# Common pascal units documentation

Pasdoc

November 30, 2005

# Contents

# Chapter 1

# Unit collects

## 1.1 Description

Collection units

This routine contains collection objects, being quite similar to those included in the objects unit. The only difference being that they compile on all compiler targets.

## 1.2 Overview

`TStackItem record`

`TStack Object`

`TExtendedCollection Object` Base collection object

`TExtendedSortedCollection Object` Base sorted collection object

`TExtendedStringCollection Object` String pointer collection object

`TExtendedSortedStringCollection Object` Sorted string pointer collection object

`TLongintCollection Object`

## 1.3 Classes, Interfaces, Objects and Records

### TStackItem record

#### Fields

**next** `next: PStackItem;`

**data** `data: pointer;`

## TStack Object

### Hierarchy

TStack > TObject

### Methods

**init**

**Declaration** `public constructor init;`

**done**

**Declaration** `public destructor done;`

**push**

**Declaration** `public procedure push(p: pointer);`

**pop**

**Declaration** `public function pop: pointer;`

**peek**

**Declaration** `public function peek: pointer;`

**isEmpty**

**Declaration** `public function isEmpty: boolean;`

## TExtendedCollection Object

### Hierarchy

TExtendedCollection > TObject

### Description

Base collection object

### Fields

**Items** `public Items: PItemList;`

**Count** `public Count: Integer;`

**Limit** `public Limit: Integer;`

**Delta** `public Delta: Integer;`

## Methods

**Init**

**Declaration** `public CONSTRUCTOR Init (ALimit, ADelta:  Integer);`

**Done**

**Declaration** `public DESTRUCTOR Done; Virtual;`

**At**

**Declaration** `public FUNCTION At (Index:  Integer):  Pointer;`

**IndexOf**

**Declaration** `public FUNCTION IndexOf (Item:  Pointer):  Integer; Virtual;`

**LastThat**

**Declaration** `public FUNCTION LastThat (Test:  Pointer):  Pointer;`

**FirstThat**

**Declaration** `public FUNCTION FirstThat (Test:  Pointer):  Pointer;`

**Pack**

**Declaration** `public PROCEDURE Pack;`

**FreeAll**

**Declaration** `public PROCEDURE FreeAll;`

**DeleteAll**

**Declaration** `public PROCEDURE DeleteAll;`

**Free**

**Declaration** `public PROCEDURE Free (Item:  Pointer);`

**Insert**

**Declaration** `public PROCEDURE Insert (Item:  Pointer); Virtual;`

**Delete**

**Declaration** `public PROCEDURE Delete (Item:  Pointer);`

**AtFree**

**Declaration** `public PROCEDURE AtFree (Index:  Integer);`

**FreeItem**

**Declaration** `public PROCEDURE FreeItem (Item:  Pointer); Virtual;`

**AtDelete**

**Declaration** `public PROCEDURE AtDelete (Index:  Integer);`

**ForEach**

**Declaration** `public PROCEDURE ForEach (Action:  Pointer);`

**SetLimit**

**Declaration** `public PROCEDURE SetLimit (ALimit:  Integer); Virtual;`

**Error**

**Declaration** `public PROCEDURE Error (Code, Info:  Integer); Virtual;`

**AtPut**

**Declaration** `public PROCEDURE AtPut (Index:  Integer; Item:  Pointer);`

**AtInsert**

**Declaration** `public PROCEDURE AtInsert (Index:  Integer; Item:  Pointer);`

## TExtendedSortedCollection Object

### Hierarchy

TExtendedSortedCollection > `TExtendedCollection`(1.3) > TObject

### Description

Base sorted collection object

### Fields

**Duplicates** `public Duplicates: Boolean;`

### Methods

#### Init

**Declaration** `public CONSTRUCTOR Init (ALimit, ADelta: Integer);`

#### KeyOf

**Declaration** `public FUNCTION KeyOf (Item: Pointer): Pointer; Virtual;`

#### IndexOf

**Declaration** `public FUNCTION IndexOf (Item: Pointer): Integer; Virtual;`

#### Compare

**Declaration** `public FUNCTION Compare (Key1, Key2: Pointer): Integer; Virtual;`

#### Search

**Declaration**
```
public FUNCTION Search (Key: Pointer; Var Index: Integer): Boolean;
Virtual;
```

#### Insert

**Declaration** `public PROCEDURE Insert (Item: Pointer); Virtual;`

## TExtendedStringCollection Object

### Hierarchy

TExtendedStringCollection > `TExtendedCollection`(1.3) > TObject

### Description

String pointer collection object

This collection accepts pointers to shortstrings as input. The data is not sorted.

### Methods

#### FreeItem

**Declaration** `public PROCEDURE FreeItem (Item: Pointer); Virtual;`

### TExtendedSortedStringCollection Object

#### Hierarchy

TExtendedSortedStringCollection > `TExtendedSortedCollection`(1.3) > `TExtendedCollection`(1.3) > TObject

#### Description

Sorted string pointer collection object

This collection accepts pointers to shortstrings as input. The data is sorted as it is added in.

#### Methods

**Init**

**Declaration** `public CONSTRUCTOR Init (ALimit, ADelta:  Integer);`

**Compare**

**Declaration** `public FUNCTION Compare (Key1, Key2:  Pointer):  Integer; Virtual;`

**FreeItem**

**Declaration** `public PROCEDURE FreeItem (Item:  Pointer); Virtual;`

### TLongintCollection Object

#### Hierarchy

TLongintCollection > `TExtendedCollection`(1.3) > TObject

#### Description

no description available, TExtendedCollection description followsBase collection object

#### Methods

**FreeItem**

**Declaration** `public procedure FreeItem(Item:  pointer); virtual;`

## 1.4 Types

### TItemList

**Declaration** `TItemList = Array [0..MaxCollectionSize - 1] Of Pointer;`

### PItemList

**Declaration** `PItemList = ^TItemList;`

### PStackItem

**Declaration** `PStackItem = ^TStackItem;`

### PStack

**Declaration** `PStack = ^TStack;`

**Description** Stack object

This implement an object that is used as a LIFO stack containing pointers as data.

### ProcedureType

**Declaration** `ProcedureType = Function(Item: Pointer): Boolean;`

### ForEachProc

**Declaration** `ForEachProc = Procedure(Item: Pointer);`

### PExtendedCollection

**Declaration** `PExtendedCollection = ^TExtendedCollection;`

### PExtendedSortedCollection

**Declaration** `PExtendedSortedCollection = ^TExtendedSortedCollection;`

### PExtendedStringCollection

**Declaration** `PExtendedStringCollection = ^TExtendedStringCollection;`

### PExtendedSortedStringCollection

**Declaration** `PExtendedSortedStringCollection = ^TExtendedSortedStringCollection;`

## 1.5 Constants

### MaxCollectionSize

**Declaration** `MaxCollectionSize = 8192;`

### coIndexError

**Declaration** `coIndexError = -1;`

### coOverflow

**Declaration** `coOverflow = -2;`

## 1.6 Author

Carl Eric Codere

# Chapter 2

# Unit crc

## 2.1 Description

CRC and checksum generation unit

CRC and checksum generation routines, compatible with ISO 3309 and ITU-T-V42 among others.

## 2.2 Overview

`UpdateCrc32` Calculates a CRC-32 CCITT value

`UpdateCrc16` Calculates a CRC-16 CCITT value

`UpdateAdler32` Calculates an Adler-32 checksum value

`UpdateFletcher8` Calculates an 8-bit fletcher checksum value

`UpdateCRC` Calculates a standard 16-bit CRC

## 2.3 Functions and Procedures

### UpdateCrc32

**Declaration** `function UpdateCrc32(InitCrc:longword; b:  byte):longword;`

**Description** Calculates a CRC-32 CCITT value

Routine to get the CRC-32 CCITT value.

Normally to be compatible with the ISO 3309 standard, the first call to this routine should set `InitCRC` to `$FFFFFFFF`, and the final result of the CRC-32 should be `XOR`'ed with $FFFFFFFF.

**Parameters** **InitCRC** The value of the previous CRC

**b** The data byte to get the CRC-32 of

**Returns** The updated CRC-32 value

## UpdateCrc16

**Declaration** `function UpdateCrc16(InitCrc: word; b: byte): word;`

**Description** Calculates a CRC-16 CCITT value

Routine to get the CRC-16 CCITT value.

Normally to be compatible with the CCITT standards, the first call to this routine should set `InitCRC` to `$FFFF`, and the final result of the CRC-16 should be taken as is.

p.s : This has not been verified against hardware.

**Parameters** **InitCRC** The value of the previous CRC

**b** The data byte to get the CRC-16 of

**Returns** The updated CRC-16 value

## UpdateAdler32

**Declaration** `function UpdateAdler32(InitAdler: longword; b: byte): longword;`

**Description** Calculates an Adler-32 checksum value

Routine to get the Adler-32 checksum as defined in IETF RFC 1950.

Normally to be compatible with the standard, the first call to this routine should set `InitAdler` to `1`, and the final result of the should be taken as is.

**Parameters** **InitAdler** The value of the previous Adler32

**b** The data byte to get the Adler32 of

**Returns** The updated Adler32 value

## UpdateFletcher8

**Declaration** `function UpdateFletcher8(InitFletcher: word; b: byte): word;`

**Description** Calculates an 8-bit fletcher checksum value

Routine to get the Fletcher 8-bit checksum as defined in IETF RFC 1146

Normally to be compatible with the standard, the first call to this routine should set `InitFletcher` to `0`, and the final result of the should be taken as is.

**Parameters** **InitCRC** The value of the previous Adler32

**b** The data byte to get the Adler32 of

**Returns** The updated Adler32 value

### UpdateCRC

**Declaration** `function UpdateCRC(InitCrc: word; b: byte): word;`

**Description** Calculates a standard 16-bit CRC

Standard CRC-16 bit algorithm as used in the ARC archiver.

The first call to this routine should set `InitCRC` to `0`, and the final result of the should be taken as is.

**Parameters** **InitCRC** The value of the previous Crc

**b** The data byte to get the Crc of

**Returns** The updated Crc value

## 2.4 Constants

### crctable32

**Declaration**

```
crctable32:array[0..255] of longword = ( $00000000, $77073096, $ee0e612c,
$990951ba, $076dc419, $706af48f, $e963a535, $9e6495a3, $0edb8832, $79dcb8a4,
$e0d5e91e, $97d2d988, $09b64c2b, $7eb17cbd, $e7b82d07, $90bf1d91, $1db71064,
$6ab020f2, $f3b97148, $84be41de, $1adad47d, $6ddde4eb, $f4d4b551, $83d385c7,
$136c9856, $646ba8c0, $fd62f97a, $8a65c9ec, $14015c4f, $63066cd9, $fa0f3d63,
$8d080df5, $3b6e20c8, $4c69105e, $d56041e4, $a2677172, $3c03e4d1, $4b04d447,
$d20d85fd, $a50ab56b, $35b5a8fa, $42b2986c, $dbbbc9d6, $acbcf940, $32d86ce3,
$45df5c75, $dcd60dcf, $abd13d59, $26d930ac, $51de003a, $c8d75180, $bfd06116,
$21b4f4b5, $56b3c423, $cfba9599, $b8bda50f, $2802b89e, $5f058808, $c60cd9b2,
$b10be924, $2f6f7c87, $58684c11, $c1611dab, $b6662d3d, $76dc4190, $01db7106,
$98d220bc, $efd5102a, $71b18589, $06b6b51f, $9fbfe4a5, $e8b8d433, $7807c9a2,
$0f00f934, $9609a88e, $e10e9818, $7f6a0dbb, $086d3d2d, $91646c97, $e6635c01,
$6b6b51f4, $1c6c6162, $856530d8, $f262004e, $6c0695ed, $1b01a57b, $8208f4c1,
$f50fc457, $65b0d9c6, $12b7e950, $8bbeb8ea, $fcb9887c, $62dd1ddf, $15da2d49,
$8cd37cf3, $fbd44c65, $4db26158, $3ab551ce, $a3bc0074, $d4bb30e2, $4adfa541,
$3dd895d7, $a4d1c46d, $d3d6f4fb, $4369e96a, $346ed9fc, $ad678846, $da60b8d0,
$44042d73, $33031de5, $aa0a4c5f, $dd0d7cc9, $5005713c, $270241aa, $be0b1010,
$c90c2086, $5768b525, $206f85b3, $b966d409, $ce61e49f, $5edef90e, $29d9c998,
$b0d09822, $c7d7a8b4, $59b33d17, $2eb40d81, $b7bd5c3b, $c0ba6cad, $edb88320,
$9abfb3b6, $03b6e20c, $74b1d29a, $ead54739, $9dd277af, $04db2615, $73dc1683,
$e3630b12, $94643b84, $0d6d6a3e, $7a6a5aa8, $e40ecf0b, $9309ff9d, $0a00ae27,
$7d079eb1, $f00f9344, $8708a3d2, $1e01f268, $6906c2fe, $f762575d, $806567cb,
$196c3671, $6e6b06e7, $fed41b76, $89d32be0, $10da7a5a, $67dd4acc, $f9b9df6f,
$8ebeeff9, $17b7be43, $60b08ed5, $d6d6a3e8, $a1d1937e, $38d8c2c4, $4fdff252,
$d1bb67f1, $a6bc5767, $3fb506dd, $48b2364b, $d80d2bda, $af0a1b4c, $36034af6,
$41047a60, $df60efc3, $a867df55, $316e8eef, $4669be79, $cb61b38c, $bc66831a,
$256fd2a0, $5268e236, $cc0c7795, $bb0b4703, $220216b9, $5505262f, $c5ba3bbe,
$b2bd0b28, $2bb45a92, $5cb36a04, $c2d7ffa7, $b5d0cf31, $2cd99e8b, $5bdeae1d,
```

```
$9b64c2b0, $ec63f226, $756aa39c, $026d930a, $9c0906a9, $eb0e363f, $72076785,
$05005713, $95bf4a82, $e2b87a14, $7bb12bae, $0cb61b38, $92d28e9b, $e5d5be0d,
$7cdcefb7, $0bdbdf21, $86d3d2d4, $f1d4e242, $68ddb3f8, $1fda836e, $81be16cd,
$f6b9265b, $6fb077e1, $18b74777, $88085ae6, $ff0f6a70, $66063bca, $11010b5c,
$8f659eff, $f862ae69, $616bffd3, $166ccf45, $a00ae278, $d70dd2ee, $4e048354,
$3903b3c2, $a7672661, $d06016f7, $4969474d, $3e6e77db, $aed16a4a, $d9d65adc,
$40df0b66, $37d83bf0, $a9bcae53, $debb9ec5, $47b2cf7f, $30b5ffe9, $bdbdf21c,
$cabac28a, $53b39330, $24b4a3a6, $bad03605, $cdd70693, $54de5729, $23d967bf,
$b3667a2e, $c4614ab8, $5d681b02, $2a6f2b94, $b40bbe37, $c30c8ea1, $5a05df1b,
$2d02ef8d );
```

## crctable16ccitt

**Declaration**

```
crctable16ccitt:  ARRAY[0..255] OF WORD = ( $0000, $1021, $2042, $3063,
$4084, $50a5, $60c6, $70e7, $8108, $9129, $a14a, $b16b, $c18c, $d1ad, $e1ce,
$f1ef, $1231, $0210, $3273, $2252, $52b5, $4294, $72f7, $62d6, $9339, $8318,
$b37b, $a35a, $d3bd, $c39c, $f3ff, $e3de, $2462, $3443, $0420, $1401, $64e6,
$74c7, $44a4, $5485, $a56a, $b54b, $8528, $9509, $e5ee, $f5cf, $c5ac, $d58d,
$3653, $2672, $1611, $0630, $76d7, $66f6, $5695, $46b4, $b75b, $a77a, $9719,
$8738, $f7df, $e7fe, $d79d, $c7bc, $48c4, $58e5, $6886, $78a7, $0840, $1861,
$2802, $3823, $c9cc, $d9ed, $e98e, $f9af, $8948, $9969, $a90a, $b92b, $5af5,
$4ad4, $7ab7, $6a96, $1a71, $0a50, $3a33, $2a12, $dbfd, $cbdc, $fbbf, $eb9e,
$9b79, $8b58, $bb3b, $ab1a, $6ca6, $7c87, $4ce4, $5cc5, $2c22, $3c03, $0c60,
$1c41, $edae, $fd8f, $cdec, $ddcd, $ad2a, $bd0b, $8d68, $9d49, $7e97, $6eb6,
$5ed5, $4ef4, $3e13, $2e32, $1e51, $0e70, $ff9f, $efbe, $dfdd, $cffc, $bf1b,
$af3a, $9f59, $8f78, $9188, $81a9, $b1ca, $a1eb, $d10c, $c12d, $f14e, $e16f,
$1080, $00a1, $30c2, $20e3, $5004, $4025, $7046, $6067, $83b9, $9398, $a3fb,
$b3da, $c33d, $d31c, $e37f, $f35e, $02b1, $1290, $22f3, $32d2, $4235, $5214,
$6277, $7256, $b5ea, $a5cb, $95a8, $8589, $f56e, $e54f, $d52c, $c50d, $34e2,
$24c3, $14a0, $0481, $7466, $6447, $5424, $4405, $a7db, $b7fa, $8799, $97b8,
$e75f, $f77e, $c71d, $d73c, $26d3, $36f2, $0691, $16b0, $6657, $7676, $4615,
$5634, $d94c, $c96d, $f90e, $e92f, $99c8, $89e9, $b98a, $a9ab, $5844, $4865,
$7806, $6827, $18c0, $08e1, $3882, $28a3, $cb7d, $db5c, $eb3f, $fb1e, $8bf9,
$9bd8, $abbb, $bb9a, $4a75, $5a54, $6a37, $7a16, $0af1, $1ad0, $2ab3, $3a92,
$fd2e, $ed0f, $dd6c, $cd4d, $bdaa, $ad8b, $9de8, $8dc9, $7c26, $6c07, $5c64,
$4c45, $3ca2, $2c83, $1ce0, $0cc1, $ef1f, $ff3e, $cf5d, $df7c, $af9b, $bfba,
$8fd9, $9ff8, $6e17, $7e36, $4e55, $5e74, $2e93, $3eb2, $0ed1, $1ef0 );
```

## crctable16

**Declaration**

```
crctable16:  ARRAY[0..255] OF WORD = (
$00000,$0C0C1,$0C181,$00140,$0C301,$003C0,$00280,$0C241,
$0C601,$006C0,$00780,$0C741,$00500,$0C5C1,$0C481,$00440,
$0CC01,$00CC0,$00D80,$0CD41,$00F00,$0CFC1,$0CE81,$00E40,
$00A00,$0CAC1,$0CB81,$00B40,$0C901,$009C0,$00880,$0C841,
$0D801,$018C0,$01980,$0D941,$01B00,$0DBC1,$0DA81,$01A40,
```

```
$01E00,$0DEC1,$0DF81,$01F40,$0DD01,$01DC0,$01C80,$0DC41,
$01400,$0D4C1,$0D581,$01540,$0D701,$017C0,$01680,$0D641,
$0D201,$012C0,$01380,$0D341,$01100,$0D1C1,$0D081,$01040,
$0F001,$030C0,$03180,$0F141,$03300,$0F3C1,$0F281,$03240,
$03600,$0F6C1,$0F781,$03740,$0F501,$035C0,$03480,$0F441,
$03C00,$0FCC1,$0FD81,$03D40,$0FF01,$03FC0,$03E80,$0FE41,
$0FA01,$03AC0,$03B80,$0FB41,$03900,$0F9C1,$0F881,$03840,
$02800,$0E8C1,$0E981,$02940,$0EB01,$02BC0,$02A80,$0EA41,
$0EE01,$02EC0,$02F80,$0EF41,$02D00,$0EDC1,$0EC81,$02C40,
$0E401,$024C0,$02580,$0E541,$02700,$0E7C1,$0E681,$02640,
$02200,$0E2C1,$0E381,$02340,$0E101,$021C0,$02080,$0E041,
$0A001,$060C0,$06180,$0A141,$06300,$0A3C1,$0A281,$06240,
$06600,$0A6C1,$0A781,$06740,$0A501,$065C0,$06480,$0A441,
$06C00,$0ACC1,$0AD81,$06D40,$0AF01,$06FC0,$06E80,$0AE41,
$0AA01,$06AC0,$06B80,$0AB41,$06900,$0A9C1,$0A881,$06840,
$07800,$0B8C1,$0B981,$07940,$0BB01,$07BC0,$07A80,$0BA41,
$0BE01,$07EC0,$07F80,$0BF41,$07D00,$0BDC1,$0BC81,$07C40,
$0B401,$074C0,$07580,$0B541,$07700,$0B7C1,$0B681,$07640,
$07200,$0B2C1,$0B381,$07340,$0B101,$071C0,$07080,$0B041,
$05000,$090C1,$09181,$05140,$09301,$053C0,$05280,$09241,
$09601,$056C0,$05780,$09741,$05500,$095C1,$09481,$05440,
$09C01,$05CC0,$05D80,$09D41,$05F00,$09FC1,$09E81,$05E40,
$05A00,$09AC1,$09B81,$05B40,$09901,$059C0,$05880,$09841,
$08801,$048C0,$04980,$08941,$04B00,$08BC1,$08A81,$04A40,
$04E00,$08EC1,$08F81,$04F40,$08D01,$04DC0,$04C80,$08C41,
$04400,$084C1,$08581,$04540,$08701,$047C0,$04680,$08641,
$08201,$042C0,$04380,$08341,$04100,$081C1,$08081,$04040 );
```

## 2.5 Author

Carl Eric Codere

# Chapter 3

# Unit dateutil

## 3.1 Description

Date and time utility routines

This unit is quite similar to the unit dateutils provided with Delphi 6 and Delphi 7. Only a subset of the API found in those units is implemented in this unit, furthermore it contains a new set of extended API's included in the dateexth.inc file.
There are subtle differences with the Delphi implementation: 1. All string related parameters and function results use ISO 8601 formatted date and time strings. 2. The internal format of TDatetime is not the same as on the Delphi compilers (Internally TDateTime is stored as a Julian date) 3. The milliseconds field is only an approximation, and should not be considered as accurate. 4. Becasue everything is coded with floats, the seconds field has a precision of +/- 2 seconds.
All dates are assumed to be in Gregorian calendar date format (This is a proleptic Gregorian calendar unit).

## 3.2 Overview

`TDateInfo record`

`tfiletime packed record`

`CurrentYear`

`Date`

`DateOf`

`DateTimeToStr`

`DateToStr`

`DayOf`

`DaysBetween`

DecodeDate

DecodeDateTime

DecodeTime

HourOf

IncDay

IncHour

IncMilliSecond

IncMinute

IncSecond

IncWeek

IsPM

IsValidDate

IsValidDateTime

IsValidTime

MinuteOf

MonthOf

Now

SameDate

SameDateTime

SameTime

SecondOf

Time

GetTime

TimeOf

TimeToStr

Today

TryEncodeDate

TryEncodeTime

`TryEncodeDateTime`

`TryStrToDate`

`TryStrToDateTime`

`TryStrToTime`

`YearOf`

`TryStrToDateTimeExt`

`TryEncodeDateAndTimeToStr`

`DateTimeToStrExt`

`GetCurrentDate` Returns the current date set in the operating system

`GetCurrentTime` Returns the current time set in the operating system

`TryUNIXToDateTimeExt`

`TryFileTimeToDateTimeExt`

## 3.3 Classes, Interfaces, Objects and Records

### TDateInfo record

#### Description

Useful structure that contains additional information on a date and time

#### Fields

**DateTime** `DateTime: TDateTime;`

Actual date and time value *

**UTC** `UTC: boolean;`

Is this value local or according to UTC?

### tfiletime packed record

#### Description

Win32 FILETIME timestamp

#### Fields

**LowDateTime** `LowDateTime: longword;`

**HighDateTime** `HighDateTime: longword;`

## 3.4 Functions and Procedures

### CurrentYear

**Declaration** `function CurrentYear: word;`

**Description** Returns the current year

### Date

**Declaration** `function Date: TDatetime;`

**Description** Returns the current date, with the time value equal to midnight.

### DateOf

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

**Description** Strips the time portion from a TDateTime value.

### DateTimeToStr

**Declaration** `function DateTimeToStr(DateTime: TDateTime): string;`

**Description** Converts a TDateTime value to a string in standard ISO 8601 format.

### DateToStr

**Declaration** `function DateToStr(date: TDatetime): string;`

**Description** Converts a TDatetime value to a string in ISO 8601 format

### DayOf

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

**Description** Returns the day of the month represented by a TDateTime value.

### DaysBetween

**Declaration** `function DaysBetween(const ANow, AThen: TDateTime): integer;`

**Description** Returns the number of days between two specified TDateTime values.

### DecodeDate

**Declaration** `procedure DecodeDate(Date: TDateTime; var Year, Month, Day: Word);`

**Description** Returns Year, Month, and Day values for a TDateTime value.

## DecodeDateTime

**Declaration** `procedure DecodeDateTime(const AValue: TDateTime; var Year, Month, Day, Hour, Minute, Second, MilliSecond: Word);`

**Description** Returns Year, Month, Day, Hour, Minute, Second, and Millisecond values for a TDateTime.

## DecodeTime

**Declaration** `procedure DecodeTime(Time: TDateTime; var Hour, Min, Sec, MSec: Word);`

**Description** Breaks a TDateTime value into hours, minutes, seconds, and milliseconds.

## HourOf

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

**Description** Returns the hour of the day represented by a TDateTime value.

## IncDay

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

**Description** Returns a date shifted by a specified number of days.

## IncHour

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

**Description** Returns a date/time value shifted by a specified number of hours.

## IncMilliSecond

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

**Description** Returns a date/time value shifted by a specified number of milliseconds.

## IncMinute

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

**Description** Returns a date/time value shifted by a specified number of minutes.

## IncSecond

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

**Description** Returns a date/time value shifted by a specified number of seconds.

## IncWeek

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

**Description** Returns a date shifted by a specified number of weeks.

## IsPM

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

**Description** Indicates whether the time portion of a specified TDateTime value occurs after noon.

## IsValidDate

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

**Description** Indicates whether a specified year, month, and day represent a valid date.

## IsValidDateTime

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

**Description** Indicates whether a specified year, month, day, hour, minute, second, and millisecond represent a valid date and time.

## IsValidTime

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

**Description** Indicates whether a specified hour, minute, second, and millisecond represent a valid date and time.

## MinuteOf

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

**Description** Returns the minute of the hour represented by a TDateTime value.

## MonthOf

**Declaration** `function MonthOf(const AValue: TDateTime): Word;`

**Description** Returns the month of the year represented by a TDateTime value.

## Now

**Declaration** `function Now: TDateTime;`

**Description** Returns the current date and time.

## SameDate

**Declaration** `function SameDate(const A, B: TDateTime): Boolean;`

**Description** Indicates whether two TDateTime values represent the same year, month, and day.

## SameDateTime

**Declaration** `function SameDateTime(const A, B: TDateTime): Boolean;`

**Description** Indicates whether two TDateTime values represent the same year, month, day, hour, minute, second, and millisecond.

## SameTime

**Declaration** `function SameTime(const A, B: TDateTime): Boolean;`

**Description** Indicates whether two TDateTime values represent the same time of day, ignoring the date portion.

## SecondOf

**Declaration** `function SecondOf(const AValue: TDateTime): Word;`

**Description** Returns the second of the minute represented by a TDateTime value.

## Time

**Declaration** `function Time: TDateTime;`

**Description** Returns the current time.

## GetTime

**Declaration** `function GetTime: TDateTime;`

**Description** Returns the current time.

## TimeOf

**Declaration** `function TimeOf(const AValue: TDateTime): TDatetime;`

**Description** Strips the date portion from a TDatetime value

## TimeToStr

**Declaration** `function TimeToStr(Time: TDateTime): string;`

**Description** Returns a string that represents a TDateTime value.

## Today

**Declaration** `function Today: TDateTime;`

**Description** Returns a TDateTime value that represents the current date.

## TryEncodeDate

**Declaration**
```
function TryEncodeDate(Year, Month, Day: Word; var Date: TDateTime):
Boolean;
```

**Description** Returns a TDateTime value that represents a specified Year, Month, and Day.

## TryEncodeTime

**Declaration**
```
function TryEncodeTime(Hour, Min, Sec, MSec: Word; var Time: TDateTime):
Boolean;
```

**Description** Returns a TDateTime value for a specified Hour, Min, Sec, and MSec.

## TryEncodeDateTime

**Declaration**
```
function TryEncodeDateTime(const AYear, AMonth, ADay, AHour, AMinute,
ASecond, AMilliSecond: Word; var AValue: TDateTime): Boolean;
```

**Description** Returns a TDateTime that represents a specified year, month, day, hour, minute, second, and millisecond.

## TryStrToDate

**Declaration** `function TryStrToDate(const S: string; var Value: TDateTime): Boolean;`

**Description** Converts a string to a TDateTime value, with a Boolean success code.

In the case where the date does not contain the full representation of a date (for examples, YYYY or YYYY-MM), then the missing values will be set to 1 to be legal.

## TryStrToDateTime

**Declaration** `function TryStrToDateTime(const S: string; var Value: TDateTime): Boolean;`

**Description** Converts a string to a TDateTime value with a Boolean success code.

Supported formats: 1) Format of Complete Representation for calendar dates (as specified in ISO 8601), which should include the Time designator character. 2) Format: 'YYYY-MM-DD HH:mm:ss [GMT—UTC—UT]' 3) Openoffice 1.1.x HTML date format: 'YYYYMMDD;HHmmssuu' 4) Adobe PDF 'D:YYYYMMDDHHMMSSOHH'mm" format

The returned value will be according to UTC if timezone information is uncluded, otherwise, it will be left as is. To determine if the value was actually converted to UTC, use TryStrToDateTimeExt.

In the case where the date does not contain the full representation of a date (for examples, YYYY or YYYY-MM), then the missing values will be set to 1 to be legal.

## TryStrToTime

**Declaration** `function TryStrToTime(const S: string; var Value: TDateTime): Boolean;`

**Description** Converts a string to a TDateTime value with an error default,

Supported formats: 1) ISO 8601 time format (complete representation) with optional timezone designators. 2) Format: 'HH:mm:ss [GMT—UTC—UT]' 3) Openoffice 1.1.x HTML time format: 'HHmmssuu' 4) Adobe PDF 'D:YYYYMMDDHHMMSSOHH'mm" format

The returned value will be according to UTC if timezone information is specified. The Date field is truncated and is equal to zero upon return.

## YearOf

**Declaration** `function YearOf(const AValue: TDateTime): Word;`

**Description** Returns the year represented by a TDateTime value.

## TryStrToDateTimeExt

**Declaration** `function TryStrToDateTimeExt(const S: string; var Value: TDateTime; var UTC: boolean) : Boolean;`

**Description** Converts a string to a TDateTime value with a Boolean success code. This routine also gives information if the value was successfully converted to UTC time or not (if no timezone information was available in the string then the utc value will be false).

Supported formats: 1) Format of Complete Representation for calendar dates (as specified in ISO 8601), which should include the Time designator character. 3) Format: 'YYYY-MM-DD HH:mm:ss [GMT—UTC—UT]' 4) Openoffice 1.1.x HTML date format: 'YYYYMMDD;HHmmssuu' 5) Adobe PDF 'D:YYYYMMDDHHMMSSOHH'mm" format

The returned value will be according to UTC if timezone information is specified.

In the case where the date does not contain the full representation of a date (for examples, YYYY or YYYY-MM), then the missing values will be set to 1 to be legal.

### TryEncodeDateAndTimeToStr

**Declaration** `function TryEncodeDateAndTimeToStr(const Year, Month, Day, Hour, Minute, Second, MilliSecond: word; UTC: boolean; var AValue: string):boolean;`

**Description** This routine encodes a complete date and time to its string representation. The encoded string conforms to the ISO 8601 complete representation extended format (YYYY-MM-DDTHH:MM:SS[Z]).

The year value is required, while all other fields are optional. The other fields can be set to EMPTY_DATETIME_FIELD to indicate that they are empty. It also adds the UTC marker if required and if it is set and time information is present.

### DateTimeToStrExt

**Declaration** `function DateTimeToStrExt(DateTime: TDateTime; utc: boolean): string;`

### GetCurrentDate

**Declaration** `procedure GetCurrentDate(var Year,Month,Day,DayOfWeek: integer);`

**Description** Returns the current date set in the operating system

### GetCurrentTime

**Declaration** `procedure GetCurrentTime(var Hour,Minute,Second,Sec100: integer);`

**Description** Returns the current time set in the operating system

Ranges of the values returned are Hour 0..23, Minute 0..59, Second 0..60, Sec100 0..99.

### TryUNIXToDateTimeExt

**Declaration** `function TryUNIXToDateTimeExt(unixtime: big_integer_t; var DateTime: TDateTime; var UTC: boolean): boolean;`

### TryFileTimeToDateTimeExt

**Declaration** `function TryFileTimeToDateTimeExt(ftime: tfiletime; var DateTime: TDateTime; var UTC: boolean): boolean;`

## 3.5 Types

### TDatetime

**Declaration** `TDatetime = real;`

**Description** This is the Julian Day number

### float

**Declaration** `float = real;`

### platformword

**Declaration** `platformword = system.word;`

## 3.6 Constants

### DayMonday

**Declaration** `DayMonday = 1;`

### DayTuesday

**Declaration** `DayTuesday = 2;`

### DayWednesday

**Declaration** `DayWednesday = 3;`

### DayThursday

**Declaration** `DayThursday = 4;`

### DayFriday

**Declaration** `DayFriday = 5;`

### DaySaturday

**Declaration** `DaySaturday = 6;`

### DaySunday

**Declaration** `DaySunday = 7;`

## 3.7 Author

Carl Eric Codere

# Chapter 4

# Unit fileio

## 4.1 Description

File I/O unit

This a replacement File I/O unit containing routines to access files on disk. They are a better replacement of the standard system I/O routines since they support larger file sizes, as well as debugging features.
If DEBUG is defined, when the application quits, files that are still opened will be displayed on the console. It is important to note, that only the files opened with this API can be checked this way.

## 4.2 Overview

`FileAssign` Assign a filename to a file

`FileReset` Open a file for reading or writing

`FileRewrite` Open/Overwrite a file

`FileClose` Close a previously opened file

`FileBlockRead` Read data from a file

`FileBlockWrite` Write data to a file

`FileSeek` Change the file pointer position of an opened file

`FileGetSize` Get the size of a file in bytes

`FileGetPos` Return the current file pointer position

`FileIOResult` Return the result of the last I/O operation

`FileTruncate` Truncate a file at the current file position

## 4.3 Functions and Procedures

### FileAssign

**Declaration** `procedure FileAssign(var F: file; const Name:  string);`

**Description** Assign a filename to a file

This is uased to assign a filename with a file. This assignment will then permit to operate on the file. The assignment is completely compatible with the standard `AssignFile` system unit routine.

### FileReset

**Declaration** `procedure FileReset(var F: file; mode:  integer);`

**Description** Open a file for reading or writing

This opens a file using a specified file mode. The filemonde constants are defined in other units (fmXXXX constants). The file should have previously been assigned using `FileAssign`.

### FileRewrite

**Declaration** `procedure FileRewrite(var F: file; mode:  integer);`

**Description** Open/Overwrite a file

This creates a file or overwrites a file (if it does not exist). Currently the mode constant is not used, since the file is always opened in read/write mode.

The file should have previously been assigned using `FileAssign`.

### FileClose

**Declaration** `procedure FileClose(var F: file);`

**Description** Close a previously opened file

This closes a file that was previously opened by a call to FileOpen or FileReset.

### FileBlockRead

**Declaration** `function FileBlockRead(var F: file; var Buf; Count:  integer):  integer;`

**Description** Read data from a file

Reads data bytes from the specified opened file. Returns the number of bytes actually read.

## FileBlockWrite

**Declaration** `function FileBlockWrite(var F: file; var Buf; Count: integer): integer;`

**Description** Write data to a file

Write data bytes to the specified opened file. Returns the number of bytes actually written.

## FileSeek

**Declaration** `procedure FileSeek(var F: file; N: big_integer_t);`

**Description** Change the file pointer position of an opened file

Seeks to a specific file position in a file (starting from zero which is the start of the file).

## FileGetSize

**Declaration** `function FileGetSize(var F: file): big_integer_t;`

**Description** Get the size of a file in bytes

Returns the current size of the file in bytes. The file must be assigned and opened to access this routine.

## FileGetPos

**Declaration** `function FileGetPos(var F: file): big_integer_t;`

**Description** Return the current file pointer position

Returns the current file pointer position within the file. The start of the file is at position zero.

## FileIOResult

**Declaration** `function FileIOResult: integer;`

**Description** Return the result of the last I/O operation

Returns the I/O Result of the last operation. If this routine is not called and the unit is compiled with the DEBUG define, and if there was an error in the last operation, then the next File I/O operation will cause a runtime error with the I/O error code.

## FileTruncate

**Declaration** `procedure FileTruncate(var F: file);`

**Description** Truncate a file at the current file position

Truncates a file at the current file position. File must be assigned and opened.

## 4.4 Author

Carl Eric Codere

# Chapter 5

# Unit ietf

## 5.1 Description

ietf/web related support unit

This unit contains routines to validate strings, and characters according to different IETF standards (such as URL's, URI's and MIME types).

## 5.2 Overview

`mime_isvalidcontenttype`

`langtag_isvalid`

`langtag_split`

`uri_split` Extract information from an URI string

`urn_isvalid` Verifies the validity of a complete URN string

`urn_isvalidnid`

`urn_split` Splits an URN string in its separate components

`urn_pathsplit`

`http_pathsplit`

`file_pathsplit`

## 5.3 Functions and Procedures

### mime_isvalidcontenttype

**Declaration** `function mime_isvalidcontenttype(const s: shortstring): boolean;`

## langtag_isvalid

**Declaration** `function langtag_isvalid(const s: string): boolean;`

## langtag_split

**Declaration** `function langtag_split(const s: string; var primary,sub: string): boolean;`

## uri_split

**Declaration** `function uri_split(url: string; var scheme, authority,path, query: string): boolean;`

**Description** Extract information from an URI string

Given an URI complete specification string, extract and return the scheme, authority, path and query components of the URI. The exact definition of these terms is specified in IETF RFC 2396.

**Parameters** **url** URI to check

**Returns** FALSE if the URI is not valid, otherwise returns TRUE

## urn_isvalid

**Declaration** `function urn_isvalid(s: shortstring): boolean;`

**Description** Verifies the validity of a complete URN string

This checks the conformance of the URN address. It is based on IETF RFC 2141.

**Returns** TRUE if this is a valid URN string

## urn_isvalidnid

**Declaration** `function urn_isvalidnid(nid: string): boolean;`

**Description** This routine checks that the specified NID (namespace) is either registered to IANA, or that it is an experimental NID, as described in IETF RFC 2611. More assignment information can be obtained from: http://www.iana.org/assignments/urn-namespaces

**Returns** TRUE if this is a registered or experimental NID string

## urn_split

**Declaration** `function urn_split(urn:string; var urnidstr,nidstr,nssstr: string): boolean;`

**Description** Splits an URN string in its separate components

It is based on IETF RFC 2141.

(nidstr Namespace identifier NID

**Parameters** **urn** Complete URN string to separate

**urnidstr** Signature URN:

**nssstr** Namespace specific string NSS

**Returns** TRUE if the operation was successfull, or FALSE if the URN is malformed

### urn_pathsplit

**Declaration** `function urn_pathsplit(path: string; var namespace, nss: string): boolean;`

**Description** Splits a path string returned by uri_split into its individual components for URN.

### http_pathsplit

**Declaration** `function http_pathsplit(path: string; var directory, name: string): boolean;`

**Description** Splits a path string returned by uri_split into its individual components for the http URI.

### file_pathsplit

**Declaration** `function file_pathsplit(path: string; var directory, name: string): boolean;`

**Description** Splits a path string returned by uri_split into its individual components for the file URI.

## 5.4 Constants

### URI_START_DELIMITER_CHAR

**Declaration** `URI_START_DELIMITER_CHAR = '<';`

**Description** Suggested start delimiter character for an URI, c.f. RFC 2396

### URI_END_DELIMITER_CHAR

**Declaration** `URI_END_DELIMITER_CHAR = '>';`

**Description** Suggested end delimiter character for an URI, c.f. RFC 2396

### URI_SCHEME_NAME_EMAIL

**Declaration** `URI_SCHEME_NAME_EMAIL = 'mailto';`

### URI_SCHEME_SEPARATOR

**Declaration** `URI_SCHEME_SEPARATOR = ':';`

## 5.5 Author

Carl Eric Codere

# Chapter 6

# Unit iso3166

## 6.1 Description

Country code unit

This unit is used to check the country codes as well as return information on the country, according to ISO 3166.
The lists were converted from the semicolon delimited version available here: http://www.iso.org/iso/en/prods-services/iso3166ma/

## 6.2 Overview

`isvalidcountrycode` Verifies if the 2 letter country code is valid

`getcountryname_fr`

`getcountryname_en`

## 6.3 Functions and Procedures

### isvalidcountrycode

**Declaration** `function isvalidcountrycode(s: shortstring): boolean;`

**Description** Verifies if the 2 letter country code is valid

This routine checks if the two letter country code is valid (as defined in ISO3166-1). The country code is not case sensitive.

**Parameters** **s** The three digit country code

**Returns** TRUE if the country code is valid, otherwise returns FALSE

### getcountryname_fr

**Declaration** `function getcountryname_fr(s: shortstring): shortstring;`

### getcountryname_en

**Declaration** `function getcountryname_en(s: shortstring): shortstring;`

## 6.4 Author

Carl Eric Codere

# Chapter 7

# Unit iso639

## 7.1 Description

Language code unit

This unit is used to check the language codes as well as return information on the country, according to ISO 639-1 and ISO 639-2.
The database was taken from the following site: http://www.loc.gov/standards/iso639-2/ISO-639-2_values_8bits.txt

## 7.2 Overview

`isvalidlangcode` Verifies if the 2 or 3 letter language code is valid

`getlangname_fr`

`getlangname_en`

`getlangcode_en`

`getlangcode_fr`

## 7.3 Functions and Procedures

### isvalidlangcode

**Declaration** `function isvalidlangcode(s: shortstring): boolean;`

**Description** Verifies if the 2 or 3 letter language code is valid

This routine checks if the two or three letter language code is valid (as defined in ISO 639, part 1 and part 2 respectively). The language code IS case sensitive and should be in lower case.

**Parameters** **s** The two or three digit language code

**Returns** TRUE if the language code is valid, otherwise returns FALSE

## getlangname_fr

**Declaration** `function getlangname_fr(s: shortstring): shortstring;`

**Description** This routine returns the language name in french for the specified language code. The language code IS case insensitive and can be either 2 or 3 characters in length (according to ISO 639-1 and ISO 639-2 respectively)

The returned string is encoded according to ISO-8859-1. If there are alternate names for the language, only the first alternate name is returned.

**Parameters** **s** The two or three digit language code

## getlangname_en

**Declaration** `function getlangname_en(s: shortstring): shortstring;`

**Description** This routine returns the language name in english for the specified language code. The language code IS case insensitive and can be either 2 or 3 characters in length (according to ISO 639-1 and ISO 639-2 respectively).

The returned string is encoded according to ISO-8859-1.If there are alternate names for the language, only the first alternate name is returned.

**Parameters** **s** The two or three digit language code

## getlangcode_en

**Declaration** `function getlangcode_en(name: shortstring): shortstring;`

**Description** This routine returns the 2 character code related to the english name of the language. The search is not case (according to ISO 639-1). If there is no 2 character language code for this language, or if the language name is not found, the routine returns an empty string.

The language name string should be encoded according to ISO-8859-1.

**Parameters** **name** The name of the language

**Returns** The 2 character language code

## getlangcode_fr

**Declaration** `function getlangcode_fr(name: shortstring): shortstring;`

**Description** This routine returns the 2 character code related to the french name of the language. The search is not case (according to ISO 639-1). If there is no 2 character language code for this language, or if the language name is not found, the routine returns an empty string.

The language name string should be encoded according to ISO-8859-1.

**Parameters** **name** The name of the language

**Returns** The 2 character language code

## 7.4 Author

Carl Eric Codere

# Chapter 8

# Unit locale

## 8.1 Description

Localisation unit

This unit is used to convert different locale information. ISO Standards are used where appropriate.
The exact representations that are supported are the following: Calendar Date: Complete Representation - basic Caldedar Date: Complete Representation - extended Calendar Date: Representations with reduced precision Time of the day: Local time of the day: Complete representation - basic Time of the day: Local time of the day: Complete representation - extended Time of the day: UTC Time : Complete representation - basic Time of the day: UTC Time: Complete representation - extended Time of the day: Local and UTC Time: extended format
Credits where credits are due, information on the ISO and date formats where taken from http://www.cl.cam.ac.uk/ mgk25/isc time.html

## 8.2 Overview

`GetISODateString`

`GetISODateStringBasic`

`IsValidISODateString` Verifies if the date is in a valid ISO 8601 format

`IsValidISOTimeString` Verifies if the time is in a valid ISO 8601 format

`IsValidISODateTimeString` Verifies if the date and time is in a valid ISO 8601 format

`GetISOTimeString`

`GetISOTimeStringBasic`

`GetISODateTimeString`

`UNIXToDateTime`

`GetCharEncoding`

`MicrosoftCodePageToMIMECharset`

`MicrosoftLangageCodeToISOCode`

## 8.3 Functions and Procedures

### GetISODateString

**Declaration** `function GetISODateString(Year, Month, Day: Word): shortstring;`

**Description** Returns the extended format representation of a date as recommended by ISO 8601 (Gregorian Calendar).

Returns an empty string if there is an error. The extended representation separates each member (year,month,day) with a dash character (-).

**Parameters** **year** Year of the date - valid values are from 0000 to 9999

**month** Month of the date - valid values are from 0 to 12

**day** Day of the month - valid values are from 1 to 31

### GetISODateStringBasic

**Declaration** `function GetISODateStringBasic(Year, Month, Day: Word): shortstring;`

**Description** Returns the basic format representation of a date as recommended by ISO 8601 (Gregorian Calendar).

Returns an empty string if there is an error.

**Parameters** **year** Year of the date - valid values are from 0000 to 9999

**month** Month of the date - valid values are from 0 to 12

**day** Day of the month - valid values are from 1 to 31

### IsValidISODateString

**Declaration** `function IsValidISODateString(datestr: shortstring; strict: boolean): boolean;`

**Description** Verifies if the date is in a valid ISO 8601 format

**Parameters** **datestr** Date string in valid ISO 8601 format

**strict** If set, the format must exactly be YYYYMMDD or YYYY-MM-DD. If not set, less precision is allowed

**Returns** TRUE if the date string is valid otherwise false

## IsValidISOTimeString

**Declaration** `function IsValidISOTimeString(timestr: shortstring; strict: boolean): boolean;`

**Description** Verifies if the time is in a valid ISO 8601 format

Currently does not support the fractional second parameters, and only the extemded time format recommended by W3C when used with the time zone designator.

**Parameters** **timestr** Time string in valid ISO 8601 format

**strict** The value must contain all the required parameters with UTC, either in basic or extended format to be valid

**Returns** TRUE if the time string is valid otherwise false

## IsValidISODateTimeString

**Declaration** `function IsValidISODateTimeString(str: shortstring; strict: boolean): boolean;`

**Description** Verifies if the date and time is in a valid ISO 8601 format

Currently does not support the fractional second parameters, and only the format recommended by W3C when used with the time zone designator. Also validates an entry if it only contains the date component (it is automatically detected).

**Parameters** **str** Date-Time string in valid ISO 8601 format

**strict** If set to TRUE then the complete representation must be present, either in basic or extended format to consider the date and time valid

**Returns** TRUE if the date-time string is valid otherwise false

## GetISOTimeString

**Declaration** `function GetISOTimeString(Hour, Minute, Second: Word; UTC: Boolean): shortstring;`

**Description** Returns the extended format representation of a time as recommended by ISO 8601 (Gregorian Calendar).

Returns an empty string if there is an error. The extended representation separates each member (hour,minute,second) with a colon (:).

## GetISOTimeStringBasic

**Declaration** `function GetISOTimeStringBasic(Hour, Minute, Second: Word; UTC: Boolean): shortstring;`

**Description** Returns the basic format representation of a time as recommended by ISO 8601 (Gregorian Calendar).

Returns an empty string if there is an error. The extended representation separates each member (hour,minute,second) with a colon (:).

### GetISODateTimeString

**Declaration** `function GetISODateTimeString(Year, Month, Day, Hour, Minute, Second: Word; UTC: Boolean): shortstring;`

### UNIXToDateTime

**Declaration** `procedure UNIXToDateTime(epoch: big_integer_t; var year, month, day, hour, minute, second: Word);`

**Description** Converts a UNIX styled time (the number of seconds since 1970) to a standard date and time representation.

### GetCharEncoding

**Declaration** `function GetCharEncoding(alias: string; var _name: string): integer;`

**Description** Using a registered ALIAS name for a specific character encoding, return the common or MIME name associated with this character set, and indicate the type of stream format used. The type of stream format used can be one of the `CHAR_ENCODING_XXXX` constants.

### MicrosoftCodePageToMIMECharset

**Declaration** `function MicrosoftCodePageToMIMECharset(cp: word): string;`

**Description** Using a code page identifier (as defined by Microsoft and OS/2) return the resulting IANA encoding alias string

### MicrosoftLangageCodeToISOCode

**Declaration** `function MicrosoftLangageCodeToISOCode(langcode: integer): string;`

**Description** Using a code page identifier (as defined by Microsoft and OS/2) return the resulting IANA encoding alias string

## 8.4 Constants

### CHAR_ENCODING_UTF8

**Declaration** `CHAR_ENCODING_UTF8 = 0;`

**Description** Character encoding value: UTF-8 storage format

### CHAR_ENCODING_UNKNOWN

**Declaration** `CHAR_ENCODING_UNKNOWN = -1;`

**Description** Character encoding value: unknown format

### CHAR_ENCODING_UTF32BE

**Declaration** `CHAR_ENCODING_UTF32BE = 1;`

**Description** Character encoding value: UTF-32 Big endian

### CHAR_ENCODING_UTF32LE

**Declaration** `CHAR_ENCODING_UTF32LE = 2;`

**Description** Character encoding value: UTF-32 Little endian

### CHAR_ENCODING_UTF16LE

**Declaration** `CHAR_ENCODING_UTF16LE = 3;`

**Description** Character encoding value: UTF-16 Little endian

### CHAR_ENCODING_UTF16BE

**Declaration** `CHAR_ENCODING_UTF16BE = 4;`

**Description** Character encoding value: UTF-16 Big endian

### CHAR_ENCODING_BYTE

**Declaration** `CHAR_ENCODING_BYTE = 5;`

**Description** Character encoding value: One byte per character storage format

### CHAR_ENCODING_UTF16

**Declaration** `CHAR_ENCODING_UTF16 = 6;`

**Description** Character encoding value: UTF-16 unknown endian (determined by BOM)

### CHAR_ENCODING_UTF32

**Declaration** `CHAR_ENCODING_UTF32 = 7;`

**Description** Character encoding value: UTF-32 unknown endian (determined by BOM)

## 8.5 Author

Carl Eric Codere

# Chapter 9

# Unit unicode

## 9.1 Description

unicode support unit

This unit contains routines to convert between the different unicode encoding schemes.
All UNICODE/ISO 10646 pascal styled strings are limited to MAX_STRING_LENGTH characters. Null terminated unicode strings are limited by the compiler and integer type size.
Since all these encoding are variable length, except the UCS-4 (which is equivalent to UTF-32 according to ISO 10646:2003) and UCS-2 encoding, to parse through characters, every string should be converted to UCS-4 or UCS-2 before being used.
The principal encoding scheme for this unit is UCS-4.

## 9.2 Overview

`ucs4_length` Returns the current length of an UCS-4 string

`ucs4_setlength` Set the new dynamic length of an UCS-4 string

`ucs4_iswhitespace` Determines if the specified character is a whitespace character

`ucs4_upcase` Converts a character to an uppercase character

`ucs4_lowcase` Converts a character to a lowercase character

`ucs4_trimleft` Trims leading spaces and control characters from an UCS-4 string.

`ucs4_trimright` Trims trailing spaces and control characters from an UCS-4 string.

`ucs4_trim` Trims trailing and leading spaces and control characters from an UCS-4 string.

`ucs4_copy` Returns an UCS-4 substring of an UCS-4 string

`ucs4_delete` Deletes a substring from a string

`ucs4_concat` Concatenates two UCS-4 strings, and gives a resulting UCS-4 string

`ucs4_removeaccents` Replaces all accented characters with their base representation

`ucs4_concatascii` Concatenates an UCS-4 string with an ASCII string, and gives a resulting UCS-4 string

`ucs4_posascii` Searches for an ASCII substring in an UCS-4 string

`ucs4_equalascii` Checks if an ASCII string is equal to an UCS-4 string

`ucs4_pos` Searches for an UCS-4 substring in an UCS-4 string

`ucs4_equal` Checks if both UCS-4 strings are equal

`ucs4_isvalid` Checks if the UCS-4 character is valid

`ucs4_issupported` Checks if conversion from/to this character set format to/from UCS-4 is supported

`ucs4_converttoiso8859_1` Converts an UCS-4 string to an ISO-8859-1 string

`ucs4_converttoutf8` Converts an UCS-4 string to an UTF-8 string

`ucs4strlen` Returns the number of characters in the null terminated UCS-4 string

`ucs4strpas` Converts a null-terminated UCS-4 string to a Pascal-style UCS-4 string.

`ucs4strpastoISO8859_1` Converts a null-terminated UCS-4 string to a Pascal-style ISO 8859-1 encoded string.

`ucs4strpastoASCII` Converts a null-terminated UCS-4 string to a Pascal-style ASCII encoded string.

`ucs4strpastoUTF8` Converts a null-terminated UCS-4 string to a Pascal-style UTF-8 encoded string.

`ucs4strpcopy` Copies a Pascal-style UCS-4 string to a null-terminated UCS-4 string.

`ucs4strnewstr` Converts a pascal string to an UCS-4 null terminated string

`ucs4strnew` Converts a null terminated string to an UCS-4 null terminated string

`ucs4strnewucs4` Allocates and copies an UCS-4 null terminated string

`ucs4strnewucs2` Converts an UCS-2 null terminated string to an UCS-4 null terminated string

`ucs4strdispose` Disposes of an UCS-4 null terminated string on the heap

`ucs4StrPosISO8859_1`

`ucs4strtrim`

`ucs4strfill`

`ucs4strcheck`

`ucs2_length` Returns the current length of an UCS-2 string

`ucs2_setlength` Set the new dynamic length of an ucs-2 string

`ucs2_isvalid` Checks if the UCS-2 character is valid

`ucs2_upcase` Converts a character to an uppercase character

`ucs2strlcopyucs4` Convert an UCS-2 null terminated string to an UCS-4 null terminated string

`ucs2strlen` Returns the number of characters in the null terminated UCS-2 string

`ucs2strnew` Converts an UCS-4 null terminated string to an UCS-2 null terminated string

`ucs2strdispose` Disposes of an UCS-2 null terminated string on the heap

`utf8strnew` Converts an UCS-4 null terminated string to an UTF-8 null terminated string

`utf8strlen` Returns the length of the string, not counting the null character

`utf8strnewutf8` Allocates and copies an UTF-8 null terminated string

`utf8strdispose` Disposes of an UTF-8 null terminated string on the heap

`utf8strlcopyucs4` Convert an UTF-8 null terminated string to an UCS-4 null terminated string

`utf8strpastoISO8859_1` Converts a null-terminated UTF-8 string to a Pascal-style ISO 8859-1 encoded string.

`utf8strpastoASCII` Converts a null-terminated UTF-8 string to a Pascal-style ASCII encoded string.

`utf8strpastostring` Converts an UTF-8 null terminated string to a string encoded to a different code page

`utf8strpas` Converts a null-terminated UTF-8 string to a Pascal-style UTF-8 encoded string.

`utf8strnewstr` Converts an UTF-8 string to a null terminated UTF-8 string.

`utf8stringdispose`

`utf8stringdup`

`asciistrpas` Converts a null-terminated ASCII string to a Pascal-style ASCII encoded string.

`asciistrnew` Allocates and copies an ascii null-terminated string from a null-terminated string

`asciistrnewstr`

`ansistrpas` Converts a null-terminated ISO-8859-1 string to a Pascal-style ASCII encoded string.

`ansistrnew` Allocates and copies an ISO-8859-1 null-terminated string from a null-terminated string

`ansistrnewstr`

`ansistrdispose`

`utf16_sizeencoding` Returns the number of characters that are used to encode this character

`utf8_sizeencoding` Returns the number of characters that are used to encode this character

`utf16_length` Returns the current length of an UTF-16 string

`utf8_length` Returns the current length of an UTF-8 string

`utf8_islegal` Returns if the specified UTF-8 string is legal or not

`utf8_setlength` Set the length of an UTF-8 string

`utf16_setlength` Set the length of an UTF-16 string

`convertUCS4toUTF8` Convert an UCS-4 string to an UTF-8 string

`ConvertFromUCS4` Convert an UCS-4 string to a single byte encoded string

`ConvertToUCS4` Convert a byte encoded string to an UCS-4 string

`ConvertUTF16ToUCS4` Convert an UTF-16 string to an UCS-4 string

`ConvertUCS4toUTF16` Convert an UCS-4 string to an UTF-16 string

`ConvertUTF8ToUCS4` Convert an UTF-8 string to an UCS-4 string

`ConvertUCS4ToUCS2` Convert an UCS-4 string to an UCS-2 string

`ConvertUCS2ToUCS4` Convert an UCS-2 string to an UCS-4 string

## 9.3 Functions and Procedures

### ucs4_length

**Declaration** `function ucs4_length(const s: array of ucs4char): integer;`

**Description** Returns the current length of an UCS-4 string

### ucs4_setlength

**Declaration** `procedure ucs4_setlength(var s: array of ucs4char; l: integer);`

**Description** Set the new dynamic length of an UCS-4 string

### ucs4_iswhitespace

**Declaration** `function ucs4_iswhitespace(c: ucs4char): boolean;`

**Description** Determines if the specified character is a whitespace character

### ucs4_upcase

**Declaration** `function ucs4_upcase(c: ucs4char): ucs4char;`

**Description** Converts a character to an uppercase character

This routine only supports the simple form case folding algorithm (e.g full form is not supported).

### ucs4_lowcase

**Declaration** `function ucs4_lowcase(c: ucs4char): ucs4char;`

**Description** Converts a character to a lowercase character

This routine only supports the simple form case folding algorithm (e.g full form is not supported).

### ucs4_trimleft

**Declaration** `procedure ucs4_trimleft(var s: ucs4string);`

**Description** Trims leading spaces and control characters from an UCS-4 string.

### ucs4_trimright

**Declaration** `procedure ucs4_trimright(var s: ucs4string);`

**Description** Trims trailing spaces and control characters from an UCS-4 string.

### ucs4_trim

**Declaration** `procedure ucs4_trim(var s: ucs4string);`

**Description** Trims trailing and leading spaces and control characters from an UCS-4 string.

### ucs4_copy

**Declaration** `procedure ucs4_copy(var resultstr: ucs4string; const s: array of ucs4char; index: integer; count: integer);`

**Description** Returns an UCS-4 substring of an UCS-4 string

### ucs4_delete

**Declaration** `procedure ucs4_delete(var s: ucs4string; index: integer; count: integer);`

**Description** Deletes a substring from a string

### ucs4_concat

**Declaration** `procedure ucs4_concat(var resultstr: ucs4string;s1: ucs4string; const s2: array of ucs4char);`

**Description** Concatenates two UCS-4 strings, and gives a resulting UCS-4 string

### ucs4_removeaccents

**Declaration** `procedure ucs4_removeaccents(var resultstr: ucs4string;s2: ucs4string);`

**Description** Replaces all accented characters with their base representation

. This routine is useful for converted multilangual strings to their ASCII equivalents.

### ucs4_concatascii

**Declaration** `procedure ucs4_concatascii(var resultstr: ucs4string;s1: ucs4string; s2: string);`

**Description** Concatenates an UCS-4 string with an ASCII string, and gives a resulting UCS-4 string

### ucs4_posascii

**Declaration** `function ucs4_posascii(substr: string; s: ucs4string): integer;`

**Description** Searches for an ASCII substring in an UCS-4 string

### ucs4_equalascii

**Declaration** `function ucs4_equalascii(s1 : array of ucs4char; s2: string): boolean;`

**Description** Checks if an ASCII string is equal to an UCS-4 string

### ucs4_pos

**Declaration** `function ucs4_pos(substr: ucs4string; const s: ucs4string): integer;`

**Description** Searches for an UCS-4 substring in an UCS-4 string

### ucs4_equal

**Declaration** `function ucs4_equal(const s1,s2: ucs4string): boolean;`

**Description** Checks if both UCS-4 strings are equal

### ucs4_isvalid

**Declaration** `function ucs4_isvalid(c: ucs4char): boolean;`

**Description** Checks if the UCS-4 character is valid

This routine verifies if the UCS-4 character is within the valid ranges of UCS-4 characters, as specified in the Unicode standard 4.0. BOM characters are NOT valid with this routine.

### ucs4_issupported

**Declaration** `function ucs4_issupported(s: string): boolean;`

**Description** Checks if conversion from/to this character set format to/from UCS-4 is supported

**Parameters** **s** This is an alias for a character set, as defined by IANA

**Returns** true if conversion to/from UCS-4 is supported with this character set, otherwise FALSE

### ucs4_converttoiso8859_1

**Declaration** `function ucs4_converttoiso8859_1(s: ucs4string): string;`

**Description** Converts an UCS-4 string to an ISO-8859-1 string

**Parameters** **s** The UCS-4 string to convert

**Returns** The converted string with possible escape characters if a character could not be converted

### ucs4_converttoutf8

**Declaration** `function ucs4_converttoutf8(src: ucs4string): utf8string;`

**Description** Converts an UCS-4 string to an UTF-8 string

If there is an error (such as reserved special characters), returns an empty string

**Parameters** **s** The UCS-4 string to convert

**Returns** The converted string

### ucs4strlen

**Declaration** `function ucs4strlen(str: pucs4char): integer;`

**Description** Returns the number of characters in the null terminated UCS-4 string

**Parameters** **str** The UCS-4 null terminated string to check

**Returns** The number of characters in str, not counting the null character

### ucs4strpas

**Declaration** `procedure ucs4strpas(str: pucs4char; var res:ucs4string);`

**Description** Converts a null-terminated UCS-4 string to a Pascal-style UCS-4 string.

### ucs4strpastoISO8859_1

**Declaration** `function ucs4strpastoISO8859_1(str: pucs4char): string;`

**Description** Converts a null-terminated UCS-4 string to a Pascal-style ISO 8859-1 encoded string.

If the length is greater than the supported maximum string length, the string is truncated.

Characters that cannot be converted are converted to escape sequences of the form : \uxxxxxxxx where xxxxxxxx is the hex representation of the character.

### ucs4strpastoASCII

**Declaration** `function ucs4strpastoASCII(str: pucs4char): string;`

**Description** Converts a null-terminated UCS-4 string to a Pascal-style ASCII encoded string.

If the length is greater than the supported maximum string length, the string is truncated.

Characters that cannot be converted are converted to escape sequences of the form : \uxxxxxxxx where xxxxxxxx is the hex representation of the character.

### ucs4strpastoUTF8

**Declaration** `function ucs4strpastoUTF8(str: pucs4char): utf8string;`

**Description** Converts a null-terminated UCS-4 string to a Pascal-style UTF-8 encoded string.

If the length is greater than the supported maximum string length, the string is truncated.

### ucs4strpcopy

**Declaration** `function ucs4strpcopy(dest: pucs4char; source: ucs4string):pucs4char;`

**Description** Copies a Pascal-style UCS-4 string to a null-terminated UCS-4 string.

This routine does not perform any length checking. If the source string contains some null characters, those nulls are removed from the resulting string.

The destination buffer must have room for at least Length(Source)+1 characters.

### ucs4strnewstr

**Declaration** `function ucs4strnewstr(str: string; srctype: string): pucs4char;`

**Description** Converts a pascal string to an UCS-4 null terminated string

The memory for the buffer is allocated. Use `ucs4strdispose`(9.3) to dispose of the allocated string. The string is null terminated. If the original string contains some null characters, those nulls are removed from the resulting string.

**Parameters** **str** The string to convert, single character coded

**srctype** The encoding of the string, UTF-8 is also valid - case-insensitive

### ucs4strnew

**Declaration** `function ucs4strnew(str: pchar; srctype: string): pucs4char;`

**Description** Converts a null terminated string to an UCS-4 null terminated string

The memory for the buffer is allocated. Use `ucs4strdispose`(9.3) to dispose of the allocated string. The string is null terminated. If str is nil, then this routine returns nil and does not allocate anything.

**Parameters** **str** The string to convert, single character coded, or UTF-8 coded

**srctype** The encoding of the string, UTF-8 is also valid - case-insensitive

### ucs4strnewucs4

**Declaration** `function ucs4strnewucs4(src: pucs4char): pucs4char;`

**Description** Allocates and copies an UCS-4 null terminated string

The memory for the buffer is allocated. Use `ucs4strdispose`(9.3) to dispose of the allocated string. The string is copied from src and is null terminated. If the parameter is nil, this routine returns nil and does not allocate anything.

### ucs4strnewucs2

**Declaration** `function ucs4strnewucs2(str: pucs2char): pucs4char;`

**Description** Converts an UCS-2 null terminated string to an UCS-4 null terminated string

The memory for the buffer is allocated. Use `ucs4strdispose`(9.3) to dispose of the allocated string. The string is null terminated. If str is nil, then this routine returns nil and does not allocate anything.

**Parameters** **str** The string to convert, UCS-2 encoded

### ucs4strdispose

**Declaration** `function ucs4strdispose(str: pucs4char): pucs4char;`

**Description** Disposes of an UCS-4 null terminated string on the heap

Disposes of a string that was previously allocated with `ucs4strnew`, and sets the pointer to nil.

### ucs4StrPosISO8859_1

**Declaration** `function ucs4StrPosISO8859_1(S: pucs4char; Str2: PChar): pucs4char;`

**Description** Returns a pointer to the first occurence of an ISO-8859-1 encoded null terminated string in a UCS-4 encoded null terminated string.

### ucs4strtrim

**Declaration** `function ucs4strtrim(const p: pucs4char): pucs4char;`

**Description** Allocates a new UCS-4 null terminated string, and copies the existing string, avoid a copy of the whitespace at the start and end of the string

### ucs4strfill

**Declaration** `function ucs4strfill(var p: pucs4char; count: integer; value: ucs4char): pucs4char;`

**Description** Fills a memory region consisting of ucs-4 characters with the specified UCS-4 character.

### ucs4strcheck

**Declaration** `procedure ucs4strcheck(p: pucs4char; maxcount: integer; value: ucs4char);`

**Description** This routine checks the validity of an UCS-4 null terminated string. It first skips to the null character, and if maxcount is greater than the index, verifies that the values in memory are of value VALUE.

Otherwise, returns an ERROR. This routine is used with UCS4StrFill.

### ucs2_length

**Declaration** `function ucs2_length(const s: array of ucs2char): integer;`

**Description** Returns the current length of an UCS-2 string

### ucs2_setlength

**Declaration** `procedure ucs2_setlength(var s: array of ucs2char; l: integer);`

**Description** Set the new dynamic length of an ucs-2 string

### ucs2_isvalid

**Declaration** `function ucs2_isvalid(ch: ucs2char): boolean;`

**Description** Checks if the UCS-2 character is valid

This routine verifies if the UCS-2 character is within the valid ranges of UCS-2 characters, as specified in the Unicode standard 4.0. BOM characters are NOT valid with this routine.

### ucs2_upcase

**Declaration** `function ucs2_upcase(c: ucs2char): ucs2char;`

**Description** Converts a character to an uppercase character

This routine only supports the simple form case folding algorithm (e.g full form is not supported).

### ucs2strlcopyucs4

**Declaration** `function ucs2strlcopyucs4(src: pucs2char; dst: pucs4char; maxlen: integer): pucs4char;`

**Description** Convert an UCS-2 null terminated string to an UCS-4 null terminated string

This routine converts an UCS-2 encoded null terminared string to an UCS-4 null terminated string that is stored in native byte order, up to length conversion. The destination buffer should already have been allocated.

**Returns** nil if there was an error in the conversion

### ucs2strlen

**Declaration** `function ucs2strlen(str: pucs2char): integer;`

**Description** Returns the number of characters in the null terminated UCS-2 string

**Parameters** **str** The UCS-2 null terminated string to check

**Returns** The number of characters in str, not counting the null character

### ucs2strnew

**Declaration** `function ucs2strnew(src: pucs4char): pucs2char;`

**Description** Converts an UCS-4 null terminated string to an UCS-2 null terminated string

The memory for the buffer is allocated. Use `ucs2strdispose`(9.3) to dispose of the allocated string. The string is null terminated. If src is nil, this routine returns nil, and does not allocate anything.

**Returns** nil if the conversion cannot be represented in UCS-2 encoding, or nil if there was an error

### ucs2strdispose

**Declaration** `function ucs2strdispose(str: pucs2char): pucs2char;`

**Description** Disposes of an UCS-2 null terminated string on the heap

Disposes of a string that was previously allocated with `ucs2strnew`, and sets the pointer to nil.

### utf8strnew

**Declaration** `function utf8strnew(src: pucs4char): pchar;`

**Description** Converts an UCS-4 null terminated string to an UTF-8 null terminated string

The memory for the buffer is allocated. Use `utf8strdispose`(9.3) to dispose of the allocated string. The string is null terminated. If the parameter is nil, this routine returns nil and does not allocate anything.

### utf8strlen

**Declaration** `function utf8strlen(s: putf8char): integer;`

**Description** Returns the length of the string, not counting the null character

If the pointer is nil, the value returned is zero.

### utf8strnewutf8

**Declaration** `function utf8strnewutf8(src: pchar): pchar;`

**Description** Allocates and copies an UTF-8 null terminated string

The memory for the buffer is allocated. Use `utf8strdispose`(9.3) to dispose of the allocated string. The string is copied from src and is null terminated. If the parameter is nil, this routine returns nil and does not allocate anything.

### utf8strdispose

**Declaration** `function utf8strdispose(p: pchar): pchar;`

**Description** Disposes of an UTF-8 null terminated string on the heap

Disposes of a string that was previously allocated with `utf8strnew`, and sets the pointer to nil.

## utf8strlcopyucs4

**Declaration** `function utf8strlcopyucs4(src: pchar; dst: pucs4char; maxlen: integer): pucs4char;`

**Description** Convert an UTF-8 null terminated string to an UCS-4 null terminated string

This routine converts an UTF-8 null terminared string to an UCS-4 null terminated string that is stored in native byte order, up to length conversion.

**Returns** nil if there was no error in the conversion

## utf8strpastoISO8859_1

**Declaration** `function utf8strpastoISO8859_1(src: pchar): string;`

**Description** Converts a null-terminated UTF-8 string to a Pascal-style ISO 8859-1 encoded string.

Characters that cannot be converted are converted to escape sequences of the form : \uxxxxxxxx where xxxxxxxx is the hex representation of the character.

## utf8strpastoASCII

**Declaration** `function utf8strpastoASCII(src: pchar): string;`

**Description** Converts a null-terminated UTF-8 string to a Pascal-style ASCII encoded string.

Characters that cannot be converted are converted to escape sequences of the form : \uxxxxxxxx where xxxxxxxx is the hex representation of the character.

## utf8strpastostring

**Declaration** `function utf8strpastostring(src: pchar; desttype: string): string;`

**Description** Converts an UTF-8 null terminated string to a string encoded to a different code page

Characters that cannot be converted are converted to escape sequences of the form : \uxxxxxxxx where xxxxxxxx is the hex representation of the character.

If the null character string does not fit in the resulting string, it is truncated.

.

**Parameters** **src** Null terminated UTF-8 encoded string

**desttype** Encoding type for resulting string

**Returns** an empty string on error, or a correctly encoded string

### utf8strpas

**Declaration** `function utf8strpas(src: pchar): string;`

**Description** Converts a null-terminated UTF-8 string to a Pascal-style UTF-8 encoded string.

The string is empty if the pointer was nil.

### utf8strnewstr

**Declaration** `function utf8strnewstr(str: utf8string): putf8char;`

**Description** Converts an UTF-8 string to a null terminated UTF-8 string.

The memory for the storage of the string is allocated by the routine, and the ending null character is also added.

**Returns** The newly allocated UTF-8 null terminated string

### utf8stringdispose

**Declaration** `procedure utf8stringdispose(var p : putf8shortstring);`

### utf8stringdup

**Declaration** `function utf8stringdup(const s : string) : putf8shortstring;`

### asciistrpas

**Declaration** `function asciistrpas(src: pchar): string;`

**Description** Converts a null-terminated ASCII string to a Pascal-style ASCII encoded string.

The returned string shall be empty if the pointer was nil.

### asciistrnew

**Declaration** `function asciistrnew(src: pchar): pchar;`

**Description** Allocates and copies an ascii null-terminated string from a null-terminated string

The value returned is nil, if the src pointer was also nil.

### asciistrnewstr

**Declaration** `function asciistrnewstr(const str: string): pchar;`

### ansistrpas

**Declaration** `function ansistrpas(src: pchar): string;`

**Description** Converts a null-terminated ISO-8859-1 string to a Pascal-style ASCII encoded string.

The returned string shall be empty if the pointer was nil.

### ansistrnew

**Declaration** `function ansistrnew(src: pchar): pchar;`

**Description** Allocates and copies an ISO-8859-1 null-terminated string from a null-terminated string

The value returned is nil, if the src pointer was also nil.

### ansistrnewstr

**Declaration** `function ansistrnewstr(const str: string): pchar;`

### ansistrdispose

**Declaration** `function ansistrdispose(p: pchar): pchar;`

### utf16_sizeencoding

**Declaration** `function utf16_sizeencoding(c: utf16char): integer;`

**Description** Returns the number of characters that are used to encode this character

Actually checks if this is a high-surrogate value, if not returns 1, indicating that the character is encoded a single `utf16` character, otherwise returns 2, indicating that 1 one other `utf16` character is required to encode this data.

### utf8_sizeencoding

**Declaration** `function utf8_sizeencoding(c: utf8char): integer;`

**Description** Returns the number of characters that are used to encode this character

### utf16_length

**Declaration** `function utf16_length(const s: array of utf16char): integer;`

**Description** Returns the current length of an UTF-16 string

### utf8_length

**Declaration** `function utf8_length(const s: utf8string): integer;`

**Description** Returns the current length of an UTF-8 string

### utf8_islegal

**Declaration** `function utf8_islegal(const s: utf8string): boolean;`

**Description** Returns if the specified UTF-8 string is legal or not

Verifies that the UTF-8 encoded strings is encoded in a legal way.

**Returns** FALSE if the string is illegal, otherwise returns TRUE

### utf8_setlength

**Declaration** `procedure utf8_setlength(var s: utf8string; l: integer);`

**Description** Set the length of an UTF-8 string

### utf16_setlength

**Declaration** `procedure utf16_setlength(var s: array of utf16char; l: integer);`

**Description** Set the length of an UTF-16 string

### convertUCS4toUTF8

**Declaration** `function convertUCS4toUTF8(s: array of ucs4char; var outstr: utf8string): integer;`

**Description** Convert an UCS-4 string to an UTF-8 string

Converts an UCS-4 string or character in native endian to an UTF-8 string.

**Parameters** **s** Either a single UCS-4 character or a complete UCS-4 string

**outstr** Resulting UTF-8 coded string

**Returns** `UNICODE_ERR_OK`(9.5) if there was no error in the conversion

### ConvertFromUCS4

**Declaration** `function ConvertFromUCS4(const source: ucs4string; var dest: string; desttype: string): integer;`

**Description** Convert an UCS-4 string to a single byte encoded string

This routine converts an UCS-4 string stored in native byte order (native endian) to a single-byte encoded string.

The string is limited to MAX_STRING_LENGTH characters, and if the conversion cannot be successfully be completed, it gives out an error.

The following `desttype` can be specified: ISO-8859-1, windows-1252, ISO-8859-2, ISO-8859-5, ISO-8859-16, macintosh, atari, cp437, cp850, ASCII and UTF-8.

**Parameters** **desttype** Indicates the single byte encoding scheme - case-insensitive

**Returns** `UNICODE_ERR_OK`(9.5) if there was no error in the conversion

## ConvertToUCS4

**Declaration** `function ConvertToUCS4(source: string; var dest: ucs4string; srctype: string): integer;`

**Description** Convert a byte encoded string to an UCS-4 string

This routine converts a single byte encoded string to an UCS-4 string stored in native byte order

Characters that cannot be converted are converted to escape sequences of the form : \uxxxxxxxx where xxxxxxxx is the hex representation of the character, an error code will also be returned by the function

The string is limited to MAX_STRING_LENGTH characters, and if the conversion cannot be successfully be completed, it gives out an error. The following `srctype` can be specified: ISO-8859-1, windows-1252, ISO-8859-2, ISO-8859-5, ISO-8859-16, macintosh, atari, cp437, cp850, ASCII.

**Parameters** **srctype** Indicates the single byte encoding scheme - case-insensitive

**Returns** `UNICODE_ERR_OK`(9.5) if there was no error in the conversion

## ConvertUTF16ToUCS4

**Declaration** `function ConvertUTF16ToUCS4(src: utf16string; var dst: ucs4string): integer;`

**Description** Convert an UTF-16 string to an UCS-4 string

This routine converts an UTF-16 string to an UCS-4 string. Both strings must be stored in native byte order (native endian).

**Returns** `UNICODE_ERR_OK`(9.5) if there was no error in the conversion

## ConvertUCS4toUTF16

**Declaration** `function ConvertUCS4toUTF16(src: array of ucs4char; var dest: utf16string): integer;`

**Description** Convert an UCS-4 string to an UTF-16 string

This routine converts an UCS-4 string to an UTF-16 string. Both strings must be stored in native byte order (native endian).

**Parameters** **src** Either a single UCS-4 character or a complete UCS-4 string

**dest** Resulting UTF-16 coded string

**Returns** `UNICODE_ERR_OK`(9.5) if there was no error in the conversion

### ConvertUTF8ToUCS4

**Declaration** `function ConvertUTF8ToUCS4(src: utf8string; var dst: ucs4string): integer;`

**Description** Convert an UTF-8 string to an UCS-4 string

This routine converts an UTF-8 string to an UCS-4 string that is stored in native byte order.

**Returns** `UNICODE_ERR_OK`(9.5) if there was no error in the conversion

### ConvertUCS4ToUCS2

**Declaration** `function ConvertUCS4ToUCS2(src: array of ucs4char; var dst: ucs2string): integer;`

**Description** Convert an UCS-4 string to an UCS-2 string

This routine converts an UCS-4 string to an UCS-2 string that is stored in native byte order. If some characters could not be converted an error will be reported.

**Parameters** **src** Either a single UCS-4 character or a complete UCS-4 string

**dest** Resulting UCS-2 coded string

**Returns** `UNICODE_ERR_OK`(9.5) if there was no error in the conversion

### ConvertUCS2ToUCS4

**Declaration** `function ConvertUCS2ToUCS4(src: array of ucs2char; var dst: ucs4string): integer;`

**Description** Convert an UCS-2 string to an UCS-4 string

This routine converts an UCS-2 string to an UCS-4 string that is stored in native byte order (big-endian). If some characters could not be converted an error will be reported.

**Parameters** **src** Either a single ucs-2 character or a complete ucs-2 string

**dest** Resulting UCS-4 coded string

**Returns** `UNICODE_ERR_OK`(9.5) if there was no error in the conversion

## 9.4 Types

### utf8char

**Declaration** `utf8char = char;`

**Description** UTF-8 base data type

### putf8char

**Declaration** `putf8char = pchar;`

### utf16char

**Declaration** `utf16char = word;`

**Description** UTF-16 base data type

### ucs4char

**Declaration** `ucs4char = longword;`

**Description** UCS-4 base data type

### pucs4char

**Declaration** `pucs4char = ^ucs4char;`

**Description** UCS-4 null terminated string

### ucs2char

**Declaration** `ucs2char = word;`

**Description** UCS-2 base data type

### pucs2char

**Declaration** `pucs2char = ^ucs2char;`

### ucs2string

**Declaration** `ucs2string = array[0..MAX_STRING_LENGTH] of ucs2char;`

**Description** UCS-2 string declaration. Index 0 contains the active length of the string in characters.

### ucs4string

**Declaration** `ucs4string = array[0..MAX_STRING_LENGTH] of ucs4char;`

**Description** UCS-4 string declaration. Index 0 contains the active length of the string in characters.

### utf8string

**Declaration** `utf8string = string;`

**Description** UTF-8 string declaration. This can either map to a short string or an ansi string depending on the compilation options.

### putf8shortstring

**Declaration** `putf8shortstring = ^shortstring;`

**Description** UTF-8 string pointer declaration. This is always a shortstring value.

### utf16string

**Declaration** `utf16string = array[0..MAX_STRING_LENGTH] of utf16char;`

**Description** UTF-16 string declaration. Index 0 contains the active length of the string in BYTES

### ucs4strarray

**Declaration** `ucs4strarray = array[0..MAX_UCS4_CHARS] of ucs4char;`

### pucs4strarray

**Declaration** `pucs4strarray = ^ucs4strarray;`

### ucs2strarray

**Declaration** `ucs2strarray = array[0..MAX_UCS2_CHARS] of ucs2char;`

### pucs2strarray

**Declaration** `pucs2strarray = ^ucs2strarray;`

## 9.5 Constants

### MAX_STRING_LENGTH

**Declaration** `MAX_STRING_LENGTH = 1024;`

**Description** Gives the number of characters that can be contained in a string

### MAX_UCS4_CHARS

**Declaration** `MAX_UCS4_CHARS = high(smallint) div (sizeof(ucs4char));`

**Description** Maximum size of a null-terminated UCS-4 character string

### MAX_UCS2_CHARS

**Declaration** `MAX_UCS2_CHARS = high(smallint) div (sizeof(ucs2char))-1;`

**Description** Maximum size of a null-terminated UCS-4 character string

## UNICODE_ERR_OK

**Declaration** `UNICODE_ERR_OK = 0;`

**Description** Return status: conversion successful

## UNICODE_ERR_SOURCEILLEGAL

**Declaration** `UNICODE_ERR_SOURCEILLEGAL = -1;`

**Description** Return status: source sequence is illegal/malformed

## UNICODE_ERR_LENGTH_EXCEED

**Declaration** `UNICODE_ERR_LENGTH_EXCEED = -2;`

**Description** Return status: Target space excedeed

## UNICODE_ERR_INCOMPLETE_CONVERSION

**Declaration** `UNICODE_ERR_INCOMPLETE_CONVERSION = -3;`

**Description** Return status: Some charactrer could not be successfully converted to this format

## UNICODE_ERR_NOTFOUND

**Declaration** `UNICODE_ERR_NOTFOUND = -4;`

**Description** Return status: The character set is not found

## BOM_UTF8

**Declaration** `BOM_UTF8 = #$EF#$BB#$BF;`

**Description** Byte order mark: UTF-8 encoding signature

## BOM_UTF32_BE

**Declaration** `BOM_UTF32_BE = #00#00#$FE#$FF;`

**Description** Byte order mark: UCS-4 big endian encoding signature

## BOM_UTF32_LE

**Declaration** `BOM_UTF32_LE = #$FF#$FE#00#00;`

**Description** Byte order mark: UCS-4 little endian encoding signature

## BOM_UTF16_BE

**Declaration** `BOM_UTF16_BE = #$FE#$FF;`

### BOM_UTF16_LE

**Declaration** `BOM_UTF16_LE = #$FF#$FE;`

## 9.6 Author

Carl Eric Codre

# Chapter 10

# Unit utils

## 10.1 Description

General utilities common to all platforms.

## 10.2 Overview

`AnsiFileExists` Verifies the existence of a filename

`AnsiDirectoryExists` Verifies the existence of a directory

`SwapLong` Change the endian of a 32-bit value

`SwapWord` Change the endian of a 16-bit value

`UpString` Convert a string to uppercase ASCII

`LowString` Convert a string to lowercase ASCII

`AddDoubleQuotes`

`RemoveDoubleQuotes`

`StreamErrorProcedure` Generic stream error procedure

`TrimLeft` Remove all whitespace from the start of a string

`TrimRight` Remove all whitespace from the end of a string

`hexstr` Convert a value to an ASCII hexadecimal representation

`decstr` Convert a value to an ASCII decimal representation

`decstrunsigned` Convert a value to an ASCII decimal representation

`boolstr` Convert a boolean value to an ASCII representation

`CompareByte`

`Trim` Trim removes leading and trailing spaces and control characters from the given string S

`Printf` Format a string and print it out to the console

`FillTo`

`stringdup`

`stringdispose`

`EscapeToPascal`

`ValDecimal`

`ValOctal`

`ValBinary`

`ValHexadecimal`

`CleanString`

`ChangeFileExt`

`fillwithzero`

`removenulls`

## 10.3 Functions and Procedures

### AnsiFileExists

**Declaration** `Function AnsiFileExists(const FName : string): Boolean;`

**Description** Verifies the existence of a filename

This routine verifies if the file named can be opened or if it actually exists.
Only works with the current active code page table.

**Parameters** **FName** Name of the file to check

**Returns** FALSE if the file cannot be opened or if it does not exist.

## AnsiDirectoryExists

**Declaration** `Function AnsiDirectoryExists(DName : string): Boolean;`

**Description** Verifies the existence of a directory

This routine verifies if the directory named can be opened or if it actually exists.
Only works with the current active code page table.

**Parameters** **DName** Name of the directory to check

**Returns** FALSE if the directory cannot be opened or if it does not exist.

## SwapLong

**Declaration** `Procedure SwapLong(var x : longword);`

**Description** Change the endian of a 32-bit value

## SwapWord

**Declaration** `Procedure SwapWord(var x : word);`

**Description** Change the endian of a 16-bit value

## UpString

**Declaration** `function UpString(s : string): string;`

**Description** Convert a string to uppercase ASCII

Converts a string containing ASCII characters to a string in upper case ASCII characters.

## LowString

**Declaration** `function LowString(s : string): string;`

**Description** Convert a string to lowercase ASCII

Converts a string containing ASCII characters to a string in lower case ASCII characters.

## AddDoubleQuotes

**Declaration** `function AddDoubleQuotes(s: string): string;`

**Description** This routine adds double quotes to the selected string, if the string contains any space character after the string has been trimmed.

## RemoveDoubleQuotes

**Declaration** `function RemoveDoubleQuotes(s: string): string;`

**Description** This routine removes double quotes to the selected string (the leading and ending double quotes only)

## StreamErrorProcedure

**Declaration** `procedure StreamErrorProcedure(Var S: TStream);`

**Description** Generic stream error procedure

Generic stream error procedure that can be used to set `streamerror`

## TrimLeft

**Declaration** `function TrimLeft(const S: string): string;`

**Description** Remove all whitespace from the start of a string

## TrimRight

**Declaration** `function TrimRight(const S: string): string;`

**Description** Remove all whitespace from the end of a string

## hexstr

**Declaration** `function hexstr(val : longint;cnt : byte) : string;`

**Description** Convert a value to an ASCII hexadecimal representation

Convert a value to an ASCII hexadecimal representation. All ascii character are returned in upper case letters.

## decstr

**Declaration** `function decstr(val : longint;cnt : byte) : string;`

**Description** Convert a value to an ASCII decimal representation

To avoid left padding with zeros, set `cnt` to zero.

**Parameters** **val** Signed 32-bit value to convert

## decstrunsigned

**Declaration** `function decstrunsigned(l : longword;cnt: byte): string;`

**Description** Convert a value to an ASCII decimal representation

To avoid left padding with zeros, set `cnt` to zero.

**Parameters** **val** unsigned 32-bit value to convert

## boolstr

**Declaration** `function boolstr(val: boolean; cnt: byte): string;`

**Description** Convert a boolean value to an ASCII representation

To avoid left padding with spaces, set `cnt` to zero.

## CompareByte

**Declaration** `function CompareByte(buf1,buf2: pchar;len:longint):integer;`

## Trim

**Declaration** `function Trim(const S: string): string;`

**Description** Trim removes leading and trailing spaces and control characters from the given string S

## Printf

**Declaration** `function Printf(const s : string; var Buf; size : word): string;`

**Description** Format a string and print it out to the console

This routine formats the string specified in s to the format specified and returns the resulting string.

The following specifiers are allowed: %d : The buffer contents contains an integer %s : The buffer contents contains a string, terminated by a null character. %bh : The buffer contents contains a byte coded in BCD format, only the high byte will be kept. %bl : The buffer contents contains a byte coded in BCD format, only the low byte will be kept.

**Parameters** **s** The string to format, with format specifiers

**buf** The buffer containing the data

**size** The size of the data in the buffer

**Returns** The resulting formatted string

## FillTo

**Declaration** `function FillTo(s : string; tolength: integer): string;`

### stringdup

**Declaration** `function stringdup(const s :  string) :  pShortstring;`

### stringdispose

**Declaration** `procedure stringdispose(var p :  pShortstring);`

### EscapeToPascal

**Declaration** `function EscapeToPascal(const s:string; var code:  integer):  string;`

**Description** Converts a C style string (containing escape characters), to a pascal style string. Returns the converted string. If there is no error in the conversion, `code` will be equal to zero.

**Parameters** **s** String to convert

**code** Result of operation, 0 when there is no error

### ValDecimal

**Declaration** `function ValDecimal(const S:String; var code:  integer):longint;`

**Description** Convert a decimal value represented by a string to its numerical value. If there is no error, `code` will be equal to zero.

### ValOctal

**Declaration** `function ValOctal(const S:String;var code:  integer):longint;`

**Description** Convert an octal value represented by a string to its numerical value. If there is no error, `code` will be equal to zero.

### ValBinary

**Declaration** `function ValBinary(const S:String; var code:  integer):longint;`

**Description** Convert a binary value represented by a string to its numerical value. If there is no error, `code` will be equal to zero.

### ValHexadecimal

**Declaration** `function ValHexadecimal(const S:String; var code:  integer):longint;`

**Description** Convert an hexadecimal value represented by a string to its numerical value. If there is no error, `code` will be equal to zero.

### CleanString

**Declaration** `function CleanString(const s: string): string;`

**Description** Cleans up a string of illegal control characters, newlines and tabs. It does the following on the string:

This only works on UTF-8/ASCII/ISO-8859 strings.

1) Removes characters from code #0..#31 at any location in the string. 2) Trims spaces at the begining and end of the string.

### ChangeFileExt

**Declaration** `function ChangeFileExt(const FileName, Extension: string): string;`

### fillwithzero

**Declaration** `function fillwithzero(s: string; newlength: integer): string;`

### removenulls

**Declaration** `function removenulls(const s: string): string;`

**Description** Remove all null characters from a string.

## 10.4 Types

### PshortString

**Declaration** `PshortString = ^ShortString;`

## 10.5 Constants

### EXIT_DOSERROR

**Declaration** `EXIT_DOSERROR = 2;`

### EXIT_ERROR

**Declaration** `EXIT_ERROR = 1;`

### WhiteSpace

**Declaration** `WhiteSpace = [' ',#10,#13,#9];`

## 10.6 Author

Carl Eric Codere