Common pascal units documentation

Pasdoc

November 22, 2004

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

`TExtendedCollection Object` Base collection object

`TExtendedSortedCollection Object` Base sorted collection object

`TExtendedStringCollection Object` String pointer collection object

`TExtendedSortedStringCollection Object` Sorted string pointer collection object

1.3 Classes, Interfaces and Objects

TStack Object

Hierarchy

TStack > TObject

TExtendedCollection Object

Hierarchy

TExtendedCollection > TObject

Description

Base collection object

TExtendedSortedCollection Object

Hierarchy

TExtendedSortedCollection > `TExtendedCollection`(1.3) > TObject

Description

Base sorted collection object

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.

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.

1.4 Types

PStack

Declaration `PStack = ^TStack;`

Description Stack object

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

1.5 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

`UpdateAdler32` Calculates an Adler-32 checksum value

`UpdateCRC` Calculates a standard 16-bit CRC

`UpdateCrc16` Calculates a CRC-16 CCITT value

`UpdateCrc32` Calculates a CRC-32 CCITT value

`UpdateFletcher8` Calculates an 8-bit fletcher checksum value

2.3 Functions and Procedures

UpdateAdler32 function

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

UpdateCRC function

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

UpdateCrc16 function

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

UpdateCrc32 function

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

UpdateFletcher8 function

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

2.4 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.
All dates are assumed to be in Gregorian calendar date format (This is a proleptic Gregorian calendar unit).

3.2 Overview

`CurrentYear`

`Date`

`DateOf`

`DateTimeToStr`

`DateToStr`

`DayOf`

`DaysBetween`

`DecodeDate`

`DecodeDateTime`

`DecodeTime`

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

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

`GetTime`

`HourOf`

`IncDay`

`IncHour`

`IncMilliSecond`

`IncMinute`

`IncSecond`

`IncWeek`

`IsPM`

`IsValidDate`

`IsValidDateTime`

`IsValidTime`

`MinuteOf`

`MonthOf`

`Now`

`SameDate`

`SameDateTime`

`SameTime`

`SecondOf`

`Time`

`TimeOf`

`TimeToStr`

`Today`

`TryEncodeDate`

`TryEncodeDateAndTimeToStr`

`TryEncodeDateTime`

TryEncodeTime

TryStrToDate

TryStrToDateTime

TryStrToDateTimeExt

TryStrToTime

YearOf

3.3 Functions and Procedures

CurrentYear function

Declaration `function CurrentYear: word;`

Description Returns the current year

Date function

Declaration `function Date: TDatetime;`

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

DateOf function

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

Description Strips the time portion from a TDateTime value.

DateTimeToStr function

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

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

DateToStr function

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

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

DayOf function

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

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

DaysBetween function

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

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

DecodeDate procedure

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

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

DecodeDateTime procedure

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 procedure

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

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

GetCurrentDate procedure

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

Description Returns the current date set in the operating system

GetCurrentTime procedure

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.

GetTime function

Declaration `function GetTime: TDateTime;`

Description Returns the current time.

HourOf function

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

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

IncDay function

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

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

IncHour function

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

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

IncMilliSecond function

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 function

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 function

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 function

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

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

IsPM function

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

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

IsValidDate function

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

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

IsValidDateTime function

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 function

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 function

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

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

MonthOf function

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

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

Now function

Declaration `function Now: TDateTime;`

Description Returns the current date and time.

SameDate function

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

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

SameDateTime function

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 function

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 function

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

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

Time function

Declaration `function Time: TDateTime;`

Description Returns the current time.

TimeOf function

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

Description Strips the date portion from a TDatetime value

TimeToStr function

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

Description Returns a string that represents a TDateTime value.

Today function

Declaration `function Today: TDateTime;`

Description Returns a TDateTime value that represents the current date.

TryEncodeDate function

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

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

TryEncodeDateAndTimeToStr function

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.

TryEncodeDateTime function

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.

TryEncodeTime function

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.

TryStrToDate function

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 function

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.

TryStrToDateTimeExt function

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.

TryStrToTime function

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 time-zone 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 function

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

Description Returns the year represented by a TDateTime value.

3.4 Types

TDatetime

Declaration `TDatetime = real;`

Description This is the Julian Day number

3.5 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

`FileBlockRead` Read data from a file

`FileBlockWrite` Write data to a file

`FileClose` Close a previously opened file

`FileGetPos` Return the current file pointer position

`FileGetSize` Get the size of a file in bytes

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

`FileReset` Open a file for reading or writing

`FileRewrite` Open/Overwrite a file

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

`FileTruncate` Truncate a file at the current file position

4.3 Functions and Procedures

FileAssign procedure

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.

FileBlockRead function

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 function

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.

FileClose procedure

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.

FileGetPos function

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.

FileGetSize function

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.

FileIOResult function

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.

FileReset procedure

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 procedure

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`.

FileSeek procedure

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).

FileTruncate procedure

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

`urn_isvalid` Verifies the validity of a complete URN string

`urn_isvalidnid`

`urn_split` Splits an URN string in its separate components

5.3 Functions and Procedures

urn_isvalid function

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 function

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 function

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

5.4 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

6.3 Functions and Procedures

isvalidcountrycode function

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

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

`getlangcode_en`

`getlangcode_fr`

`getlangname_en`

`getlangname_fr`

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

7.3 Functions and Procedures

getlangcode_en function

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 function

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

getlangname_en function

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

getlangname_fr function

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

isvalidlangcode function

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

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

`GetCharEncoding`

`GetISODateString`

`GetISODateStringBasic`

`GetISOTimeString`

`GetISOTimeStringBasic`

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

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

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

`UNIXToDateTime`

8.3 Functions and Procedures

GetCharEncoding function

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.

GetISODateString function

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 function

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

GetISOTimeString function

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 function

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 (:).

IsValidISODateString function

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

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

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

Returns TRUE if the date string is valid otherwise false

IsValidISODateTimeString function

Declaration `function IsValidISODateTimeString(str: shortstring): 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

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

IsValidISOTimeString function

Declaration `function IsValidISOTimeString(timestr: shortstring): 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

Returns TRUE if the time string is valid otherwise false

UNIXToDateTime procedure

Declaration `procedure UNIXToDateTime(epoch: longword; 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.

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

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

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

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

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

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

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

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

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

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

`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

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

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

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

`ucs4strcheck`

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

`ucs4strfill`

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

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

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

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

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

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

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

`ucs4StrPosISO8859_1`

`ucs4strtrim`

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

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

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

`ucs4_delete` Deletes a substring from a string

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

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

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

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

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

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

`ucs4_lowcase` Converts a character to a lowercase character

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

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

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

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

`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_upcase` Converts a character to an uppercase character

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

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

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

`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

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

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

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

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

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

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

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

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

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

9.3 Functions and Procedures

ConvertFromUCS4 function

Declaration `function ConvertFromUCS4(const source: ucs4string; var dest: string; const 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.

Parameters **desttype** Indicates the single byte encoding scheme

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

ConvertToUCS4 function

Declaration `function ConvertToUCS4(source: string; var dest: ucs4string; const 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

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

ConvertUCS2ToUCS4 function

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

ConvertUCS4ToUCS2 function

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

ConvertUCS4toUTF16 function

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

convertUCS4toUTF8 function

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

ConvertUTF16ToUCS4 function

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

ConvertUTF8ToUCS4 function

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

ucs2strdispose function

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.

ucs2strlcopyucs4 function

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 function

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 function

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

ucs2_isvalid function

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_length function

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

Description Returns the current length of an UCS-2 string

ucs2_setlength procedure

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

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

ucs4strcheck procedure

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.

ucs4strdispose function

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.

ucs4strfill function

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.

ucs4strlen function

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

ucs4strnew function

Declaration `function ucs4strnew(str: pchar; const 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

ucs4strnewstr function

Declaration `function ucs4strnewstr(str: string; const 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

ucs4strnewucs4 function

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.

ucs4strpas procedure

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

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

ucs4strpastoASCII function

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.

ucs4strpastoISO8859_1 function

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.

ucs4strpastoUTF8 function

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 function

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.

ucs4StrPosISO8859_1 function

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 function

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

ucs4_concat procedure

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_concatascii procedure

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_copy procedure

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 procedure

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

Description Deletes a substring from a string

ucs4_equal function

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

Description Checks if both UCS-4 strings are equal

ucs4_equalascii function

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_issupported function

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_isvalid function

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_iswhitespace function

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

Description Determines if the specified character is a whitespace character

ucs4_length function

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

Description Returns the current length of an UCS-4 string

ucs4_lowcase function

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_pos function

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

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

ucs4_posascii function

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

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

ucs4_setlength procedure

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

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

ucs4_trim procedure

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

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

ucs4_trimleft procedure

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

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

ucs4_trimright procedure

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

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

ucs4_upcase function

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).

utf16_length function

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

Description Returns the current length of an UTF-16 string

utf16_setlength procedure

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

Description Set the length of an UTF-16 string

utf16_sizeencoding function

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.

utf8strdispose function

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 function

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

utf8strnew function

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.

utf8strnewutf8 function

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.

utf8strpas function

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

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

utf8strpastoASCII function

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.

utf8strpastoISO8859_1 function

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.

utf8_islegal function

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_length function

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

Description Returns the current length of an UTF-8 string

utf8_setlength procedure

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

Description Set the length of an UTF-8 string

utf8_sizeencoding function

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

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

.

9.4 Types

utf8char

Declaration `utf8char = char;`

Description UTF-8 base data type

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

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 = shortstring;`

Description UTF-8 string declaration. Index 0 contains the active length of the string in BYTES

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

9.5 Constants

MAX_STRING_LENGTH

Declaration `MAX_STRING_LENGTH = 255;`

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

9.6 Author

Carl Eric Codère

Chapter 10

Unit utils

10.1 Description

General utilities common to all platforms.

10.2 Overview

`boolstr` Convert a boolean value to an ASCII representation

`decstr` Convert a value to an ASCII decimal representation

`decstrunsigned` Convert a value to an ASCII decimal representation

`DirectoryExists` Verifies the existence of a directory

`EscapeToPascal`

`FileExists` Verifies the existence of a filename

`hexstr` Convert a value to an ASCII hexadecimal representation

`LowString` Convert a string to lowercase ASCII

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

`removenulls`

`StreamErrorProcedure` Generic stream error procedure

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

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

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

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

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

`UpString` Convert a string to uppercase ASCII

`ValBinary`

`ValDecimal`

`ValHexadecimal`

`ValOctal`

10.3 Functions and Procedures

boolstr function

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.

decstr function

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 function

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

DirectoryExists function

Declaration `Function DirectoryExists(const 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.
DName Name of the directory to check
Returns FALSE if the directory cannot be opened or if it does not exist.

EscapeToPascal function

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

FileExists function

Declaration `Function FileExists(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.

FName Name of the file to check

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

hexstr function

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

Description Convert a value to an ASCII hexadecimal representation

LowString function

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

Description Convert a string to lowercase ASCII

Printf function

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.

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

removenulls function

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

Description Remove all null characters from a string.

StreamErrorProcedure procedure

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

Description Generic stream error procedure

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

SwapLong procedure

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

Description Change the endian of a 32-bit value

SwapWord procedure

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

Description Change the endian of a 16-bit value

Trim function

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

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

TrimLeft function

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

Description Remove all whitespace from the start of a string

TrimRight function

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

Description Remove all whitespace from the end of a string

UpString function

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

Description Convert a string to uppercase ASCII

ValBinary function

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.

ValDecimal function

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.

ValHexadecimal function

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.

ValOctal function

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.

10.4 Author

Carl Eric Codere