPchar(const S: Array of AnsiString;
                             reserveentries: LongInt) : ppchar
```

Visibility: default

Description: `ArrayStringToPPchar` creates an array of null-terminated strings that point to strings which are the same as the strings in the array `S`. The function returns a pointer to this array. The array and the strings it contains must be disposed of after being used, because it they are allocated on the heap.

The `ReserveEntries` parameter tells `ArrayStringToPPchar` to allocate room at the end of the array for another `ReserveEntries` entries.

Errors: If not enough memory is available, an error may occur.

See also: StringToPPChar (751)

84.3.2 EpochToLocal

Synopsis: Convert epoch time to local time

Declaration:
```
procedure EpochToLocal(epoch: LongInt;var year: Word;var month: Word;
                       var day: Word;var hour: Word;var minute: Word;
                       var second: Word)
```

Visibility: default

Description: Converts the epoch time (=Number of seconds since 00:00:00, January 1, 1970, corrected for your time zone) to local date and time.

This function takes into account the timzeone settings of your system.

Errors: None

See also: LocalToEpoch (751)

Listing: ./unutilex/ex3.pp

```
Program Example3;

{ Program to demonstrate the EpochToLocal function. }

Uses BaseUnix,Unix,UnixUtil;

Var Year,month,day,hour,minute,seconds : Word;

begin
  EpochToLocal (FPTime,Year,month,day,hour,minute,seconds);
  Writeln ('Current date : ',Day:2,'/',Month:2,'/',Year:4);
  Writeln ('Current time : ',Hour:2,':',minute:2,':',seconds:2);
end.
```

84.3.3 GetFS

Synopsis: Return file selector

Declaration:
```
function GetFS(var T: Text) : LongInt
function GetFS(var F: File) : LongInt
```

Visibility: default

Description: `GetFS` returns the file selector that the kernel provided for your file. In principle you don' need this file selector. Only for some calls it is needed, such as the #rtl.baseunix.fpSelect (51) call or so.

Errors: In case the file was not opened, then -1 is returned.

See also: #rtl.baseunix.fpSelect (51)

Listing: ./unutilex/ex34.pp

```
Program Example33;

{ Program to demonstrate the SelectText function. }

Uses Unix;

Var tv : TimeVal;

begin
  WriteLn ('Press the <ENTER> to continue the program.');
  { Wait until File descriptor 0 (=Input) changes }
  SelectText (Input,nil);
  { Get rid of <ENTER> in buffer }
  readln;
  WriteLn ('Press <ENTER> key in less than 2 seconds...');
  tv.tv_sec:=2;
  tv.tv_sec:=0;
  if SelectText (Input,@tv)>0 then
    WriteLn ('Thank you !')
  else
    WriteLn ('Too late !');
end.
```

84.3.4 GregorianToJulian

Synopsis: Converts a gregorian date to a julian date

Declaration:
```
function GregorianToJulian(Year: LongInt;Month: LongInt;Day: LongInt)
                           : LongInt
```

Visibility: default

Description: `GregorianToJulian` takes a gregorian date and converts it to a Julian day.

Errors: None.

See also: JulianToGregorian (751)

84.3.5 JulianToGregorian

Synopsis: Converts a julian date to a gregorian date

Declaration:
```
procedure JulianToGregorian(JulianDN: LongInt;var Year: Word;
                            var Month: Word;var Day: Word)
```

Visibility: default

Description: `JulianToGregorian` takes a julian day and converts it to a gregorian date. (Start of the Julian Date count is from 0 at 12 noon 1 JAN -4712 (4713 BC),)

Errors: None.

See also: GregorianToJulian (750)

84.3.6 LocalToEpoch

Synopsis: Convert local time to epoch (unix) time

Declaration:
```
function LocalToEpoch(year: Word;month: Word;day: Word;hour: Word;
                      minute: Word;second: Word) : LongInt
```

Visibility: default

Description: Converts the Local time to epoch time (=Number of seconds since 00:00:00, January 1, 1970).

Errors: None

See also: EpochToLocal (749)

Listing: ./unutilex/ex4.pp

```
Program Example4;

{ Program to demonstrate the LocalToEpoch function. }

Uses UnixUtil;

Var year,month,day,hour,minute,second : Word;

begin
  Write ('Year    : ');readln(Year);
  Write ('Month   : ');readln(Month);
  Write ('Day     : ');readln(Day);
  Write ('Hour    : ');readln(Hour);
  Write ('Minute  : ');readln(Minute);
  Write ('Seonds  : ');readln(Second);
  Write ('This is : ');
  Write (LocalToEpoch(year,month,day,hour,minute,second));
  Writeln (' seconds past 00:00 1/1/1980');
end.
```

84.3.7 StringToPPChar

Synopsis: Split string in list of null-terminated strings

Declaration: `function StringToPPChar(S: PChar;ReserveEntries: Integer) : ppchar`
`function StringToPPChar(var S: AnsiString;ReserveEntries: Integer) : ppchar`

Visibility: default

Description: `StringToPPChar` splits the string `S` in words, replacing any whitespace with zero characters. It returns a pointer to an array of pchars that point to the first letters of the words in `S`. This array is terminated by a `Nil` pointer.

The function does *not* add a zero character to the end of the string unless it ends on whitespace.

The function reserves memory on the heap to store the array of `PChar`; The caller is responsible for freeing this memory.

This function can be called to create arguments for the various `Exec` calls.

Errors: None.

See also: ArrayStringToPPchar (749), #rtl.baseunix.FpExecve (51)

Listing: ./unutilex/ex70.pp

```
Program Example70;

{ Program to demonstrate the StringToPPchar function. }

Uses UnixUtil;

Var S : String;
    P : PPChar;
    I : longint;

begin
  // remark whitespace at end.
  S:='This is a string with words. ';
  P:=StringToPPChar(S,0);
  I:=0;
  While P[i]<>Nil do
    begin
    Writeln('Word ',i,' : ',P[i]);
    Inc(I);
    end;
  FreeMem(P,i*SizeOf(Pchar));
end.
```

Chapter 85

Reference for unit 'video'

85.1 Overview

The Video unit implements a screen access layer which is system independent. It can be used to write on the screen in a system-independent way, which should be optimal on all platforms for which the unit is implemented.

The working of the Video is simple: After calling InitVideo (753), the array `VideoBuf` contains a representation of the video screen of size `ScreenWidth*ScreenHeight`, going from left to right and top to bottom when walking the array elements: `VideoBuf[0]` contains the character and color code of the top-left character on the screen. `VideoBuf[ScreenWidth]` contains the data for the character in the first column of the second row on the screen, and so on.

To write to the 'screen', the text to be written should be written to the `VideoBuf` array. Calling UpdateScreen (753) will then cp the text to the screen in the most optimal way. (an example can be found further on).

The color attribute is a combination of the foreground and background color, plus the blink bit. The bits describe the various color combinations:

bits 0-3 The foreground color. Can be set using all color constants.

bits 4-6 The background color. Can be set using a subset of the color constants.

bit 7 The blinking bit. If this bit is set, the character will appear blinking.

Each possible color has a constant associated with it, see the constants section for a list of constants.

The foreground and background color can be combined to a color attribute with the following code:

```
Attr:=ForeGroundColor + (BackGroundColor shl 4);
```

The color attribute can be logically or-ed with the blink attribute to produce a blinking character:

```
Atrr:=Attr or blink;
```

But not all drivers may support this.

The contents of the `VideoBuf` array may be modified: This is 'writing' to the screen. As soon as everything that needs to be written in the array is in the `VideoBuf` array, calling `UpdateScreen` will copy the contents of the array screen to the screen, in a manner that is as efficient as possible.

The updating of the screen can be prohibited to optimize performance; To this end, the LockScreenUpdate (753) function can be used: This will increment an internal counter. As long as the counter differs from zero, calling UpdateScreen (753) will not do anything. The counter can be lowered with UnlockScreenUpdate (753). When it reaches zero, the next call to UpdateScreen (753) will actually update the screen. This is useful when having nested procedures that do a lot of screen writing.

The video unit also presents an interface for custom screen drivers, thus it is possible to override the default screen driver with a custom screen driver, see the SetVideoDriver (753) call. The current video driver can be retrieved using the GetVideoDriver (753) call.

Remark: The video unit should *not* be used together with the crt unit. Doing so will result in very strange behaviour, possibly program crashes.

85.2 Examples utility unit

The examples in this section make use of the unit vidutil, which contains the `TextOut` function. This function writes a text to the screen at a given location. It looks as follows:

Listing: ./videoex/vidutil.pp

```
unit vidutil;

Interface

uses
  video;

Procedure TextOut(X,Y : Word;Const S : String);

Implementation

Procedure TextOut(X,Y : Word;Const S : String);

Var
  W,P,I,M : Word;

begin
  P:=((X-1)+(Y-1)*ScreenWidth);
  M:=Length(S);
  If P+M>ScreenWidth*ScreenHeight then
    M:=ScreenWidth*ScreenHeight-P;
  For I:=1 to M do
    VideoBuf^[P+I-1]:=Ord(S[i])+($07 shl 8);
end;

end.
```

85.3 Writing a custom video driver

Writing a custom video driver is not difficult, and generally means implementing a couple of functions, which whould be registered with the SetVideoDriver (753) function. The various functions that can be implemented are located in the TVideoDriver (753) record:

```
TVideoDriver = Record
  InitDriver        : Procedure;
```

```
  DoneDriver        : Procedure;
  UpdateScreen      : Procedure(Force : Boolean);
  ClearScreen       : Procedure;
  SetVideoMode      : Function (Const Mode : TVideoMode) : Boolean;
  GetVideoModeCount : Function : Word;
  GetVideoModeData  : Function(Index : Word; Var Data : TVideoMode) : Boolean;
  SetCursorPos      : procedure (NewCursorX, NewCursorY: Word);
  GetCursorType     : function : Word;
  SetCursorType     : procedure (NewType: Word);
  GetCapabilities   : Function : Word;
end;
```

Not all of these functions must be implemented. In fact, the only absolutely necessary function to write a functioning driver is the `UpdateScreen` function. The general calls in the Video unit will check which functionality is implemented by the driver.

The functionality of these calls is the same as the functionality of the calls in the video unit, so the expected behaviour can be found in the previous section. Some of the calls, however, need some additional remarks.

InitDriver Called by `InitVideo`, this function should initialize any data structures needed for the functionality of the driver, maybe do some screen initializations. The function is guaranteed to be called only once; It can only be called again after a call to `DoneVideo`. The variables `ScreenWidth` and `ScreenHeight` should be initialized correctly after a call to this function, as the `InitVideo` call will initialize the `VideoBuf` and `OldVideoBuf` arrays based on their values.

DoneDriver This should clean up any structures that have been initialized in the `InitDriver` function. It should possibly also restore the screen as it was before the driver was initialized. The VideoBuf and `OldVideoBuf` arrays will be disposed of by the general `DoneVideo` call.

UpdateScreen This is the only required function of the driver. It should update the screen based on the `VideoBuf` array's contents. It can optimize this process by comparing the values with values in the `OldVideoBuf` array. After updating the screen, the `UpdateScreen` procedure should update the `OldVideoBuf` by itself. If the `Force` parameter is `True`, the whole screen should be updated, not just the changed values.

ClearScreen If there is a faster way to clear the screen than to write spaces in all character cells, then it can be implemented here. If the driver does not implement this function, then the general routines will write spaces in all video cells, and will call `UpdateScreen(True)`.

SetVideoMode Should set the desired video mode, if available. It should return `True` if the mode was set, `False` if not.

GetVideoModeCount Should return the number of supported video modes. If no modes are supported, this function should not be implemented; the general routines will return 1. (for the current mode)

GetVideoModeData Should return the data for the `Index`-th mode; `Index` is zero based. The function should return true if the data was returned correctly, false if `Index` contains an invalid index. If this is not implemented, then the general routine will return the current video mode when `Index` equals 0.

GetCapabilities If this function is not implemented, zero (i.e. no capabilities) will be returned by the general function.

The following unit shows how to override a video driver, with a driver that writes debug information to a file. The unit can be used in any of the demonstration programs, by simply including it in the `uses` clause. Setting `DetailedVideoLogging` to `True` will create a more detailed log (but will also slow down functioning)

Listing: ./videoex/viddbg.pp

```
unit viddbg;

Interface

uses video;


Procedure StartVideoLogging;
Procedure StopVideoLogging;
Function  IsVideoLogging : Boolean;
Procedure  SetVideoLogFileName(FileName : String);

Const
  DetailedVideoLogging : Boolean = False;

Implementation

uses sysutils,keyboard;

var
  NewVideoDriver,
  OldVideoDriver : TVideoDriver;
  Active,Logging : Boolean;
  LogFileName : String;
  VideoLog : Text;

Function TimeStamp : String;

begin
  TimeStamp:=FormatDateTime('hh:nn:ss',Time());
end;

Procedure StartVideoLogging;

begin
  Logging:=True;
  WriteIn(VideoLog,'Start logging video operations at: ',TimeStamp);
end;

Procedure StopVideoLogging;

begin
  WriteIn(VideoLog,'Stop logging video operations at: ',TimeStamp);
  Logging:=False;
end;

Function IsVideoLogging : Boolean;

begin
  IsVideoLogging:=Logging;
end;
```

```
Var
  ColUpd,RowUpd : Array[0..1024] of Integer;

Procedure DumpScreenStatistics(Force : Boolean);

Var
  I,Count : Integer;

begin
  If Force then
    Write(VideoLog,'forced ');
  WriteIn(VideoLog,'video update at ',TimeStamp,' : ');
  FillChar(Colupd,SizeOf(ColUpd),#0);
  FillChar(Rowupd,SizeOf(RowUpd),#0);
  Count:=0;
  For I:=0 to VideoBufSize div SizeOf(TVideoCell) do
    begin
    If VideoBuf^[i]<>OldVideoBuf^[i] then
      begin
      Inc(Count);
      Inc(ColUpd[I mod ScreenWidth]);
      Inc(RowUpd[I div ScreenHeight]);
      end;
    end;
  Write(VideoLog,Count,' videocells differed divided over ');
  Count:=0;
  For I:=0 to ScreenWidth-1 do
    If ColUpd[I]<>0 then
      Inc(Count);
  Write(VideoLog,Count,' columns and ');
  Count:=0;
  For I:=0 to ScreenHeight-1 do
    If RowUpd[I]<>0 then
      Inc(Count);
  WriteIn(VideoLog,Count,' rows.');
  If DetailedVideoLogging Then
   begin
   For I:=0 to ScreenWidth-1 do
     If (ColUpd[I]<>0) then
       WriteIn(VideoLog,'Col ',i,' : ',ColUpd[I]:3,' rows changed');
   For I:=0 to ScreenHeight-1 do
     If (RowUpd[I]<>0) then
       WriteIn(VideoLog,'Row ',i,' : ',RowUpd[I]:3,' colums changed');
   end;
end;

Procedure LogUpdateScreen(Force : Boolean);

begin
  If Logging then
    DumpScreenStatistics(Force);
  OldVideoDriver.UpdateScreen(Force);
end;

Procedure LogInitVideo;

begin
  OldVideoDriver.InitDriver();
```

```
  Assign(VideoLog,logFileName);
  Rewrite(VideoLog);
  Active:=True;
  StartVideoLogging;
end;

Procedure LogDoneVideo;

begin
  StopVideoLogging;
  Close(VideoLog);
  Active:=False;
  OldVideoDriver.DoneDriver();
end;

Procedure SetVideoLogFileName(FileName : String);

begin
  If Not Active then
    LogFileName:=FileName;
end;

Initialization
  GetVideoDriver(OldVideoDriver);
  NewVideoDriver:=OldVideoDriver;
  NewVideoDriver.UpdateScreen:=@LogUpdateScreen;
  NewVideoDriver.InitDriver:=@LogInitVideo;
  NewVideoDriver.DoneDriver:=@LogDoneVideo;
  LogFileName:='Video.log';
  Logging:=False;
  SetVideoDriver(NewVideoDriver);
end.
```

Chapter 86

Reference for unit 'wincrt'

86.1 Overview

The wincrt unit provides some auxiliary routines for use with the graph (318) unit, namely keyboard support. It has no connection with the crt (112) unit, nor with the Turbo-Pascal for Windows WinCrt unit. As such, it should not be used by end users. Refer to the crt (112) unit instead.

86.2 Constants, types and variables

86.2.1 Variables

```
directvideo : Boolean
```

On windows, this variable is ignored.

```
lastmode : Word
```

Is supposed to contain the last used video mode, but is actually unused.

86.3 Procedures and functions

86.3.1 delay

Synopsis: Pause program execution

Declaration: `procedure delay(ms: Word)`

Visibility: default

Description: `Delay` stops program execution for the indicated number `ms` of milliseconds.

See also: sound (760), nosound (760)

86.3.2 keypressed

Synopsis: Check if a key was pressed.

Declaration: `function keypressed : Boolean`

Visibility: default

Description: `KeyPressed` returns `True` if the user pressed a key, or `False` if not. It does not wait for the user to press a key.

See also: readkey (760)

86.3.3 nosound

Synopsis: Stop the speaker

Declaration: `procedure nosound`

Visibility: default

Description: `NoSound` does nothing, windows does not support this.

See also: sound (760)

86.3.4 readkey

Synopsis: Read a key from the keyboard

Declaration: `function readkey : Char`

Visibility: default

Description: `ReadKey` reads a key from the keyboard, and returns the ASCII value of the key, or the scancode of the key in case it is a special key.

The function waits until a key is pressed.

See also: KeyPressed (759)

86.3.5 sound

Synopsis: Sound PC speaker

Declaration: `procedure sound(hz: Word)`

Visibility: default

Description: `Sound` sounds the PC speaker. It emits a tone with frequency `Hz` for 500 milliseconds. (the time argument is required by the windows API)

See also: nosound (760)

86.3.6 textmode

Synopsis: Set indicated text mode

Declaration: `procedure textmode(mode: Integer)`

Visibility: default

Description: `TextMode` does nothing.

Chapter 87

Reference for unit 'windirs'

87.1 Used units

Table 87.1: Used units by unit 'windirs'

Name	Page
System	622
windows	??

87.2 Overview

This unit contains only one function: GetWindowsSpecialDir (764). It is a simple pascal wrapper around the shfolder.dll library, and is available under Windows only.

87.3 Constants, types and variables

87.3.1 Constants

`CSIDL_ADMINTOOLS = $0030`

ID for the personal "Start Menu\Programs\Administrative tools" folder (In the user's personal folder)

`CSIDL_APPDATA = $001A`

ID for the roaming "Application Data" folder (in the user's personal folder)

`CSIDL_CDBURN_AREA = $003B`

ID for the personal "Local Settings\Application Data\Microsoft\CD Burning" folder (Under "All users" folder)

`CSIDL_COMMON_ADMINTOOLS = $002F`

ID for the common "Start Menu\Programs\Administrative tools" folder (Under "All users" folder)

CSIDL_COMMON_APPDATA = $0023

ID for the common "Application Data" folder (Under "All users" folder)

CSIDL_COMMON_DESKTOPDIRECTORY = $0019

ID for the public "Desktop" folder (Under "All users" folder)

CSIDL_COMMON_DOCUMENTS = $002E

ID for the common "Documents" folder (Under "All users" folder)

CSIDL_COMMON_FAVORITES = $001F

ID for the public "Favorites" folder (Under "All users" folder)

CSIDL_COMMON_MUSIC = $0035

ID for the common "Documents\My Music" folder (Under "All users" folder)

CSIDL_COMMON_PICTURES = $0036

ID for the common "Documents\My Pictures" folder (Under "All users" folder)

CSIDL_COMMON_PROGRAMS = $0017

ID for the public "Programs" folder (Under "All users" Start menu folder)

CSIDL_COMMON_STARTMENU = $0016

ID for the public "Start Menu" folder (Under "All users")

CSIDL_COMMON_STARTUP = $0018

ID for the public "Startup" folder (Under "All users\Start menu\Programs" folder)

CSIDL_COMMON_TEMPLATES = $002D

ID for the common "Templates" folder (Under "All users" folder)

CSIDL_COMMON_VIDEO = $0037

ID for the common "Documents\My Video" folder (Under "All users" folder)

CSIDL_COOKIES = $0021

ID for the "Cookies" folder (in the user's personal folder)

CSIDL_DESKTOPDIRECTORY = $0010

ID for the "Desktop" folder (in the user's personal folder)

CSIDL_FAVORITES = $0006

ID for the "Favourites" folder (in the user's personal folder)

CSIDL_FLAG_CREATE = $8000

Flag to specify if the requested folder must be created if it didn't exist yet.

CSIDL_HISTORY = $0022

ID for the "History" folder (in the user's personal folder)

CSIDL_INTERNET_CACHE = $0020

ID for the "Temporary Internet Files" folder (in the user's personal folder)

CSIDL_LOCAL_APPDATA = $001C

ID for the local "Application Data" folder (in the user's personal folder)

CSIDL_MYMUSIC = $000D

ID for the "My Music" folder (in the user's personal folder)

CSIDL_MYPICTURES = $0027

ID for the "My Pictures" folder (in the user's personal folder)

CSIDL_MYVIDEO = $000E

ID for the "My Videos" folder (in the user's personal folder)

CSIDL_NETHOOD = $0013

ID for the "Nethood" folder (in the user's personal folder)

CSIDL_PERSONAL = $0005

ID for the "My Documents" folder (in the user's personal folder)

CSIDL_PRINTHOOD = $001B

ID for the "Printhood" folder (in the user's personal folder)

CSIDL_PROFILE = $0028

ID for the user's personal folder

CSIDL_PROFILES = $003E

ID for the parent folder for the user's folders

`CSIDL_PROGRAMS = $0002`

ID for the "Program files" folder

`CSIDL_PROGRAM_FILES = $0026`

ID for the "Program Files" folder (alias for `CSIDL_PROGRAMS`)

`CSIDL_PROGRAM_FILES_COMMON = $002B`

ID for the Common folder under the "Program Files" folder.

`CSIDL_RECENT = $0008`

ID for the "Recent" folder (in the user's personal folder)

`CSIDL_SENDTO = $0009`

ID for the "Send to" folder (in the user's personal folder)

`CSIDL_STARTMENU = $000B`

ID for the "Start menu" folder (in the user's personal folder)

`CSIDL_STARTUP = $0007`

ID for the "Startup" menu folder (in the user's personal folder)

`CSIDL_SYSTEM = $0025`

ID for the Windows System32 dir.

`CSIDL_TEMPLATES = $0015`

ID for the "Templates" folder (in the user's personal folder)

`CSIDL_WINDOWS = $0024`

ID for the Windows installation dir.

87.4 Procedures and functions

87.4.1 GetWindowsSpecialDir

Synopsis: Get the location of a special directory.

Declaration: `function GetWindowsSpecialDir(ID: Integer) : string`

Visibility: default

Description: `GetWindowsSpecialDir` can be used to to get the path of special folders on the Windows system. The locations are identified using one of the `CSIDL_*` constants. If the ID of a location is or-ed with the `CSIDL_FLAG_CREATE` constant, then the directory will be created if it didn't exist yet (and the user has sufficient rights to do so).

Errors: On error, an empty string is returned.

Chapter 88

Reference for unit 'x86'

88.1 Used units

Table 88.1: Used units by unit 'x86'

Name	Page
BaseUnix	51
System	622

88.2 Overview

The x86 unit contains some of the routines that were present in the 1.0.X Linux unit, and which were Intel (PC) architecture specific.

These calls have been preserved for compatibility, but should be considered deprecated: they are not portable and may not even work on future linux versions.

88.3 Procedures and functions

88.3.1 fpIOperm

Synopsis: Set permission on IO ports

Declaration: `function fpIOperm(From: Cardinal;Num: Cardinal;Value: cint) : cint`

Visibility: default

Description: `FpIOperm` sets permissions on `Num` ports starting with port `From` to `Value`. The function returns zero if the call was successfull, a nonzero value otherwise.

Note:

- This works ONLY as root.
- Only the first `0x03ff` ports can be set.
- When doing a FpFork (51), the permissions are reset. When doing a FpExecVE (51) they are kept.

Errors: Extended error information can be retrieved with FpGetErrno (51)

88.3.2 fpIoPL

Synopsis: Set I/O privilege level

Declaration:
```
function fpIoPL(Level: cint) : cint
```

Visibility: default

Description: `FpIoPL` sets the I/O privilige level. It is intended for completeness only, one should normally not use it.

88.3.3 ReadPort

Synopsis: Read data from a PC port

Declaration:
```
procedure ReadPort(Port: LongInt;var Value: Byte)
procedure ReadPort(Port: LongInt;var Value: LongInt)
procedure ReadPort(Port: LongInt;var Value: Word)
```

Visibility: default

Description: `ReadPort` reads one Byte, Word or Longint from port `Port` into `Value`.

Note that you need permission to read a port. This permission can be set by the root user with the FpIOPerm (765) call.

Errors: In case of an error (not enough permissions read this port), runtime 216 (*Access Violation*) will occur.

See also: FpIOPerm (765), ReadPortB (766), ReadPortW (767), ReadPortL (767), WritePort (767), WritePortB (768), WritePortL (768), WritePortW (768)

88.3.4 ReadPortB

Synopsis: Read bytes from a PC port

Declaration:
```
function ReadPortB(Port: LongInt) : Byte
procedure ReadPortB(Port: LongInt;var Buf;Count: LongInt)
```

Visibility: default

Description: The procedural form of `ReadPortB` reads `Count` bytes from port `Port` and stores them in `Buf`. There must be enough memory allocated at `Buf` to store `Count` bytes.

The functional form of `ReadPortB` reads 1 byte from port `B` and returns the byte that was read.

Note that you need permission to read a port. This permission can be set by the root user with the FpIOPerm (765) call.

Errors: In case of an error (not enough permissions read this port), runtime 216 (*Access Violation*) will occur.

See also: FpIOPerm (765), ReadPort (766), ReadPortW (767), ReadPortL (767), WritePort (767), WritePortB (768), WritePortL (768), WritePortW (768)

88.3.5 ReadPortL

Synopsis: Read longints from a PC port

Declaration:
```
function ReadPortL(Port: LongInt) : LongInt
procedure ReadPortL(Port: LongInt;var Buf;Count: LongInt)
```

Visibility: default

Description: The procedural form of `ReadPortL` reads `Count` longints from port `Port` and stores them in `Buf`. There must be enough memory allocated at `Buf` to store `Count` Longints.

The functional form of `ReadPortL` reads 1 longint from port `B` and returns the longint that was read.

Note that you need permission to read a port. This permission can be set by the root user with the FpIOPerm (765) call.

Errors: In case of an error (not enough permissions read this port), runtime 216 (*Access Violation*) will occur.

See also: FpIOPerm (765), ReadPort (766), ReadPortW (767), ReadPortB (766), WritePort (767), WritePortB (768), WritePortL (768), WritePortW (768)

88.3.6 ReadPortW

Synopsis: Read Words from a PC port

Declaration:
```
function ReadPortW(Port: LongInt) : Word
procedure ReadPortW(Port: LongInt;var Buf;Count: LongInt)
```

Visibility: default

Description: The procedural form of `ReadPortW` reads `Count` words from port `Port` and stores them in `Buf`. There must be enough memory allocated at `Buf` to store `Count` words.

The functional form of `ReadPortW` reads 1 word from port `B` and returns the word that was read.

Note that you need permission to read a port. This permission can be set by the root user with the FpIOPerm (765) call.

Errors: In case of an error (not enough permissions read this port), runtime 216 (*Access Violation*) will occur.

See also: FpIOPerm (765), ReadPort (766), ReadPortB (766), ReadPortL (767), WritePort (767), WritePortB (768), WritePortL (768), WritePortW (768)

88.3.7 WritePort

Synopsis: Write data to PC port

Declaration:
```
procedure WritePort(Port: LongInt;Value: Byte)
procedure WritePort(Port: LongInt;Value: LongInt)
procedure WritePort(Port: LongInt;Value: Word)
```

Visibility: default

Description: `WritePort` writes `Value` – 1 byte, Word or longint – to port `Port`.

Remark: You need permission to write to a port. This permission can be set with root permission with the FpIOPerm (765) call.

Errors: In case of an error (not enough permissions to write to this port), runtime 216 (*Access Violation*) will occur.

See also: FpIOPerm (765), WritePortB (768), WritePortL (768), WritePortW (768), ReadPortB (766), ReadPortL (767), ReadPortW (767)

88.3.8 WritePortB

Synopsis: Write byte to PC port

Declaration:
```
procedure WritePortB(Port: LongInt;Value: Byte)
procedure WritePortB(Port: LongInt;var Buf;Count: LongInt)
```

Visibility: default

Description: The first form of `WritePortB` writes 1 byte to port `Port`. The second form writes `Count` bytes from `Buf` to port `Port`.

Remark: You need permission to write to a port. This permission can be set with root permission with the FpIOPerm (765) call.

Errors: In case of an error (not enough permissions to write to this port), runtime 216 (*Access Violation*) will occur.

See also: FpIOPerm (765), WritePort (767), WritePortL (768), WritePortW (768), ReadPortB (766), ReadPortL (767), ReadPortW (767)

88.3.9 WritePortl

Synopsis: Write longint to PC port.

Declaration:
```
procedure WritePortL(Port: LongInt;Value: LongInt)
procedure WritePortl(Port: LongInt;var Buf;Count: LongInt)
```

Visibility: default

Description: The first form of `WritePortB` writes 1 byte to port `Port`. The second form writes `Count` bytes from `Buf` to port `Port`.

Remark: You need permission to write to a port. This permission can be set with root permission with the FpIOPerm (765) call.

Errors: In case of an error (not enough permissions to write to this port), runtime 216 (*Access Violation*) will occur.

See also: FpIOPerm (765), WritePort (767), WritePortB (768), WritePortW (768), ReadPortB (766), ReadPortL (767), ReadPortW (767)

88.3.10 WritePortW

Synopsis: Write Word to PC port

Declaration:
```
procedure WritePortW(Port: LongInt;Value: Word)
procedure WritePortW(Port: LongInt;var Buf;Count: LongInt)
```

Visibility: default

Description: The first form of `WritePortB` writes 1 byte to port `Port`. The second form writes `Count` bytes from `Buf` to port `Port`.

Remark: You need permission to write to a port. This permission can be set with root permission with the FpIOPerm (765) call.

Errors: In case of an error (not enough permissions to write to this port), runtime 216 (*Access Violation*) will occur.

See also: FpIOPerm (765), WritePort (767), WritePortL (768), WritePortB (768), ReadPortB (766), ReadPortL (767), ReadPortW (767)