PasDoc's autodoc

Pasdoc

February 14, 2026

Contents

Chapter 1

Pasdoc Sources Overview

This is the documentation of the pasdoc sources, intended for pasdoc developers. For user's documentation see [https://pasdoc.github.io/].
Contents:
General overview of the data flow in pasdoc:

1.1 Parsing

`TTokenizer`(28.4) reads the source file, and converts it to a series of `TToken`(28.4)s.
`TScanner`(20.4) uses an underlying `TTokenizer`(28.4) and also returns a series of `TToken`(28.4)s, but in addition it understands and interprets $define, $ifdef and similar compiler directives. While `TTokenizer`(28.4) simply returns all tokens, `TScanner`(20.4) returns only those tokens that are not "$ifdefed out". E.g. if WIN32 is not defined then the `TScanner`(20.4) returns only tokens "`const LineEnding = #10;`" for the following code: **const** `LineEnding = {$ifdef WIN32} #13#10 {$else} #10 {$endif};`

Finally `TParser`(17.4) uses an underlying `TScanner`(20.4) and interprets the series of tokens, as e.g. "here I see a declaration of variable Foo, of type Integer". The Parser stores everything it reads in a `TPasUnit`(12.4) instance.
If you ever wrote a program that interprets a text language, you will see that there is nothing special here: We have a lexer (`TScanner`(20.4), a simplified lexer in `TTokenizer`(28.4)) and a parser (`TParser`(17.4)).
It is important to note that pasdoc's parser is somewhat unusual, compared to "normal" parsers that are used e.g. in Pascal compilers.

1. Pasdoc's parser is "cheating": It does not really understand everything it reads. E.g. the parameter section of a procedure declaration is parsed "blindly", by simply reading tokens up to a matching closing parenthesis. Such cheating obviously simplifies the parser implementation, but it also makes pasdoc's parser "dumber", see [https://pasdoc.github.io/ToDoParser].

2. Pasdoc's parser collects the comments before each declaration, since these comments must be converted and placed in the final documentation (while "normal" parsers usually treat comments as a meaningless white-space).

1.2 Storing

The unit PasDoc_Items(12) provides a comfortable class hierarchy to store a parsed Pascal source tree. TPasUnit(12.4) is a "root class" (container-wise), it contains references to all other items within a unit, every item is some instance of TPasItem(12.4).

1.3 Generators

The last link in the chain are the generators. A generator uses the stored TPasItem(12.4) tree and generates the final documentation. The base abstract class for a generator is TDocGenerator(4.4), this provides some general mechanisms used by all generators. From TDocGenerator(4.4) descend more specialized generator classes, like TGenericHTMLDocGenerator(5.4), THTMLDocGenerator(5.4), TTexDocGenerator(7.4) and others.

1.4 Notes

Note that the parser and the generators do not communicate with each other directly. The parser stores things in the TPasItem(12.4) tree. Generators read and process the TPasItem(12.4) tree.
So the parser cannot do any stupid thing like messing with some HTML-specific or LaTeX-specific issues of generating documentation. And the generator cannot deal with parsing Pascal source code.
Actually, this makes the implementation of the generator independent enough to be used in other cases, e.g. to generate an "introduction" file for the final documentation, like the one you are reading right now.

Chapter 2

Unit PasDoc_Aspell

2.1 Description

Spellchecking using Aspell.

2.2 Uses

- `SysUtils`
- `Classes`
- `Contnrs`
- `PasDoc_ProcessLineTalk`(18)
- `PasDoc_ObjectVector`(15)
- `PasDoc_Types`(29)

2.3 Overview

`TSpellingError Class` Single misspelling found by `TAspellProcess.CheckString`(2.4).

`TAspellProcess Class` This is a class to interface with aspell through pipe.

2.4 Classes, Interfaces, Objects and Records

TSpellingError Class

Hierarchy

TSpellingError > TObject

Description

Single misspelling found by `TAspellProcess.CheckString`(2.4). One instance is created per misspelled word.

Fields

Word `public Word: string;`

The misspelled word exactly as it appeared in the checked text.

Offset `public Offset: Integer;`

Offset of the misspelled word within the string that was passed to `TAspellProcess.CheckString`(2.4).

Suggestions `public Suggestions: string;`

Comma-separated list of replacement suggestions from Aspell, or empty string if Aspell offered none (the '#' response).

TAspellProcess Class

Hierarchy

TAspellProcess > TObject

Description

This is a class to interface with aspell through pipe. It uses underlying `TProcessLineTalk`(18.4) to execute and "talk" with aspell.

Properties

AspellMode `public property AspellMode: string read FAspellMode;`

Aspell input mode (passed to aspell --mode command-line option) passed at construction. Empty string means the Aspell default.

AspellLanguage `public property AspellLanguage: string read FAspellLanguage;`

Language code for aspell (passed to aspell --lang command-line option) passed at construction. Empty string means the Aspell default.

OnMessage `public property OnMessage: TPasDocMessageEvent read FOnMessage write FOnMessage;`

Callback for diagnostic messages.

Methods

Create

Declaration `public constructor Create(const AAspellMode, AAspellLanguage: string; AOnMessage: TPasDocMessageEvent);`

Description Constructor. Values for AspellMode and AspellLanguage are the same as for aspell --mode and --lang command-line options. You can pass here '', then we will not pass appropriate command-line option to aspell.

Destroy

Declaration `public destructor Destroy; override;`

SetIgnoreWords

Declaration `public procedure SetIgnoreWords(Value: TStringList);`

Description Tell Aspell to ignore all words in Value for subsequent `CheckString`(2.4) calls.

CheckString

Declaration `public procedure CheckString(const AString: string; const AErrors: TObjectVector);`

Description Spellchecks AString and returns result. Will create an array of TSpellingError objects, one entry for each misspelled word. Offsets of TSpellingErrors will be relative to AString.

Chapter 3

Unit PasDoc_Base

3.1 Description

Manages parsing and documentation generation: `TPasDoc`(3.4).

Note: Unit name must be `PasDoc_Base` instead of just `PasDoc` to not conflict with the name of base program name `pasdoc.dpr`.

3.2 Uses

- `SysUtils`
- `Classes`
- `Contnrs`
- `PasDoc_Items`(12)
- `PasDoc_Languages`(13)
- `PasDoc_Gen`(4)
- `PasDoc_Types`(29)
- `PasDoc_StringVector`(25)
- `PasDoc_SortSettings`(22)
- `PasDoc_StreamUtils`(23)
- `PasDoc_TagManager`(26)

3.3 Overview

`TPasDoc Class` Manages parsing and documentation generation.

3.4 Classes, Interfaces, Objects and Records

TPasDoc Class

Hierarchy

TPasDoc > TComponent

Description

Manages parsing and documentation generation.

Properties

Units `public property Units: TPasUnits read FUnits;`

After `Execute`(3.4) has been called, `Units` holds the units that have been parsed.

Conclusion `public property Conclusion: TExternalItem read FConclusion;`

After `Execute`(3.4) has been called, `Conclusion` holds the conclusion.

Introduction `public property Introduction: TExternalItem read FIntroduction;`

After `Execute`(3.4) has been called, `Introduction` holds the introduction.

AdditionalFiles `public property AdditionalFiles: TExternalItemList read FAdditionalFiles;`

After `Execute`(3.4) has been called, `AdditionalFiles` holds the additional external files.

DescriptionFileNames `published property DescriptionFileNames: TStringVector read FDescriptionFileNames write SetDescriptionFileNames;`

Directives `published property Directives: TStringVector read FDirectives write SetDirectives;`

IncludeDirectories `published property IncludeDirectories: TStringVector read FIncludeDirectories write SetIncludeDirectories;`

OnWarning `published property OnWarning: TPasDocMessageEvent read FOnMessage write FOnMessage stored false;`

This is deprecated name for `OnMessage`(3.4)

OnMessage `published property OnMessage: TPasDocMessageEvent read FOnMessage write FOnMessage;`

ProjectName `published property ProjectName: string read FProjectName write FProjectName;`

The name PasDoc shall give to this documentation project, also used to name some of the output files.

SourceFileNames `published property SourceFileNames: TStringVector read FSourceFileNames write SetSourceFileNames;`

Title `published property Title: string read FTitle write FTitle;`

Verbosity `published property Verbosity: Cardinal read FVerbosity write FVerbosity default DEFAULT_VERBOSITY_LEVEL;`

StarOnly `published property StarOnly: boolean read GetStarOnly write SetStarOnly stored false;`

CommentMarkers `published property CommentMarkers: TStringList read FCommentMarkers write SetCommentMarkers;`

IgnoreMarkers `published property IgnoreMarkers: TStringList read FIgnoreMarkers write SetIgnoreMarkers;`

MarkerOptional `published property MarkerOptional: boolean read FMarkerOptional write FMarkerOptional default false;`

IgnoreLeading `published property IgnoreLeading: string read FIgnoreLeading write FIgnoreLeading;`

Generator `published property Generator: TDocGenerator read FGenerator write SetGenerator;`

ShowVisibilities `published property ShowVisibilities: TVisibilities read FShowVisibilities write FShowVisibilities;`

CacheDir `published property CacheDir: string read FCacheDir write FCacheDir;`

SortSettings `published property SortSettings: TSortSettings read FSortSettings write FSortSettings default [];`

This determines how items inside will be sorted. See –sort documentation.

IntroductionFileName `published property IntroductionFileName: string read FIntroductionFileName write FIntroductionFileName;`

ConclusionFileName `published property ConclusionFileName: string read FConclusionFileName write FConclusionFileName;`

AdditionalFilesNames `published property AdditionalFilesNames: TStringList read FAdditionalFilesNames;`

ImplicitVisibility `published property ImplicitVisibility: TImplicitVisibility read FImplicitVisibility write FImplicitVisibility default ivPublic;`

See command-line option --implicit-visibility documentation at –implicit-visibility documentation. This will be passed to parser instance.

HandleMacros `published property HandleMacros: boolean read FHandleMacros write FHandleMacros default true;`

AutoLink `published property AutoLink: boolean read FAutoLink write FAutoLink default false;`

This controls auto-linking, see –auto-link documentation.

AutoBackComments `published property AutoBackComments: boolean read FAutoBackComments write FAutoBackComments default false;`

InfoMergeType `published property InfoMergeType: TInfoMergeType read FInfoMergeType write FInfoMergeType;`

Methods

RemoveExcludedItems

Declaration `protected procedure RemoveExcludedItems(const c: TPasItems);`

Description Searches the description of each TPasUnit item in the collection for an excluded tag. If one is found, the item is removed from the collection. If not, the fields, methods and properties collections are called with RemoveExcludedItems If the collection is empty after removal of all items, it is disposed of and the variable is set to nil.

Notification

Declaration `protected procedure Notification(AComponent: TComponent; Operation: TOperation); override;`

Create

Declaration `public constructor Create(AOwner: TComponent); override;`

Description Creates object and sets fields to default values.

Destroy

Declaration `public destructor Destroy; override;`

AddSourceFileNames

Declaration `public procedure AddSourceFileNames(const AFileNames: TStringList);`

Description Adds source filenames from a stringlist

AddSourceFileNamesFromFile

Declaration `public procedure AddSourceFileNamesFromFile(const FileName: string; DashMeansStdin: boolean);`

Description Loads names of Pascal unit source code files from a text file. Adds all file names to `SourceFileNames`(3.4). If DashMeansStdin and AFileName = '-' then it will load filenames from stdin.

DoError

Declaration `public procedure DoError(const AMessage: string; const AArguments: array of const; const AExitCode: Word);`

Description Raises an exception.

DoMessage

Declaration `public procedure DoMessage(const AVerbosity: Cardinal; const AMessageType: TPasDocMessageType; const AMessage: string; const AArguments: array of const);`

Description Forwards a message to the `OnMessage`(3.4) event.

GenMessage

Declaration `public procedure GenMessage(const MessageType: TPasDocMessageType; const AMessage: string; const AVerbosity: Cardinal);`

Description for Generator messages

Execute

Declaration `public procedure Execute;`

Description Starts creating the documentation.

3.5 Constants

DEFAULT_VERBOSITY_LEVEL

Declaration `DEFAULT_VERBOSITY_LEVEL = 2;`

3.6 Authors

Johannes Berg <johannes@sipsolutions.de>
Ralf Junker (delphi@zeitungsjunge.de)
Erwin Scheuch-Heilig (ScheuchHeilig@t-online.de)
Marco Schmidt (marcoschmidt@geocities.com)
Michael van Canneyt (michael@tfdec1.fys.kuleuven.ac.be)
Michalis Kamburelis
Richard B. Winston <rbwinst@usgs.gov>
Arno Garrels <first name.name@nospamgmx.de>

3.7 Created

24 Sep 1999

Chapter 4

Unit PasDoc_Gen

4.1 Description

Base generator class `TDocGenerator`(4.4), to be specialized for specific output formats.

`PasDoc_Gen` contains the basic documentation generator object `TDocGenerator`(4.4). It is not sufficient by itself but the basis for all generators that produce documentation in a specific format like HTML or LaTex. They override `TDocGenerator`(4.4)'s virtual methods.

4.2 Uses

- `PasDoc_Items`(12)
- `PasDoc_Languages`(13)
- `PasDoc_StringVector`(25)
- `PasDoc_ObjectVector`(15)
- `PasDoc_HierarchyTree`(11)
- `PasDoc_Types`(29)
- `Classes`
- `Contnrs`
- `PasDoc_TagManager`(26)
- `PasDoc_Aspell`(2)
- `PasDoc_StreamUtils`(23)
- `PasDoc_StringPairVector`(24)

4.3 Overview

`TOverviewFileInfo Record`

`TListItemData Class` Collected information about @xxxList item.

`TListData Class` Collected information about @xxxList content.

`TRowData Class` Collected information about @row (or @rowHead).

`TTableData Class` Collected information about @table.

`TDocGenerator Class` Base generator class, to be specialized for specific output formats.

4.4 Classes, Interfaces, Objects and Records

TOverviewFileInfo Record

Fields

BaseFileName `public BaseFileName: string;`

TranslationId `public TranslationId: TTranslationId;`

TranslationHeadlineId `public TranslationHeadlineId: TTranslationId;`

NoItemsTranslationId `public NoItemsTranslationId: TTranslationId;`

TListItemData Class

Hierarchy

TListItemData > TObject

Description

Collected information about @xxxList item.

Properties

ItemLabel `public property ItemLabel: string read FItemLabel;`

This is only for @definitionList: label for this list item, taken from @itemLabel. Already in the processed form. For other lists this will always be ".

Text `public property Text: string read FText;`

This is content of this item, taken from @item. Already in the processed form, after `TDocGenerator.ConvertStr` etc. Ready to be included in final documentation.

Index `public property Index: Integer read FIndex;`

Number of this item. This should be used for @orderedList. When you iterate over `TListData.Items`, you should be aware that Index of list item is *not* necessarily equal to the position of item inside `TListData.Items`. That's because of @itemSetNumber tag.

Normal list numbering (when no @itemSetNumber tag was used) starts from 1. Using @itemSetNumber user is able to change following item's Index.

For unordered and definition lists this is simpler: Index is always equal to the position within `TListData.Items` (because @itemSetNumber is not allowed there). And usually you will just ignore Index of items on unordered and definition lists.

Methods

Create

Declaration `public constructor Create(AItemLabel, AText: string; AIndex: Integer);`

TListData Class

Hierarchy

TListData > `TObjectVector`(15.4) > TObjectList

Description

Collected information about @xxxList content. Passed to `TDocGenerator.FormatList`(4.4). Every item of this list should be non-nil instance of `TListItemData`(4.4).

Properties

ItemSpacing `public property ItemSpacing: TListItemSpacing read FItemSpacing;`

ListType `public property ListType: TListType read FListType;`

Methods

Create

Declaration `public constructor Create(const AOwnsObject: boolean); override;`

TRowData Class

Hierarchy

TRowData > TObject

Description

Collected information about @row (or @rowHead).

Fields

Head `public Head: boolean;`

`True` if this is for @rowHead tag.

Cells `public Cells: TStringList;`

Each item on this list is already converted (with @-tags parsed, converted by ConvertString etc.) content of given cell tag.

Methods

Create

Declaration `public constructor Create;`

Destroy

Declaration `public destructor Destroy; override;`

TTableData Class

Hierarchy

TTableData > `TObjectVector`(15.4) > TObjectList

Description

Collected information about @table. Passed to `TDocGenerator.FormatTable`(4.4). Every item of this list should be non-nil instance of `TRowData`(4.4).

Properties

MaxCellCount `public property MaxCellCount: Cardinal read FMaxCellCount;`

Maximum Cells.Count, considering all rows.

MinCellCount `public property MinCellCount: Cardinal read FMinCellCount;`

Minimum Cells.Count, considering all rows.

TDocGenerator Class

Hierarchy

TDocGenerator > TComponent

Description

Base generator class, to be specialized for specific output formats. This abstract object will do the complete process of writing documentation files. It will be given the collection of units that was the result of the parsing process and a configuration object that was created from default values and program parameters. Depending on the output format, one or more files may be created (HTML will create several, Tex only one).

Properties

CurrentStream

```
protected property CurrentStream: TStream read FCurrentStream;
```

Units

```
public property Units: TPasUnits read FUnits write FUnits;
```

Introduction

```
public property Introduction: TExternalItem read FIntroduction write FIntroduction;
```

Conclusion

```
public property Conclusion: TExternalItem read FConclusion write FConclusion;
```

AdditionalFiles

```
public property AdditionalFiles: TExternalItemList read FAdditionalFiles write FAdditionalFiles;
```

OnMessage

```
public property OnMessage: TPasDocMessageEvent read FOnMessage write FOnMessage;
```

Callback receiving messages from generator.

This is usually used internally by TPasDoc class, that assigns it's internal callback here when using this generator. Also, for the above reason, do not make this published.

See TPasDoc.OnMessage for something more useful for final programs.

Language

```
published property Language: TLanguageID read GetLanguage write SetLanguage default DEFAULT_LANGUAGE;
```

the (human) output language of the documentation file(s)

ProjectName

```
published property ProjectName: string read FProjectName write FProjectName;
```

Name of the project to create.

ExcludeGenerator

```
published property ExcludeGenerator: Boolean read FExcludeGenerator write FExcludeGenerator default false;
```

"Generator info" are things that can change with each invocation of pasdoc, with different pasdoc binary etc.

This includes

- pasdoc's compiler name and version,
- pasdoc's version and time of compilation

See –exclude-generator documentation. Default value is false (i.e. show them), as this information is generally considered useful.

Setting this to true is useful for automatically comparing two versions of pasdoc's output (e.g. when trying to automate pasdoc's tests).

IncludeCreationTime

```
published property IncludeCreationTime: Boolean read
FIncludeCreationTime write FIncludeCreationTime default
false;
```

Show creation time in the output.

UseLowercaseKeywords

```
published property UseLowercaseKeywords: Boolean read
FUseLowercaseKeywords write FUseLowercaseKeywords default
false;
```

Setting to define how literal tag keywords should appear in documentaion.

Title

```
published property Title: string read FTitle write
FTitle;
```

Title of the documentation, supplied by user. May be empty. See `TPasDoc.Title`(3.4).

DestinationDirectory

```
published property DestinationDirectory: string read
FDestDir write SetDestDir;
```

Destination directory for documentation. Must include terminating forward slash or backslash so that valid file names can be created by concatenating DestinationDirectory and a pathless file name.

OutputGraphVizUses

```
published property OutputGraphVizUses: boolean read
FGraphVizUses write FGraphVizUses default false;
```

generate a GraphViz diagram for the units dependencies

OutputGraphVizClassHierarchy

```
published property OutputGraphVizClassHierarchy: boolean
read FGraphVizClasses write FGraphVizClasses default
false;
```

generate a GraphViz diagram for the Class hierarchy

LinkGraphVizUses

```
published property LinkGraphVizUses: string read
FLinkGraphVizUses write FLinkGraphVizUses;
```

link the GraphViz uses diagram

LinkGraphVizClasses

```
published property LinkGraphVizClasses: string read
FLinkGraphVizClasses write FLinkGraphVizClasses;
```

link the GraphViz classes diagram

Abbreviations

```
published property Abbreviations: TStringList read
FAbbreviations write SetAbbreviations;
```

CheckSpelling

```
published property CheckSpelling: boolean read
FCheckSpelling write FCheckSpelling default false;
```

AspellLanguage

```
published property AspellLanguage: string read
FAspellLanguage write FAspellLanguage;
```

SpellCheckIgnoreWords

```
published property SpellCheckIgnoreWords: TStringList
read FSpellCheckIgnoreWords write
SetSpellCheckIgnoreWords;
```

AutoAbstract

```
published property AutoAbstract: boolean read
FAutoAbstract write FAutoAbstract default false;
```

The meaning of this is just like --auto-abstract command-line option. It is used in `ExpandDescriptions`(4.4).

LinkLook

```
published property LinkLook: TLinkLook read FLinkLook
write FLinkLook default llDefault;
```

This controls `SearchLink`(4.4) behavior, as described in –link-look documentation.

WriteUsesClause

```
published property WriteUsesClause: boolean read
FWriteUsesClause write FWriteUsesClause default false;
```

AutoLink

```
published property AutoLink: boolean read FAutoLink
write FAutoLink default false;
```

This controls auto-linking, see –auto-link documentation.

AutoLinkExclude

```
published property AutoLinkExclude: TStringList read
FAutoLinkExclude;
```

ExternalClassHierarchy

```
published property ExternalClassHierarchy: TStrings read
FExternalClassHierarchy write SetExternalClassHierarchy
stored StoredExternalClassHierarchy;
```

Markdown

```
published property Markdown: boolean read FMarkdown
write FMarkdown default false;
```

ShowSourcePosition

```
published property ShowSourcePosition: boolean read
FShowSourcePosition write FShowSourcePosition default
false;
```

Show source filename and line number in documentation output.

SourceRoot

```
published property SourceRoot: string read FSourceRoot
write FSourceRoot;
```

Root path for source files. Used to make relative paths, shown by `ShowSourcePosition`(4.4) and replaced by `SourceUrlPattern`(4.4). Leave empty to make `ShowSourcePosition`(4.4) and `SourceUrlPattern`(4.4) just take the final filename part to show / replace.

SourceUrlPattern

```
published property SourceUrlPattern: string read
FSourceUrlPattern write FSourceUrlPattern;
```

URL pattern for linking source positions. Use {`FILE`} for filename and {`LINE`} for line number. When set, source positions in the output become clickable links. Example: `https://github.com/owner/repo/blob/main/{FILE`

Fields

FLanguage `protected FLanguage: TPasDocLanguages;`

the (human) output language of the documentation file(s)

FClassHierarchy `protected FClassHierarchy: TStringCardinalTree;`

FUnits `protected FUnits: TPasUnits;`

list of all units that were successfully parsed

Methods

DoError

Declaration
```
protected procedure DoError(const AMessage: string; const AArguments:
array of const; const AExitCode: Word);
```

DoMessage

Declaration
```
protected procedure DoMessage(const AVerbosity: Cardinal; const
MessageType: TPasDocMessageType; const AMessage: string; const AArguments:
array of const);
```

CreateClassHierarchy

Declaration
```
protected procedure CreateClassHierarchy;
```

MakeItemLink

Declaration
```
protected function MakeItemLink(const Item: TBaseItem; const LinkCaption:
string; const LinkContext: TLinkContext): string; virtual;
```

Description Return a link to item Item which will be displayed as LinkCaption. Returned string may be directly inserted inside output documentation. LinkCaption will be always converted using ConvertString before writing, so don't worry about doing this yourself when calling this method.

LinkContext may be used in some descendants to present the link differently, see `TLinkContext`(4.5) for it's meaning.

If some output format doesn't support this feature, it can return simply ConvertString(LinkCaption). This is the default implementation of this method in this class.

WriteCodeWithLinksCommon

Declaration
```
protected procedure WriteCodeWithLinksCommon(const Item: TPasItem; const
Code: string; WriteItemLink: boolean; const NameLinkBegin, NameLinkEnd:
string);
```

Description This writes Code as a Pascal code. Links inside the code are resolved from Item. If WriteItemLink then Item.Name is made a link. Item.Name is printed between NameLinkBegin and NameLinkEnd.

HasSourcePosition

Declaration `protected function HasSourcePosition(const AItem: TPasItem; out ItemName, ItemFilenameInRoot, ItemUrl: string): boolean;`

Description Utility to process information from `TPasItem.SourceAbsoluteFileName`(12.4) and `TPasItem.SourceLine`(12. and decide whether to show it. If `True`, then we should show it.

Parameters **ItemName** is the name to show.

ItemFilenameInRoot is the filename, relative to SourceRoot.

ItemUrl is the URL to link to, if any (don't make a link if this is '').

CloseStream

Declaration `protected procedure CloseStream;`

Description If field `CurrentStream`(4.4) is assigned, it is disposed and set to nil.

CodeString

Declaration `protected function CodeString(const s: string): string; virtual; abstract;`

Description Return S formatted to look like code, e.g. <code>xxx</code> in HTML.

Given S is already in the final output format (with characters converted using `ConvertString`(4.4), @-tags expanded etc.).

ConvertString

Declaration `protected function ConvertString(const s: string): string; virtual; abstract;`

Description Converts for each character in S, thus assembling a String that is returned and can be written to the documentation file.

The @ character should not be converted, this will be done later on.

ConvertChar

Declaration `protected function ConvertChar(c: char): string; virtual; abstract;`

Description Converts a character to its converted form. This method should always be called to add characters to a string.

@ should also be converted by this routine.

CreateLink

Declaration `protected function CreateLink(const Item: TBaseItem): string; virtual;`

Description This function is supposed to return a reference to an item, that is the name combined with some linking information like a hyperlink element in HTML or a page number in Tex.

CreateStream

Declaration `protected function CreateStream(const AName: string): Boolean;`

Description Open output stream in the destination directory. If `CurrentStream`(4.4) still exists (<> nil), it is closed. Then, a new output stream in the destination directory is created and assigned to `CurrentStream`(4.4). The file is overwritten if exists.

Use this only for text files that you want to write using WriteXxx methods of this class (like WriteConverted). There's no point to use if for other files.

Returns `True` if creation was successful, `False` otherwise. When it returns `False`, the error message was already shown by DoMessage.

ExtractEmailAddress

Declaration `protected function ExtractEmailAddress(s: string; out S1, S2, EmailAddress: string): Boolean;`

Description Searches for an email address in String S. Searches for first appearance of the @ character

FixEmailaddressWithoutMailTo

Declaration `protected function FixEmailaddressWithoutMailTo(const PossibleEmailAddress: String): String;`

Description Searches for an email address in PossibleEmailAddress and appends mailto: if it's an email address and mailto: wasn't provided. Otherwise it simply returns the input.

Needed to link email addresses properly which doesn't start with mailto:

ExtractWebAddress

Declaration `protected function ExtractWebAddress(s: string; out S1, S2, WebAddress: string): Boolean;`

Description Searches for a web address in String S. It must either contain a http:// or start with www.

FindGlobal

Declaration `protected function FindGlobal(const NameParts: TNameParts): TBaseItem;`

Description Searches all items in all units (given by field `Units`(4.4)) for item with NameParts. Returns a pointer to the item on success, nil otherwise.

FindGlobalPasItem

Declaration `protected function FindGlobalPasItem(const NameParts: TNameParts): TPasItem; overload;`

Description Find a Pascal item, searching global namespace. Returns `Nil` if not found.

FindGlobalPasItem

Declaration `protected function FindGlobalPasItem(const ItemName: String): TPasItem; overload;`

Description Find a Pascal item, searching global namespace. Assumes that Name is only one component (not something with dots inside). Returns `Nil` if not found.

GetClassDirectiveName

Declaration `protected function GetClassDirectiveName(const Directive: TClassDirective): string;`

Description `GetClassDirectiveName` returns ' abstract', or ' sealed' for classes that abstract or sealed respectively. `GetClassDirectiveName` is used by `TTexDocGenerator`(7.4) and `TGenericHTMLDocGenerator`(5.4) in writing the declaration of the class.

GetCIOTypeName

Declaration `protected function GetCIOTypeName(const MyType: TCIOType): string;`

Description `GetCIOTypeName` writes a translation of MyType based on the current language. However, 'record' and 'packed record' are not translated.

LoadDescriptionFile

Declaration `protected procedure LoadDescriptionFile(n: string);`

Description Loads descriptions from file N and replaces or fills the corresponding comment sections of items.

SearchItem

Declaration `protected function SearchItem(s: string; const Item: TBaseItem; WarningIfNotSplittable: boolean): TBaseItem;`

Description Searches for item with name S.

If S is not splittable by SplitNameParts, returns nil. If WarningIfNotSplittable, additionally does DoMessage with appropriate warning.

Else (if S is "splittable"), seeks for S (first trying Item.FindName, if Item is not nil, then trying FindGlobal). Returns nil if not found.

SearchLink

Declaration `protected function SearchLink(s: string; const Item: TBaseItem; const LinkDisplay: string; const WarningIfLinkNotFound: TLinkNotFoundAction; out FoundItem: TBaseItem): string; overload;`

Description Searches for an item of name S which was linked in the description of Item. Starts search within item, then does a search on all items in all units using `FindGlobal`(4.4). Returns a link as String on success.

If S is not splittable by SplitNameParts, it always does DoMessage with appropriate warning and returns something like 'UNKNOWN' (no matter what is the value of WarningIfLinkNotFound). FoundItem will be set to nil in this case.

When item will not be found then:

- if WarningIfLinkNotFound is true then it returns CodeString(ConvertString(S)) and makes DoMessage with appropriate warning.
- else it returns '' (and does not do any DoMessage)

If LinkDisplay is not '', then it specifies explicite the display text for link. Else how exactly link does look like is controlled by `LinkLook`(4.4) property.

Parameters **FoundItem** is the found item instance or nil if not found.

SearchLink

Declaration `protected function SearchLink(s: string; const Item: TBaseItem; const LinkDisplay: string; const WarningIfLinkNotFound: TLinkNotFoundAction): string; overload;`

Description Just like previous overloaded version, but this doesn't return FoundItem (in case you don't need it).

StoreDescription

Declaration `protected procedure StoreDescription(ItemName: string; var t: string);`

WriteConverted

Declaration `protected procedure WriteConverted(const s: string; Newline: boolean); overload;`

Description Writes S to CurrentStream, converting it using `ConvertString`(4.4). Then optionally writes LineEnding.

WriteConverted

Declaration `protected procedure WriteConverted(const s: string); overload;`

Description Writes S to CurrentStream, converting it using `ConvertString`(4.4). No LineEnding at the end.

WriteConvertedLine

Declaration `protected procedure WriteConvertedLine(const s:  string);`

Description Writes S to CurrentStream, converting it using `ConvertString`(4.4). Then writes LineEnding.

WriteDirect

Declaration `protected procedure WriteDirect(const t:  string; Newline:  boolean); overload;`

Description Simply writes T to CurrentStream, with optional LineEnding.

WriteDirect

Declaration `protected procedure WriteDirect(const t:  string); overload;`

Description Simply writes T to CurrentStream.

WriteDirectLine

Declaration `protected procedure WriteDirectLine(const t:  string);`

Description Simply writes T followed by LineEnding to CurrentStream.

WriteUnit

Declaration `protected procedure WriteUnit(const HL: integer; const U: TPasUnit); virtual; abstract;`

Description Abstract method that writes all documentation for a single unit U to output, starting at heading level HL. Implementation must be provided by descendant objects and is dependent on output format.

WriteUnits

Declaration `protected procedure WriteUnits(const HL: integer);`

Description Writes documentation for all units, calling `WriteUnit`(4.4) for each unit.

WriteStartOfCode

Declaration `protected procedure WriteStartOfCode; virtual;`

WriteEndOfCode

Declaration `protected procedure WriteEndOfCode; virtual;`

WriteGVUses

Declaration `protected procedure WriteGVUses;`

Description output graphviz uses tree

WriteGVClasses

Declaration `protected procedure WriteGVClasses;`

Description output graphviz class tree

StartSpellChecking

Declaration `protected procedure StartSpellChecking(const AMode: string);`

Description starts the spell checker

CheckString

Declaration `protected procedure CheckString(const AString: string; const AErrors: TObjectVector);`

Description If CheckSpelling and spell checking was successfully started, this will run `FAspellProcess.CheckString`(2.4) and will report all errors using DoMessage with mtWarning.

Otherwise this just clears AErrors, which means that no errors were found.

EndSpellChecking

Declaration `protected procedure EndSpellChecking;`

Description closes the spellchecker

FormatPascalCode

Declaration `protected function FormatPascalCode(const Line: string): string; virtual;`

Description FormatPascalCode will cause Line to be formatted in the way that Pascal code is formatted in Delphi. Note that given Line is taken directly from what user put inside , it is not even processed by ConvertString. You should process it with ConvertString if you want.

FormatNormalCode

Declaration `protected function FormatNormalCode(AString: string): string; virtual;`

Description This will cause AString to be formatted in the way that normal Pascal statements (not keywords, strings, comments, etc.) look in Delphi.

FormatComment

Declaration `protected function FormatComment(AString: string): string; virtual;`

Description FormatComment will cause AString to be formatted in the way that comments other than compiler directives are formatted in Delphi. See: `FormatCompilerComment`(4.4).

FormatHex

Declaration `protected function FormatHex(AString: string): string; virtual;`

Description FormatHex will cause AString to be formatted in the way that Hex are formatted in Delphi.

FormatNumeric

Declaration `protected function FormatNumeric(AString: string): string; virtual;`

Description FormatNumeric will cause AString to be formatted in the way that Numeric are formatted in Delphi.

FormatFloat

Declaration `protected function FormatFloat(AString: string): string; virtual;`

Description FormatFloat will cause AString to be formatted in the way that Float are formatted in Delphi.

FormatString

Declaration `protected function FormatString(AString: string): string; virtual;`

Description FormatString will cause AString to be formatted in the way that strings are formatted in Delphi.

FormatKeyWord

Declaration `protected function FormatKeyWord(AString: string): string; virtual;`

Description FormatKeyWord will cause AString to be formatted in the way that reserved words are formatted in Delphi.

FormatCompilerComment

Declaration `protected function FormatCompilerComment(AString: string): string; virtual;`

Description FormatCompilerComment will cause AString to be formatted in the way that compiler directives are formatted in Delphi.

Paragraph

Declaration `protected function Paragraph:  string; virtual;`

Description This is paragraph marker in output documentation.

Default implementation in this class simply returns ' ' (one space).

ShortDash

Declaration `protected function ShortDash:  string; virtual;`

Description See `TTagManager.ShortDash`(26.4). Default implementation in this class returns '-'.

EnDash

Declaration `protected function EnDash:  string; virtual;`

Description See `TTagManager.EnDash`(26.4). Default implementation in this class returns '--'.

EmDash

Declaration `protected function EmDash:  string; virtual;`

Description See `TTagManager.EmDash`(26.4). Default implementation in this class returns '---'.

HtmlString

Declaration `protected function HtmlString(const S: string):  string; virtual;`

Description Process HTML content, like provided by the @html tag. Override this function to decide what to put in output on such thing.

Note that S is not processed in any way, even with ConvertString. So you're able to copy user's input inside @html() verbatim to the output.

The default implementation is this class simply discards it, i.e. returns always ". Generators that know what to do with HTML can override this with simple "Result := S".

LatexString

Declaration `protected function LatexString(const S: string):  string; virtual;`

Description Process LaTeX content, like provided by the @latex tag.

The default implementation is this class simply discards it, i.e. returns always ". Generators that know what to do with raw LaTeX markup can override this with simple "Result := S".

LineBreak

Declaration `protected function LineBreak: string; virtual;`

Description Markup that forces line break in given output format (e.g. '
' in html or '\\' in LaTeX).

It is used on
tag (but may also be used on other occasions in the future).

In this class it returns ", because it's valid for an output generator to simply ignore tags if linebreaks can't be expressed in given output format.

URLLink

Declaration `protected function URLLink(const URL: string): string; overload; virtual;`

Description Markup to display URL in a description. E.g. HTML generator will want to wrap this in ....

Note that passed here URL is *not* processed by `ConvertString`(4.4) (because sometimes it could be undesirable). If you want you can process URL with ConvertString when overriding this method.

Default implementation in this class simply returns ConvertString(URL). This is good if your documentation format does not support anything like URL links.

URLLink

Declaration `protected function URLLink(const URL, LinkDisplay: string): string; overload; virtual;`

Description Text which will be shown for an URL tag.

URL is a link to a website or e-mail address. LinkDisplay is an optional parameter which will be used as the display name of the URL.

WriteExternal

Declaration `protected procedure WriteExternal(const ExternalItem: TExternalItem; const Id: TTranslationID);`

Description Write the introduction and conclusion of the project.

WriteExternalCore

Declaration `protected procedure WriteExternalCore(const ExternalItem: TExternalItem; const Id: TTranslationID); virtual; abstract;`

Description This is called from `WriteExternal`(4.4) when ExternalItem.Title and ShortTitle are already set, message about generating appropriate item is printed etc. This should write ExternalItem, including ExternalItem.DetailedDescription, ExternalItem.Authors, ExternalItem.Created, ExternalItem.LastMod.

WriteConclusion

Declaration `protected procedure WriteConclusion;`

Description Writes a conclusion for the project. See `WriteExternal`(4.4).

WriteIntroduction

Declaration `protected procedure WriteIntroduction;`

Description Writes an introduction for the project. See `WriteExternal`(4.4).

WriteAdditionalFiles

Declaration `protected procedure WriteAdditionalFiles;`

Description Writes the other files for the project. See `WriteExternal`(4.4).

FormatSection

Declaration
```
protected function FormatSection(HL: integer; const Anchor:  string; const
Caption:  string):  string; virtual; abstract;
```

Description Writes a section heading and a link-anchor.

FormatAnchor

Declaration
```
protected function FormatAnchor(const Anchor:  string):  string; virtual;
abstract;
```

Description Writes a link-anchor.

FormatBold

Declaration `protected function FormatBold(const Text:  string):  string; virtual;`

Description Return Text formatted using bold font.

Given Text is already in the final output format (with characters converted using `ConvertString`(4.4), @-tags expanded etc.).

Implementation of this method in this class simply returns `Result := Text`. Output generators that can somehow express bold formatting (or at least emphasis of some text) should override this.

See also `FormatItalic`**(4.4)** Return Text formatted using italic font.

FormatItalic

Declaration `protected function FormatItalic(const Text:  string):  string; virtual;`

Description Return Text formatted using italic font. Analogous to `FormatBold`(4.4).

FormatWarning

Declaration `protected function FormatWarning(const Text: string): string; virtual;`

Description Return Text using bold font by calling FormatBold(Text).

FormatNote

Declaration `protected function FormatNote(const Text: string): string; virtual;`

Description Return Text using italic font by calling FormatItalic(Text).

FormatPreformatted

Declaration `protected function FormatPreformatted(const Text: string): string; virtual;`

Description Return Text preserving spaces and line breaks. Note that Text passed here is not yet converted with ConvertString. The implementation of this method in this class just returns ConvertString(Text).

FormatImage

Declaration `protected function FormatImage(FileNames: TStringList): string; virtual;`

Description Return markup to show an image. FileNames is a list of possible filenames of the image. FileNames always contains at least one item (i.e. FileNames.Count >= 1), never contains empty lines (i.e. Trim(FileNames[I]) <> ''), and contains only absolute filenames.

E.g. HTML generator will want to choose the best format for HTML, then somehow copy the image from FileNames[Chosen] and wrap this in <img src="...">.

Implementation of this method in this class simply shows `FileNames[0]`. Output generators should override this.

FormatList

Declaration `protected function FormatList(ListData: TListData): string; virtual; abstract;`

Description Format a list from given ListData.

FormatTable

Declaration `protected function FormatTable(Table: TTableData): string; virtual; abstract;`

Description Return appropriate content for given Table. It's guaranteed that the Table passed here will have at least one row and in each row there will be at least one cell, so you don't have to check it within descendants.

FormatTableOfContents

Declaration `protected function FormatTableOfContents(Sections: TStringPairVector): string; virtual;`

Description Override this if you want to insert something on @tableOfContents tag. As a parameter you get already prepared tree of sections that your table of contents should show. Each item of Sections is a section on the level 1. Item's Name is section name, item's Value is section caption, item's Data is a TStringPairVector instance that describes subsections (on level 2) below this section. And so on, recursively.

Sections given here are never nil, and item's Data is never nil. But of course they may contain 0 items, and this should be a signal to you that given section doesn't have any subsections.

Default implementation of this method in this class just returns empty string.

BuildLinks

Declaration `public procedure BuildLinks; virtual;`

Description Creates anchors and links for all items in all units.

ExpandDescriptions

Declaration `public procedure ExpandDescriptions;`

Description Expands description for each item in each unit of `Units`(4.4). "Expands description" means that TTagManager.Execute is called, and item's DetailedDescription, AbstractDescription, AbstractDescriptionWasAutomatic (and many others, set by @-tags handlers) properties are calculated.

GetFileExtension

Declaration `public function GetFileExtension: string; virtual; abstract;`

Description Abstract function that provides file extension for documentation format. Must be overwritten by descendants.

LoadDescriptionFiles

Declaration `public procedure LoadDescriptionFiles(const c: TStringVector);`

Description Assumes C contains file names as PString variables. Calls `LoadDescriptionFile`(4.4) with each file name.

WriteDocumentation

Declaration `public procedure WriteDocumentation; virtual;`

Description Must be overwritten, writes all documentation. Will create either a single file or one file for each unit and each class, interface or object, depending on output format.

Create

Declaration `public constructor Create(AOwner:  TComponent); override;`

Destroy

Declaration `public destructor Destroy; override;`

ParseAbbreviationsFile

Declaration `public procedure ParseAbbreviationsFile(const AFileName:  string);`

4.5 Types

TOverviewFile

Declaration `TOverviewFile = (...);`

Description Overview files that pasdoc generates for multiple-document-formats like HTML (see `TGenericHTMLDocGenerat`

But not all of them are supposed to be generated by pasdoc, some must be generated by external programs by user, e.g. uses and class diagrams must be made by user using programs such as GraphViz. See type TCreatedOverviewFile for subrange type of TOverviewFile that specifies only overview files that are really supposed to be made by pasdoc.

Values
`ofUnits`
`ofClassHierarchy`
`ofCios`
`ofTypes`
`ofVariables`
`ofConstants`
`ofFunctionsAndProcedures`
`ofIdentifiers`
`ofGraphVizUses`
`ofGraphVizClasses`

TCreatedOverviewFile

Declaration `TCreatedOverviewFile = Low(TOverviewFile) ..  ofIdentifiers;`

TLinkLook

Declaration `TLinkLook = (...);`

Description

Values
`llDefault`
`llFull`
`llStripped`

TLinkNotFoundAction

Declaration `TLinkNotFoundAction = (...);`

Description

Values `lnfIgnore`

`lnfWarn`

`lnfWarnIfNotInternal`

TLinkContext

Declaration `TLinkContext = (...);`

Description This is used by `TDocGenerator.MakeItemLink`(4.4)

Values `lcCode` This means that link is inside some larger code piece, e.g. within FullDeclaration of some item etc. This means that we *may* be inside a context where used font has constant width.

`lcNormal` This means that link is inside some "normal" description text.

TListType

Declaration `TListType = (...);`

Description

Values `ltUnordered`

`ltOrdered`

`ltDefinition`

TListItemSpacing

Declaration `TListItemSpacing = (...);`

Description

Values `lisCompact`

`lisParagraph`

4.6 Constants

OverviewFilesInfo

Declaration
```
OverviewFilesInfo:  array[TOverviewFile] of TOverviewFileInfo = (
(BaseFileName:  'AllUnits' ; TranslationId:  trUnits ; TranslationHeadlineId:
trHeadlineUnits ; NoItemsTranslationId:  trNone ; ), (BaseFileName:
'ClassHierarchy'; TranslationId:  trClassHierarchy ; TranslationHeadlineId:
```

```
trClassHierarchy ; NoItemsTranslationId:  trNoCIOs ; ), (BaseFileName:
'AllClasses' ; TranslationId:  trCio ; TranslationHeadlineId:  trHeadlineCio
; NoItemsTranslationId:  trNoCIOs ; ), (BaseFileName:  'AllTypes' ;
TranslationId:  trTypes ; TranslationHeadlineId:  trHeadlineTypes ;
NoItemsTranslationId:  trNoTypes ; ), (BaseFileName:  'AllVariables' ;
TranslationId:  trVariables ; TranslationHeadlineId:  trHeadlineVariables ;
NoItemsTranslationId:  trNoVariables ; ), (BaseFileName:  'AllConstants' ;
TranslationId:  trConstants ; TranslationHeadlineId:  trHeadlineConstants ;
NoItemsTranslationId:  trNoConstants ; ), (BaseFileName:  'AllFunctions' ;
TranslationId:  trFunctionsAndProcedures; TranslationHeadlineId:
trHeadlineFunctionsAndProcedures; NoItemsTranslationId:  trNoFunctions ; ),
(BaseFileName:  'AllIdentifiers'; TranslationId:  trIdentifiers ;
TranslationHeadlineId:  trHeadlineIdentifiers ; NoItemsTranslationId:
trNoIdentifiers ; ), (BaseFileName:  'GVUses' ; TranslationId:  trGvUses ;
TranslationHeadlineId:  trGvUses ; NoItemsTranslationId:  trNone ; ),
(BaseFileName:  'GVClasses' ; TranslationId:  trGvClasses ;
TranslationHeadlineId:  trGvClasses ; NoItemsTranslationId:  trNoCIOs ; ) );
```

LowCreatedOverviewFile

Declaration `LowCreatedOverviewFile = Low(TCreatedOverviewFile);`

Description Using High(TCreatedOverviewFile) or High(Overview) where Overview: TCreatedOverview-File in PasDoc_GenHtml produces internal error in FPC 2.0.0. Same for Low(TCreatedOverviewFile).

This is submitted as FPC bug 4140, FPC bug 4140. Fixed in FPC 2.0.1 and FPC 2.1.1.

HighCreatedOverviewFile

Declaration `HighCreatedOverviewFile = High(TCreatedOverviewFile);`

4.7 Authors

Johannes Berg <johannes@sipsolutions.de>
Ralf Junker (delphi@zeitungsjunge.de)
Ivan Montes Velencoso (senbei@teleline.es)
Marco Schmidt (marcoschmidt@geocities.com)
Philippe Jean Dit Bailleul (jdb@abacom.com)
Rodrigo Urubatan Ferreira Jardim (rodrigo@netscape.net)
Grzegorz Skoczylas <gskoczylas@rekord.pl>
Pierre Woestyn <pwoestyn@users.sourceforge.net>
Michalis Kamburelis
Richard B. Winston <rbwinst@usgs.gov>
Ascanio Pressato
Arno Garrels <first name.name@nospamgmx.de>

4.8 Created

30 Aug 1998

Chapter 5

Unit PasDoc_GenHtml

5.1 Description

HTML documentation generator in `TGenericHTMLDocGenerator`(5.4).

5.2 Uses

- `PasDoc_Utils`(30)
- `PasDoc_Gen`(4)
- `PasDoc_Items`(12)
- `PasDoc_Languages`(13)
- `PasDoc_StringVector`(25)
- `PasDoc_Types`(29)
- `Classes`
- `Contnrs`
- `PasDoc_StringPairVector`(24)

5.3 Overview

`TGenericHTMLDocGenerator Class` HTML documentation generator.

`THTMLDocGenerator Class` Right now this is the same thing as TGenericHTMLDocGenerator.

`SignatureToHtmlId` DON'T EDIT – this file was automatically generated from "pasdoc.css"

5.4 Classes, Interfaces, Objects and Records

TGenericHTMLDocGenerator Class

Hierarchy

TGenericHTMLDocGenerator > `TDocGenerator`(4.4) > TComponent

Description

HTML documentation generator.

Extends `TDocGenerator`(4.4) and overwrites many of its methods to generate output in HTML format.

Properties

Header `published property Header: string read FHeader write FHeader;`

some HTML code to be written as header for every page

Footer `published property Footer: string read FFooter write FFooter;`

some HTML code to be written as footer for every page

HtmlBodyBegin `published property HtmlBodyBegin: string read FHtmlBodyBegin write FHtmlBodyBegin;`

HtmlBodyEnd `published property HtmlBodyEnd: string read FHtmlBodyEnd write FHtmlBodyEnd;`

HtmlHead `published property HtmlHead: string read FHtmlHead write FHtmlHead;`

CSS `published property CSS: string read FCSS write FCSS;`

Contents of the main CSS file (pasdoc.css).

Bootstrap `published property Bootstrap: boolean read FBootstrap write FBootstrap default true;`

If true, add Bootstrap CSS and JS. Definitions in `CSS`(5.4) will be evaluated after Bootstrap's ones.

NumericFilenames `published property NumericFilenames: boolean read FNumericFilenames write FNumericFilenames default false;`

if set to true, numeric filenames will be used rather than names with multiple dots

UseTipueSearch `published property UseTipueSearch: boolean read FUseTipueSearch write FUseTipueSearch default False;`

Enable Tipue fulltext search. See –use-tipue-search documentation.

Methods

MakeHead

Declaration `protected function MakeHead: string;`

Description Return common HTML content that goes inside <head>.

MakeBodyBegin

Declaration `protected function MakeBodyBegin: string; virtual;`

Description Return common HTML content that goes right after <body>.

MakeBodyEnd

Declaration `protected function MakeBodyEnd: string; virtual;`

Description Return common HTML content that goes right before </body>.

ConvertString

Declaration `protected function ConvertString(const s: string): string; override;`

ConvertChar

Declaration `protected function ConvertChar(c: char): string; override;`

Description Called by `ConvertString`(5.4) to convert a character. Will convert special characters to their html escape sequence -> test

WriteUnit

Declaration
```
protected procedure WriteUnit(const HL: integer; const U: TPasUnit);
override;
```

HtmlString

Declaration `protected function HtmlString(const S: string): string; override;`

Description overrides `TDocGenerator.HtmlString`(4.4).HtmlString to return the string verbatim (`TDocGenerator.HtmlSt`
discards those strings)

FormatPascalCode

Declaration `protected function FormatPascalCode(const Line: string): string; override;`

Description FormatPascalCode will cause Line to be formatted in the way that Pascal code is formatted in Delphi.

FormatComment

Declaration `protected function FormatComment(AString: string): string; override;`

Description FormatComment will cause AString to be formatted in the way that comments other than compiler directives are formatted in Delphi. See: `FormatCompilerComment`(5.4).

FormatHex

Declaration `protected function FormatHex(AString: string): string; override;`

Description FormatHex will cause AString to be formatted in the way that Hex are formatted in Delphi.

FormatNumeric

Declaration `protected function FormatNumeric(AString: string): string; override;`

Description FormatNumeric will cause AString to be formatted in the way that Numeric are formatted in Delphi.

FormatFloat

Declaration `protected function FormatFloat(AString: string): string; override;`

Description FormatFloat will cause AString to be formatted in the way that Float are formatted in Delphi.

FormatString

Declaration `protected function FormatString(AString: string): string; override;`

Description FormatKeyWord will cause AString to be formatted in the way that strings are formatted in Delphi.

FormatKeyWord

Declaration `protected function FormatKeyWord(AString: string): string; override;`

Description FormatKeyWord will cause AString to be formatted in the way that reserved words are formatted in Delphi.

FormatCompilerComment

Declaration `protected function FormatCompilerComment(AString: string): string; override;`

Description FormatCompilerComment will cause AString to be formatted in the way that compiler directives are formatted in Delphi.

CodeString

Declaration `protected function CodeString(const s: string): string; override;`

Description Makes a String look like a coded String, i.e. <CODE>TheString</CODE> in Html.

CreateLink

Declaration `protected function CreateLink(const Item: TBaseItem): string; override;`

Description Returns a link to an anchor within a document. HTML simply concatenates the strings with a "#" character between them.

WriteStartOfCode

Declaration `protected procedure WriteStartOfCode; override;`

WriteEndOfCode

Declaration `protected procedure WriteEndOfCode; override;`

WriteAnchor

Declaration `protected procedure WriteAnchor(const AName: string); overload;`

WriteAnchor

Declaration `protected procedure WriteAnchor(const AName, Caption: string); overload;`

Description Write an anchor. Note that the Caption is assumed to be already processed with the `ConvertString`(5.4).

Paragraph

Declaration `protected function Paragraph: string; override;`

EnDash

Declaration `protected function EnDash: string; override;`

EmDash

Declaration `protected function EmDash: string; override;`

LineBreak

Declaration `protected function LineBreak: string; override;`

URLLink

Declaration `protected function URLLink(const URL: string): string; override;`

URLLink

Declaration `protected function URLLink(const URL, LinkDisplay: string): string; override;`

WriteExternalCore

Declaration `protected procedure WriteExternalCore(const ExternalItem: TExternalItem; const Id: TTranslationID); override;`

MakeItemLink

Declaration `protected function MakeItemLink(const Item: TBaseItem; const LinkCaption: string; const LinkContext: TLinkContext): string; override;`

EscapeURL

Declaration `protected function EscapeURL(const AString: string): string; virtual;`

FormatSection

Declaration `protected function FormatSection(HL: integer; const Anchor: string; const Caption: string): string; override;`

FormatAnchor

Declaration `protected function FormatAnchor(const Anchor: string): string; override;`

FormatBold

Declaration `protected function FormatBold(const Text: string): string; override;`

FormatItalic

Declaration `protected function FormatItalic(const Text: string): string; override;`

FormatWarning

Declaration `protected function FormatWarning(const Text: string): string; override;`

FormatNote

Declaration `protected function FormatNote(const Text: string): string; override;`

FormatPreformatted

Declaration `protected function FormatPreformatted(const Text: string): string; override;`

FormatImage

Declaration `protected function FormatImage(FileNames: TStringList): string; override;`

FormatList

Declaration `protected function FormatList(ListData: TListData): string; override;`

FormatTable

Declaration `protected function FormatTable(Table: TTableData): string; override;`

FormatTableOfContents

Declaration `protected function FormatTableOfContents(Sections: TStringPairVector): string; override;`

Create

Declaration `public constructor Create(AOwner: TComponent); override;`

Destroy

Declaration `public destructor Destroy; override;`

GetFileExtension

Declaration `public function GetFileExtension: string; override;`

Description Returns HTML file extension ".htm".

WriteDocumentation

Declaration `public procedure WriteDocumentation; override;`

Description The method that does everything - writes documentation for all units and creates overview files.

THTMLDocGenerator Class

Hierarchy

THTMLDocGenerator > `TGenericHTMLDocGenerator`(5.4) > `TDocGenerator`(4.4) > TComponent

Description

Right now this is the same thing as TGenericHTMLDocGenerator. In the future it may be extended to include some things not needed for HtmlHelp generator.

Methods

MakeBodyBegin

Declaration `protected function MakeBodyBegin: string; override;`

MakeBodyEnd

Declaration `protected function MakeBodyEnd: string; override;`

5.5 Functions and Procedures

SignatureToHtmlId

Declaration `function SignatureToHtmlId(const Signature: string): string;`

Description DON'T EDIT – this file was automatically generated from "pasdoc.css"

5.6 Authors

Johannes Berg <johannes@sipsolutions.de>
Ralf Junker (delphi@zeitungsjunge.de)
Alexander Lisnevsky (alisnevsky@yandex.ru)
Erwin Scheuch-Heilig (ScheuchHeilig@t-online.de)
Marco Schmidt (marcoschmidt@geocities.com)
Hendy Irawan (ceefour@gauldong.net)
Wim van der Vegt (wvd_vegt@knoware.nl)
Thomas Mueller (www.dummzeuch.de)
David Berg (HTML Layout) <david@sipsolutions.de>
Grzegorz Skoczylas <gskoczylas@rekord.pl>
Michalis Kamburelis
Richard B. Winston <rbwinst@usgs.gov>
Ascanio Pressato
Arno Garrels <first name.name@nospamgmx.de>

Chapter 6

Unit PasDoc_GenHtmlHelp

6.1 Description

Generate HtmlHelp output.

6.2 Uses

- PasDoc_GenHtml(5)
- PasDoc_Utils(30)
- PasDoc_SortSettings(22)

6.3 Overview

THTMLHelpDocGenerator Class

6.4 Classes, Interfaces, Objects and Records

THTMLHelpDocGenerator Class

Hierarchy

THTMLHelpDocGenerator > TGenericHTMLDocGenerator(5.4) > TDocGenerator(4.4) > TComponent

Description

no description available, TGenericHTMLDocGenerator description followsHTML documentation generator. Extends TDocGenerator(4.4) and overwrites many of its methods to generate output in HTML format.

Properties

ContentsFile `published property ContentsFile:  string read FContentsFile write FContentsFile;`

Contains Name of a file to read HtmlHelp Contents from. If empty, create default contents file.

Methods

WriteDocumentation

Declaration `public procedure WriteDocumentation; override;`

Chapter 7

Unit PasDoc_GenLatex

7.1 Description

LaTeX documentation generator `TTexDocGenerator`(7.4).

7.2 Uses

- `PasDoc_Gen`(4)
- `PasDoc_Items`(12)
- `PasDoc_Languages`(13)
- `PasDoc_StringVector`(25)
- `PasDoc_Types`(29)
- `Classes`
- `Contnrs`

7.3 Overview

`TTexDocGenerator Class` LaTeX documentation generator.

7.4 Classes, Interfaces, Objects and Records

TTexDocGenerator Class

Hierarchy

TTexDocGenerator > `TDocGenerator`(4.4) > TComponent

Description

LaTeX documentation generator.

Extends `TDocGenerator`(4.4) and overwrites many of its methods to generate output in LaTex format.

Properties

Latex2rtf `published property Latex2rtf: boolean read FLatex2rtf write FLatex2rtf default false;`

Indicate if the output must be simplified for latex2rtf

LatexHead `published property LatexHead: TStrings read FLatexHead write SetLatexHead;`

The strings in `LatexHead` are inserted directly into the preamble of the LaTeX document. Therefore they must be valid LaTeX code.

Methods

ConvertString

Declaration `protected function ConvertString(const s: string): string; override;`

ConvertChar

Declaration `protected function ConvertChar(c: char): String; override;`

Description Called by `ConvertString`(7.4) to convert a character. Will convert special characters to their html escape sequence -> test

WriteUnit

Declaration `protected procedure WriteUnit(const HL: integer; const U: TPasUnit); override;`

LatexString

Declaration `protected function LatexString(const S: string): string; override;`

CodeString

Declaration `protected function CodeString(const s: string): string; override;`

Description Makes a String look like a coded String, i.e. '\begin{ttfamily}TheString\end{ttfamily}' in LaTeX. }

CreateLink

Declaration `protected function CreateLink(const Item: TBaseItem): string; override;`

Description Returns a link to an anchor within a document. LaTeX simply concatenates the strings with either a "-" or "." character between them.

WriteStartOfCode

Declaration `protected procedure WriteStartOfCode; override;`

WriteEndOfCode

Declaration `protected procedure WriteEndOfCode; override;`

Paragraph

Declaration `protected function Paragraph: string; override;`

ShortDash

Declaration `protected function ShortDash: string; override;`

LineBreak

Declaration `protected function LineBreak: string; override;`

URLLink

Declaration `protected function URLLink(const URL: string): string; override;`

URLLink

Declaration `protected function URLLink(const URL, LinkDisplay: string): string; override;`

WriteExternalCore

Declaration `protected procedure WriteExternalCore(const ExternalItem: TExternalItem; const Id: TTranslationID); override;`

FormatKeyWord

Declaration `protected function FormatKeyWord(AString: string): string; override;`

Description `FormatKeyWord` is called from within `FormatPascalCode`(7.4) to return AString in a bold font.

FormatCompilerComment

Declaration `protected function FormatCompilerComment(AString: string): string; override;`

Description `FormatCompilerComment` is called from within `FormatPascalCode`(7.4) to return AString in italics.

FormatComment

Declaration `protected function FormatComment(AString: string): string; override;`

Description `FormatComment` is called from within `FormatPascalCode`(7.4) to return AString in italics.

FormatAnchor

Declaration `protected function FormatAnchor(const Anchor: string): string; override;`

MakeItemLink

Declaration `protected function MakeItemLink(const Item: TBaseItem; const LinkCaption: string; const LinkContext: TLinkContext): string; override;`

FormatBold

Declaration `protected function FormatBold(const Text: string): string; override;`

FormatItalic

Declaration `protected function FormatItalic(const Text: string): string; override;`

FormatWarning

Declaration `protected function FormatWarning(const Text: string): string; override;`

FormatNote

Declaration `protected function FormatNote(const Text: string): string; override;`

FormatPreformatted

Declaration `protected function FormatPreformatted(const Text: string): string; override;`

FormatImage

Declaration `protected function FormatImage(FileNames: TStringList): string; override;`

FormatList

Declaration `protected function FormatList(ListData: TListData): string; override;`

FormatTable

Declaration `protected function FormatTable(Table: TTableData): string; override;`

FormatPascalCode

Declaration `public function FormatPascalCode(const Line: string): string; override;`

Description `FormatPascalCode` is intended to format Line as if it were Object Pascal code in Delphi or Lazarus. However, unlike Lazarus and Delphi, colored text is not used because printing colored text tends to be much more expensive than printing all black text.

GetFileExtension

Declaration `public function GetFileExtension: string; override;`

Description Returns Latex file extension ".tex".

WriteDocumentation

Declaration `public procedure WriteDocumentation; override;`

Description The method that does everything — writes documentation for all units and creates overview files.

Create

Declaration `public constructor Create(AOwner: TComponent); override;`

Destroy

Declaration `public destructor Destroy; override;`

EscapeURL

Declaration `public function EscapeURL(const AString: string): string; virtual;`

FormatSection

Declaration `public function FormatSection(HL: integer; const Anchor: string; const Caption: string): string; override;`

Chapter 8

Unit PasDoc_GenPHP

8.1 Description

PHP output generator.

8.2 Uses

- PasDoc_Utils(30)
- PasDoc_Gen(4)
- PasDoc_Items(12)
- PasDoc_Types(29)
- PasDoc_Languages(13)
- PasDoc_StringVector(25)

8.3 Overview

TPHPDocGenerator Class PHP output generator.

8.4 Classes, Interfaces, Objects and Records

TPHPDocGenerator Class

Hierarchy

TPHPDocGenerator > TDocGenerator(4.4) > TComponent

Description

PHP output generator.

Methods

CodeString

Declaration `protected function CodeString(const s: string): string; override;`

Description Overrides of ancestor abstract methods, not really used by PHP generation. As we output only a simple map (name->html_filename) for PHP now, we don't really use most of these methods. But we override them, as they are absract in ancestor.

WriteExternalCore

Declaration `protected procedure WriteExternalCore(const ExternalItem: TExternalItem; const Id: TTranslationID); override;`

FormatSection

Declaration `protected function FormatSection(HL: integer; const Anchor: string; const Caption: string): string; override;`

FormatAnchor

Declaration `protected function FormatAnchor(const Anchor: string): string; override;`

FormatList

Declaration `protected function FormatList(ListData: TListData): string; override;`

FormatTable

Declaration `protected function FormatTable(Table: TTableData): string; override;`

ConvertString

Declaration `protected function ConvertString(const s: string): string; override;`

Description Overrides actually used.

ConvertChar

Declaration `protected function ConvertChar(c: char): string; override;`

WriteUnit

Declaration `protected procedure WriteUnit(const HL: integer; const U: TPasUnit); override;`

WriteDocumentation

Declaration `public procedure WriteDocumentation; override;`

GetFileExtension

Declaration `public function GetFileExtension:  String; override;`

Chapter 9

Unit PasDoc_GenSimpleXML

9.1 Description

SimpleXML documentation generator `TSimpleXMLDocGenerator`(9.4).

9.2 Uses

- `PasDoc_Utils`(30)
- `PasDoc_Gen`(4)
- `PasDoc_Items`(12)
- `PasDoc_Languages`(13)
- `PasDoc_StringVector`(25)
- `PasDoc_Types`(29)
- `Classes`
- `Contnrs`
- `PasDoc_StringPairVector`(24)

9.3 Overview

`TSimpleXMLDocGenerator Class`

9.4 Classes, Interfaces, Objects and Records

TSimpleXMLDocGenerator Class

Hierarchy

TSimpleXMLDocGenerator > `TDocGenerator`(4.4) > TComponent

Description

no description available, TDocGenerator description followsBase generator class, to be specialized for specific output formats. This abstract object will do the complete process of writing documentation files. It will be given the collection of units that was the result of the parsing process and a configuration object that was created from default values and program parameters. Depending on the output format, one or more files may be created (HTML will create several, Tex only one).

Methods

CodeString

Declaration
```
protected function CodeString(const s:  string):  string; override;
```

ConvertString

Declaration
```
protected function ConvertString(const s:  string):  string; override;
```

ConvertChar

Declaration
```
protected function ConvertChar(c:  char):  string; override;
```

WriteUnit

Declaration
```
protected procedure WriteUnit(const HL: integer; const U: TPasUnit);
override;
```

WriteExternalCore

Declaration
```
protected procedure WriteExternalCore(const ExternalItem:  TExternalItem;
const Id:  TTranslationID); override;
```

FormatSection

Declaration
```
protected function FormatSection(HL: integer; const Anchor:  string; const
Caption:  string):  string; override;
```

FormatAnchor

Declaration
```
protected function FormatAnchor(const Anchor:  string):  string; override;
```

FormatTable

Declaration `protected function FormatTable(Table: TTableData): string; override;`

FormatList

Declaration `protected function FormatList(ListData: TListData): string; override;`

FormatBold

Declaration `protected function FormatBold(const Text: string): string; override;`

FormatItalic

Declaration `protected function FormatItalic(const Text: string): string; override;`

WriteDocumentation

Declaration `public procedure WriteDocumentation; override;`

GetFileExtension

Declaration `public function GetFileExtension: string; override;`

Chapter 10

Unit PasDoc_Hashes

10.1 Description

This unit implements an associative array. Before writing this unit, I've always missed Perl commands like `$h{abc}='def'` in Pascal.
Version 0.9.1 (works fine, don't know a bug, but 1.0? No, error checks are missing!)

Thanks to:

- Larry Wall for perl! And because I found a way how to implement a hash in perl's source code (hv.c and hv.h). This is not a direct translation from C to Pascal, but the algortithms are more or less the same.

Be warned:

- There is NOT a single ERROR CHECK in this unit. So expect anything! Especially there are NO checks on NEW and GETMEM functions — this might be dangerous on machines with low memory.

Programmer's information:

- you need Free Pascal Compiler or Delphi to compile this unit
- I recommend that you use Ansistrings {$H+} to be able to use keys longer than 255 chars

How to use this unit:

```
Simply put this unit in your uses line. You can use a new class - THash.
```

```
Initialize a hash (assuming "var h: THash;"):
h:=THash.Create;

Save a String:
h.SetString('key','value');          //perl: $h{key}='value'

Get the String back:
string_var:=h.GetString('key');      //perl: $string_var=$h{key}
returns '' if 'key' is not set

Test if a key has been set:
if h.KeyExists('key') then...        //perl: if (exists $h{key}) ...
returns a boolean

Delete a key
h.DeleteKey('key');                  //perl: delete $h{key};

Which keys do exist?
stringlist:=h.Keys;                  //perl: @list=keys %h;
returns a TStringList

Which keys do exist beginning with a special string?
stinglist:=h.Keys('abc');
returns all keys beginning with 'abc'  //perl: @list=grep /^abc/, keys %h;

How many keys are there?
number_of_keys:=h.Count;             //perl: $number=scalar keys %hash;

How many keys fit in memory allocated by THash?
c:=h.Capacity; (property)
THash automatically increases h.Capacity if needed.
This property is similar to Delphi's TList.Capacity property.
Note #1: You can't decrease h.Capacity.
Note #2: Capacity must be 2**n -- Create sets Capacity:=8;
         The same: Capacity:=17; , Capacity:=32;

I know there will be 4097 key/values in my hash. I don't want
the hash's capacity to be 8192 (wasting 50% ram). What to do?
h.MaxCapacity:=4096; => Capacity will never be > 4096.
Note: You can store more than MaxCapacity key/values in the
      hash (as many as you want) but Count should be >= Capacity
      for best performance.
MaxCapacity is -1 by default, meaning no limit.

Delete the hash
```

```
h.Free;    OR
h.Destroy;

Instead of just strings you can also save objects in my hash -
anything that is a pointer can be saved. Similar to SetString
and GetString  there are SetObject  and GetObject. The latter
returns nil if the key is unknown.
You can use both Set/GetString and Set/GetObject for a single
key string - no problem. But if DeleteKey is called, both the
string and the pointer are lost.
If you want to store a pointer  and a string, it is faster to
call  SetStringObject(key,string,pointer)  than SetString and
SetObject. The same is true getting the data back - GetString
and GetObject are  significantly slower  then a singe call to
GetStringObject(key, var string, var pointer).

Happy programming!
```

10.2 Uses

- SysUtils
- Classes

10.3 Overview

THashEntry Record

THash Class

TObjectHash Class

10.4 Classes, Interfaces, Objects and Records

THashEntry Record

Fields

next `public next: PHashEntry;`

hash `public hash: Integer;`

key `public key: String;`

value `public value: String;`

data `public data: Pointer;`

THash Class

Hierarchy

THash > TObject

Properties

Count `public property Count: Integer read FeldBelegt;`

Capacity `public property Capacity: Integer read GetCapacity write SetCapacity;`

MaxCapacity `public property MaxCapacity: Integer read FMaxCapacity write SetMaxCapacity;`

Methods

Create

Declaration `public constructor Create;`

Destroy

Declaration `public destructor Destroy; override;`

SetObject

Declaration `public procedure SetObject(_key: String; data: Pointer);`

SetString

Declaration `public procedure SetString(_key: String; data: String);`

SetStringObject

Declaration `public procedure SetStringObject(_key: String; s: String; p: Pointer);`

GetObject

Declaration `public function GetObject(_key: String): Pointer;`

GetString

Declaration `public function GetString(_key: String): String;`

GetStringObject

Declaration `public procedure GetStringObject(_key: String; var s: String; var p: Pointer);`

KeyExists

Declaration `public function KeyExists(_key: String): Boolean;`

DeleteKey

Declaration `public procedure DeleteKey(_key: String);`

Keys

Declaration `public function Keys: TStringList; overload;`

Keys

Declaration `public function Keys(beginning: String): TStringList; overload;`

TObjectHash Class

Hierarchy

TObjectHash > `THash`(10.4) > TObject

Properties

Items `public property Items[_key: string]: Pointer read GetObject write SetObject;`

Methods

Delete

Declaration `public procedure Delete(_key: String);`

10.5 Types

PPHashEntry

Declaration `PPHashEntry=^PHashEntry;`

PHashEntry

Declaration `PHashEntry=^THashEntry;`

TFakeArray

Declaration `TFakeArray=array[0..0] of PHashEntry;`

Description in FPC, I can simply use PPHashEntry as an array of PHashEntry - Delphi doesn't allow that. I need this stupid array[0..0] definition! From Delphi4, I could use a dynamic array.

PFakeArray

Declaration `PFakeArray=^TFakeArray;`

10.6 Author

Chapter 11

Unit PasDoc_HierarchyTree

11.1 Description

a n-ary tree for PasItems — for use in Class Hierarchy

11.2 Uses

- Classes
- PasDoc_Items(12)

11.3 Overview

TPasItemNode Class

TStringCardinalTree Class

NewStringCardinalTree

11.4 Classes, Interfaces, Objects and Records

TPasItemNode Class

Hierarchy

TPasItemNode > TObject

Properties

Name `public property Name: string read GetName;`

Item `public property Item: TPasItem read FItem;`

Parent `public property Parent: TPasItemNode read FParent;`

Fields

FChildren `protected FChildren: TList;`

FParent `protected FParent: TPasItemNode;`

FItem `protected FItem: TPasItem;`

FName `protected FName: string;`

Methods

GetName

Declaration `protected function GetName: string;`

AddChild

Declaration `protected procedure AddChild(const Child: TPasItemNode); overload;`

AddChild

Declaration `protected function AddChild(const AName: string): TPasItemNode; overload;`

AddChild

Declaration
```
protected function AddChild(const AItem: TPasItem): TPasItemNode;
overload;
```

FindItem

Declaration `protected function FindItem(const AName: string): TPasItemNode;`

Adopt

Declaration `protected procedure Adopt(const AChild: TPasItemNode);`

Orphan

Declaration `protected function Orphan(const AChild: TPasItemNode): boolean;`

Sort

Declaration `protected procedure Sort;`

Create

Declaration `public constructor Create;`

Destroy

Declaration `public destructor Destroy; override;`

Level

Declaration `public function Level: Integer;`

TStringCardinalTree Class

Hierarchy

TStringCardinalTree > TObject

Properties

IsEmpty `public property IsEmpty: boolean read GetIsEmpty;`

FirstItem `public property FirstItem: TPasItemNode read GetFirstItem;`

Fields

FRoot `protected FRoot: TPasItemNode;`

Methods

GetIsEmpty

Declaration `protected function GetIsEmpty: boolean;`

GetFirstItem

Declaration `protected function GetFirstItem: TPasItemNode;`

NeedRoot

Declaration `protected procedure NeedRoot;`

ItemOfName

Declaration `public function ItemOfName(const AName: string): TPasItemNode;`

InsertName

Declaration `public function InsertName(const AName: string): TPasItemNode; overload;`

InsertItem

Declaration `public function InsertItem(const AItem: TPasItem): TPasItemNode; overload;`

InsertParented

Declaration `public function InsertParented(const AParent: TPasItemNode; const AItem: TPasItem): TPasItemNode; overload;`

InsertParented

Declaration `public function InsertParented(const AParent: TPasItemNode; const AName: string): TPasItemNode; overload;`

MoveChildLast

Declaration `public procedure MoveChildLast(const Child, Parent: TPasItemNode);`

Level

Declaration `public function Level(const ANode: TPasItemNode): Integer;`

NextItem

Declaration `public function NextItem(const ANode: TPasItemNode): TPasItemNode;`

Sort

Declaration `public procedure Sort;`

Create

Declaration `public constructor Create;`

Destroy

Declaration `public destructor Destroy; override;`

11.5 Functions and Procedures

NewStringCardinalTree

Declaration `function NewStringCardinalTree: TStringCardinalTree;`

11.6 Author

Johannes Berg <johannes@sipsolutions.de>

Chapter 12

Unit PasDoc_Items

12.1 Description

All items that can appear within Pascal source code.

For each item (type, variable, class etc.) that may appear in a Pascal source code file and can thus be taken into the documentation, this unit provides an object type which will store name, unit, description and more on this item.

12.2 Uses

- SysUtils
- PasDoc_Types(29)
- PasDoc_StringVector(25)
- PasDoc_ObjectVector(15)
- PasDoc_Hashes(10)
- Classes
- Contnrs
- PasDoc_TagManager(26)
- PasDoc_Serialize(21)
- PasDoc_SortSettings(22)
- PasDoc_StringPairVector(24)
- PasDoc_Tokenizer(28)

12.3 Overview

`TRawDescriptionInfo Record` Raw description, in other words: the contents of comment before given item.

`TBaseItem Class` This is a basic item class, that is linkable, and has some `RawDescription`(12.4).

`TPasItem Class` This is a `TBaseItem`(12.4) descendant that is always declared inside some Pascal source file.

`TPasConstant Class` Pascal constant.

`TPasFieldVariable Class` Pascal global variable or field or nested constant of CIO.

`TPasType Class` Pascal type (but not a procedural type — these are expressed as `TPasRoutine`(12.4).)

`TPasEnum Class` Enumerated type.

`TPasRoutine Class` This represents:

1. global function/procedure,
2. method (function/procedure of a structure (TPasCio)),
3. type (pointer to one of the above) (in this case Name is the type name).

`TPasProperty Class`

`TPasCio Class` Extends `TPasItem`(12.4) to store all items in a class / an object, e.g. fields.

`EAnchorAlreadyExists Class`

`TExternalItem Class` `TExternalItem` extends `TBaseItem`(12.4) to store extra information about a project.

`TExternalItemList Class` `TExternalItemList` extends `TObjectVector`(15.4) to store non-nil instances of `TExternalItem`(12.4)

`TAnchorItem Class`

`TPasUnit Class` extends `TPasItem`(12.4) to store anything about a unit, its constants, types etc.; also provides methods for parsing a complete unit.

`TBaseItems Class` Container class to store a list of `TBaseItem`(12.4)s.

`TPasItems Class` Container class to store a list of `TPasItem`(12.4)s.

`TPasRoutines Class` Collection of methods.

`TPasProperties Class` Collection of properties.

`TPasNestedCios Class` Collection of classes / records / interfaces.

`TPasTypes Class` Collection of types.

`TPasUnits Class` Collection of units.

`RoutineTypeToString` Returns lowercased keyword associated with given method type.

`CioTypeToString` Returns lowercased keyword(s) associated with given structure type.

`VisibilitiesToStr` Returns VisibilityStr for each value in Visibilities, delimited by commas.

`VisToStr`

12.4 Classes, Interfaces, Objects and Records

TRawDescriptionInfo Record

Description

Raw description, in other words: the contents of comment before given item. Besides the content, this also specifies filename, begin and end positions of given comment.

Fields

Content `public Content: string;`

This is the actual content the comment.

StreamName `public StreamName: string;`

`StreamName` is the name of the TStream from which this comment was read. Will be " if no comment was found. It will be ' ' if the comment was somehow read from more than one stream.

BeginPosition `public BeginPosition: Int64;`

`BeginPosition` is the position in the stream of the start of the comment.

EndPosition `public EndPosition: Int64;`

`EndPosition` is the position in the stream of the character immediately after the end of the comment describing the item.

TBaseItem Class

Hierarchy

TBaseItem > `TSerializable`(21.4) > TObject

Description

This is a basic item class, that is linkable, and has some `RawDescription`(12.4).

Properties

DetailedDescription `public property DetailedDescription: string read FDetailedDescription write FDetailedDescription;`

Detailed description of this item.

In case of TPasItem, this is something more elaborate than `TPasItem.AbstractDescription`(12.4).

This is already in the form suitable for final output, ready to be put inside final documentation.

RawDescription `public property RawDescription: string read GetRawDescription write WriteRawDescription;`

This stores unexpanded version (as specified in user's comment in source code of parsed units) of description of this item.

Actually, this is just a shortcut to `RawDescriptionInfo(12.4).Content`

FullLink `public property FullLink: string read FFullLink write FFullLink;`

a full link that should be enough to link this item from anywhere else

LastMod `public property LastMod: string read FLastMod write FLastMod;`

Contains " or string with date of last modification. This string is already in the form suitable for final output format (i.e. already processed by TDocGenerator.ConvertString).

Name `public property Name: string read FName write FName;`

name of the item

Authors `public property Authors: TStringVector read FAuthors write SetAuthors;`

list of strings, each representing one author of this item

Created `public property Created: string read FCreated;`

Contains " or string with date of creation. This string is already in the form suitable for final output format (i.e. already processed by TDocGenerator.ConvertString).

AutoLinkHereAllowed `public property AutoLinkHereAllowed: boolean read FAutoLinkHereAllowed write FAutoLinkHereAllowed default true;`

Is auto-link mechanism allowed to create link to this item ? This may be set to `False` by @noAutoLinkHere tag in item's description.

Methods

Serialize

Declaration `protected procedure Serialize(const ADestination: TStream); override;`

Description Serialization of TPasItem need to store in stream only data that is generated by parser. That's because current approach treats "loading from cache" as equivalent to parsing a unit and stores to cache right after parsing a unit. So what is generated by parser must be written to cache.

That said,

1. It will not break anything if you will accidentally store in cache something that is not generated by parser. That's because saving to cache will be done anyway right after doing parsing, so properties not initialized by parser will have their initial values anyway. You're just wasting memory for cache, and some cache saving/loading time.
2. For now, in implementation of serialize/deserialize we try to add even things not generated by parser in a commented out code. This way if approach to cache will change some day, we will be able to use this code.

Deserialize

Declaration `protected procedure Deserialize(const ASource: TStream); override;`

Create

Declaration `public constructor Create; override;`

Destroy

Declaration `public destructor Destroy; override;`

RegisterTags

Declaration `public procedure RegisterTags(TagManager: TTagManager); virtual;`

Description It registers `TTag`(26.4)s that init `Authors`(12.4), `Created`(12.4), `LastMod`(12.4) and remove relevant tags from description. You can override it to add more handlers.

FindItem

Declaration `public function FindItem(const ItemName: string): TBaseItem; virtual;`

Description Search for an item called ItemName *inside this Pascal item*. For units, it searches for items declared *inside this unit* (like a procedure, or a class in this unit). For classes it searches for items declared *within this class* (like a method or a property). For an enumerated type, it searches for members of this enumerated type.

All normal rules of ObjectPascal scope apply, which means that e.g. if this item is a unit, `FindItem` searches for a class named ItemName but it *doesn't* search for a method named ItemName inside some class of this unit. Just like in ObjectPascal the scope of identifiers declared within the class always stays within the class. Of course, in ObjectPascal you can

qualify a method name with a class name, and you can also do such qualified links in pasdoc, but this is not handled by this routine (see `FindName`(12.4) instead).

Returns nil if not found.

Note that it never compares ItemName with Self.Name. You may want to check this yourself if you want.

Note that for TPasItem descendants, it always returns also some TPasItem descendant (so if you use this method with some TPasItem instance, you can safely cast result of this method to TPasItem).

Implementation in this class always returns nil. Override as necessary.

FindItemMaybeInAncestors

Declaration `public function FindItemMaybeInAncestors(const ItemName: string): TBaseItem; virtual;`

Description This is just like `FindItem`(12.4), but in case of classes or such it should also search within ancestors. In this class, the default implementation just calls FindItem.

FindName

Declaration `public function FindName(const NameParts: TNameParts): TBaseItem; virtual;`

Description Do all you can to find link specified by NameParts.

While searching this tries to mimic ObjectPascal identifier scope as much as it can. It seaches within this item, but also within class enclosing this item, within ancestors of this class, within unit enclosing this item, then within units used by unit of this item.

RawDescriptionInfo

Declaration `public function RawDescriptionInfo: PRawDescriptionInfo;`

Description Full info about `RawDescription`(12.4) of this item, including it's filename and position.

This is intended to be initialized by parser.

This returns `PRawDescriptionInfo`(12.6) instead of just `TRawDescriptionInfo`(12.4) to allow natural setting of properties of this record (otherwise `Item.RawDescriptionInfo.StreamName := 'foo';` would not work as expected) .

QualifiedName

Declaration `public function QualifiedName: String; virtual;`

Description Returns the qualified name of the item. This is intended to return a concise and not ambigous name. E.g. in case of TPasItem it is overridden to return Name qualified by class name and unit name.

In this class this simply returns Name.

BasePath

Declaration `public function BasePath:  string; virtual;`

Description The full (absolute) path used to resolve filenames in this item's descriptions. Must always end with PathDelim. In this class, this simply returns GetCurrentDir (with PathDelim added if needed).

Signature

Declaration `public function Signature:  string; virtual;`

TPasItem Class

Hierarchy

TPasItem > `TBaseItem`(12.4) > `TSerializable`(21.4) > TObject

Description

This is a `TBaseItem`(12.4) descendant that is always declared inside some Pascal source file.
Parser creates only items of this class (e.g. never some basic `TBaseItem`(12.4) instance). This class introduces properties and methods pointing to parent unit (`MyUnit`(12.4)) and parent class/interface/object/record (`MyObject`(12.4)). Also many other things not needed at `TBaseItem`(12.4) level are introduced here: things related to handling @abstract tag, @seealso tag, used to sorting items inside (`Sort`(12.4)) and some more.

Properties

AbstractDescription

`public property AbstractDescription:  string read FAbstractDescription write FAbstractDescription;`

Abstract description of this item. This is intended to be short (e.g. one sentence) description of this object.

This will be inited from @abstract tag in RawDescription, or cutted out from first sentence in RawDescription if --auto-abstract was used.

Note that this is already in the form suitable for final output, with tags expanded, chars converted etc.

AbstractDescriptionWasAutomatic

`public property AbstractDescriptionWasAutomatic: boolean read FAbstractDescriptionWasAutomatic write FAbstractDescriptionWasAutomatic;`

TDocGenerator.ExpandDescriptions sets this property to true if AutoAbstract was used and AbstractDescription of this item was automatically deduced from the 1st sentence of RawDescription.

Otherwise (if @abstract was specified explicitly, or there was no @abstract and AutoAbstract was false) this is set to false.

This is a useful hint for generators: it tells them that when they are printing *both* AbstractDescription and DetailedDescription of the item in one place (e.g. TTexDocGenerator.WriteItemLongDescription and TGenericHTMLDocGenerator.WriteItemLongDescription both do this) then they should *not* put any additional space between AbstractDescription and DetailedDescription.

This way when user will specify description like

```
{ First sentence. Second sentence. }
procedure Foo;
```

and --auto-abstract was on, then "First sentence." is the AbstractDescription, " Second sentence." is DetailedDescription, AbstractDescriptionWasAutomatic is true and and TGenericHTMLDocGenerator.WriteItemLongDescription can print them as "First sentence. Second sentence."

Without this property, TGenericHTMLDocGenerator.WriteItemLongDescription would not be able to say that this abstract was deduced automatically and would print additional paragraph break that was not present in desscription, i.e. "First sentence.<p> Second sentence."

MyUnit

`public property MyUnit: TPasUnit read FMyUnit write FMyUnit;`

Unit of this item.

MyObject

`public property MyObject: TPasCio read FMyObject write FMyObject;`

If this item is part of a class (or record, object., interface...), the corresponding class is stored here. `Nil` otherwise.

MyEnum

`public property MyEnum: TPasEnum read FMyEnum write FMyEnum;`

If this item is a member of an enumerated type, then the enclosing enumerated type is stored here. `Nil` otherwise.

Visibility

`public property Visibility: TVisibility read FVisibility write FVisibility;`

HintDirectives

`public property HintDirectives: THintDirectives read FHintDirectives write FHintDirectives;`

Hint directives specify is this item deprecated, platform-specific, library-specific, or experimental.

DeprecatedNote

`public property DeprecatedNote: string read FDeprecatedNote write FDeprecatedNote;`

Deprecation note, specified as a string after "deprecated" directive. Empty if none, always empty if `HintDirectives`(12.4) does not contain hdDeprecated.

FullDeclaration

```
public property FullDeclaration:  string read
FFullDeclaration write FFullDeclaration;
```

Full declaration of the item. This is full parsed declaration of the given item.

Note that that this is not used for some descendants. Right now it's used only with

- TPasConstant
- TPasFieldVariable (includes type, default values, etc.)
- TPasType
- TPasRoutine (includes parameter list, procedural directives, etc.)
- TPasProperty (includes read/write and storage specifiers, etc.)
- TPasEnum

  But in this special case, '...' is used instead of listing individual members, e.g. 'TEnumName = (...)'. You can get list of Members using TPasEnum.Members. Eventual specifics of each member should be also specified somewhere inside Members items, e.g. `TMyEnum = (meOne, meTwo = 3);`
  and `TMyEnum = (meOne, meTwo);`
  will both result in TPasEnum with equal FullDeclaration (just `'TMyEnum = (...)'`) but this `'= 3'` should be marked somewhere inside Members[1] properties.
- TPasItem when it's a CIO's field.

The intention is that in the future all TPasItem descendants will always have approprtate FullDeclaration set. It all requires adjusting appropriate places in PasDoc_Parser to generate appropriate FullDeclaration.

SeeAlso

```
public property SeeAlso:  TStringPairVector read
FSeeAlso;
```

Items here are collected from @seealso tags.

Name of each item is the 1st part of @seealso parameter. Value is the 2nd part of @seealso parameter.

Attributes

```
public property Attributes:  TStringPairVector read
FAttributes;
```

List of attributes defined for this item

Params

```
public property Params:  TStringPairVector read
FParams;
```

Parameters of method or property.

Name of each item is the name of parameter (without any surrounding whitespace), Value of each item is users description for this item (in already-expanded form).

This is already in the form processed by `TTagManager.Execute`(26.4), i.e. with links resolved, html characters escaped etc. So *don't* convert them (e.g. before writing to the final docs) once again (by some ExpandDescription or ConvertString or anything like that).

Raises

```
public property Raises:  TStringPairVector read
FRaises;
```

Exceptions raised by the method, or by property getter/setter.

Name of each item is the name of exception class (without any surrounding whitespace), Value of each item is users description for this item (in already-expanded form).

This is already in the form processed by `TTagManager.Execute`(26.4), i.e. with links resolved, html characters escaped etc. So *don't* convert them (e.g. before writing to the final docs) once again (by some ExpandDescription or ConvertString or anything like that).

SourceAbsoluteFileName

```
public property SourceAbsoluteFileName:  string read
FSourceAbsoluteFileName write FSourceAbsoluteFileName;
```

Source (absolute) file name where this item is declared. Set by the parser when parsing the item's declaration.

SourceLine

```
public property SourceLine:  Integer read FSourceLine
write FSourceLine;
```

Source line number (1-based) where this item is declared. Set by the parser when parsing the item's declaration.

Methods

Serialize

Declaration

```
protected procedure Serialize(const ADestination:  TStream); override;
```

Deserialize

Declaration

```
protected procedure Deserialize(const ASource:  TStream); override;
```

FindNameWithinUnit

Declaration `protected function FindNameWithinUnit(const NameParts: TNameParts): TBaseItem; virtual;`

Description This does the same thing as `FindName`(12.4) but it *doesn't* scan other units. If this item is a unit, it searches only inside this unit, else it searches only inside `MyUnit`(12.4) unit.

Actually `FindName`(12.4) uses this function.

Create

Declaration `public constructor Create; override;`

Destroy

Declaration `public destructor Destroy; override;`

FindName

Declaration `public function FindName(const NameParts: TNameParts): TBaseItem; override;`

RegisterTags

Declaration `public procedure RegisterTags(TagManager: TTagManager); override;`

HasDescription

Declaration `public function HasDescription: Boolean;`

Description Returns true if there is a DetailedDescription or AbstractDescription available.

QualifiedName

Declaration `public function QualifiedName: String; override;`

UnitRelativeQualifiedName

Declaration `public function UnitRelativeQualifiedName: string; virtual;`

Sort

Declaration `public procedure Sort(const SortSettings: TSortSettings); virtual;`

Description This recursively sorts all items inside this item, and all items inside these items, etc. E.g. in case of TPasUnit, this method sorts all variables, consts, CIOs etc. inside (honouring SortSettings), and also recursively calls Sort(SortSettings) for every CIO.

Note that this does not guarantee that absolutely everything inside will be really sorted. Some items may be deliberately left unsorted, e.g. Members of TPasEnum are never sorted (their declared order always matters, so we shouldn't sort them when displaying their documentation — reader of such documentation would be seriously misleaded). Sorting of other things depends on SortSettings — e.g. without ssMethods, CIOs methods will not be sorted.

So actually this method *makes sure that all things that should be sorted are really sorted.*

SetAttributes

Declaration `public procedure SetAttributes(var Value: TStringPairVector);`

InheritedItem

Declaration `public function InheritedItem: TPasItem; virtual;`

Description Get the closest item that this item inherits from.

GetInheritedItemDescriptions

Declaration `public function GetInheritedItemDescriptions: TStringPairVector; virtual;`

Description Generate a list of descriptions defined on this item in base classes.

This stops at the first class ancestor to have a description defined for an ancestor of this item. Along the way it will also collect descriptions of this item from any implemented interfaces.

If there is no description in any ancestor, it will return an empty vector.

BasePath

Declaration `public function BasePath: string; override;`

HasOptionalInfo

Declaration `public function HasOptionalInfo: boolean; virtual;`

Description Is optional information (that may be empty for after parsing unit and expanding tags) specified. Currently this checks `Params`(12.4) and `Raises`(12.4) and `TPasRoutine.Returns`(12.4).

IsOverride

Declaration `public function IsOverride: Boolean; virtual;`

Description Whether this item overrides an item in an ancestor.

TPasConstant Class

Hierarchy

TPasConstant > `TPasItem`(12.4) > `TBaseItem`(12.4) > `TSerializable`(21.4) > TObject

Description

Pascal constant.

Precise definition of "constant" for pasdoc purposes is "a name associated with a value". Optionally, constant type may also be specified in declararion. Well, Pascal constant always has some type, but pasdoc is too weak to determine the implicit type of a constant, i.e. to unserstand that constand `const A = 1` is of type Integer.

TPasFieldVariable Class

Hierarchy

TPasFieldVariable > `TPasItem`(12.4) > `TBaseItem`(12.4) > `TSerializable`(21.4) > TObject

Description

Pascal global variable or field or nested constant of CIO.

Precise definition is "a name with some type". And Optionally with some initial value, for global variables. It also holds a nested constant of extended classes and records. In the future we may introduce here some property like Type: TPasType.

Properties

IsConstant `public property IsConstant: Boolean read FIsConstant write FIsConstant;`

Set if this is a nested constant field

Methods

Serialize

Declaration `protected procedure Serialize(const ADestination: TStream); override;`

Deserialize

Declaration `protected procedure Deserialize(const ASource: TStream); override;`

TPasType Class

Hierarchy

TPasType > `TPasItem`(12.4) > `TBaseItem`(12.4) > `TSerializable`(21.4) > TObject

Description

Pascal type (but not a procedural type — these are expressed as `TPasRoutine`(12.4).)

TPasEnum Class

Hierarchy

TPasEnum > `TPasType`(12.4) > `TPasItem`(12.4) > `TBaseItem`(12.4) > `TSerializable`(21.4) > TObject

Description

Enumerated type.

Properties

Members `public property Members: TPasItems read FMembers;`

Fields

FMembers `protected FMembers: TPasItems;`

Methods

Serialize

Declaration
```
protected procedure Serialize(const ADestination: TStream); override;
```

Deserialize

Declaration
```
protected procedure Deserialize(const ASource: TStream); override;
```

StoreValueTag

Declaration
```
protected procedure StoreValueTag(ThisTag: TTag; var ThisTagData: TObject;
EnclosingTag: TTag; var EnclosingTagData: TObject; const TagParameter:
string; var ReplaceStr: string);
```

RegisterTags

Declaration
```
public procedure RegisterTags(TagManager: TTagManager); override;
```

FindItem

Declaration `public function FindItem(const ItemName: string): TBaseItem; override;`

Description Searches for a member of this enumerated type.

Destroy

Declaration `public destructor Destroy; override;`

Create

Declaration `public constructor Create; override;`

TPasRoutine Class

Hierarchy

TPasRoutine > `TPasItem`(12.4) > `TBaseItem`(12.4) > `TSerializable`(21.4) > TObject

Description

This represents:

1. global function/procedure,
2. method (function/procedure of a structure (TPasCio)),
3. type (pointer to one of the above) (in this case Name is the type name).

Properties

What `public property What: TRoutineType read FWhat write FWhat;`

Routine type, see `TRoutineType`(12.6).

IsType `public property IsType: Boolean read FIsType write FIsType default false;`

Is this a type (pointer to routine).

Returns `public property Returns: string read FReturns;`

What does the method return.

This is already in the form processed by `TTagManager.Execute`(26.4), i.e. with links resolved, html characters escaped etc. So *don't* convert them (e.g. before writing to the final docs) once again (by some ExpandDescription or ConvertString or anything like that).

Directives `public property Directives: TStandardDirectives read FDirectives write FDirectives;`

Set of method directive flags

ParamTypes `public property ParamTypes: TStringVector read FParamTypes write FParamTypes;`

Fields

FReturns `protected FReturns: string;`

FWhat `protected FWhat: TRoutineType;`

FDirectives `protected FDirectives: TStandardDirectives;`

FParamTypes `protected FParamTypes: TStringVector;`

FIsType `protected FIsType: Boolean;`

Methods

Serialize

Declaration `protected procedure Serialize(const ADestination: TStream); override;`

Deserialize

Declaration `protected procedure Deserialize(const ASource: TStream); override;`

StoreReturnsTag

Declaration
```
protected procedure StoreReturnsTag(ThisTag: TTag; var ThisTagData:
TObject; EnclosingTag: TTag; var EnclosingTagData: TObject; const
TagParameter: string; var ReplaceStr: string);
```

Create

Declaration `public constructor Create; override;`

Destroy

Declaration `public destructor Destroy; override;`

RegisterTags

Declaration `public procedure RegisterTags(TagManager: TTagManager); override;`

Description In addition to inherited, this also registers `TTag`(26.4) that inits `Returns`(12.4).

HasOptionalInfo

Declaration `public function HasOptionalInfo: boolean; override;`

Signature

Declaration `public function Signature: string; override;`

InheritedItem

Declaration `public function InheritedItem: TPasItem; override;`

Description Get the closest item that this item inherits from. Returns `Nil` if the routine does not override.

IsOverride

Declaration `public function IsOverride: Boolean; override;`

TPasProperty Class

Hierarchy

TPasProperty > `TPasItem`(12.4) > `TBaseItem`(12.4) > `TSerializable`(21.4) > TObject

Description

no description available, TPasItem description followsThis is a `TBaseItem`(12.4) descendant that is always declared inside some Pascal source file.
Parser creates only items of this class (e.g. never some basic `TBaseItem`(12.4) instance). This class introduces properties and methods pointing to parent unit (`MyUnit`(12.4)) and parent class/interface/object/record (`MyObject`(12.4)). Also many other things not needed at `TBaseItem`(12.4) level are introduced here: things related to handling @abstract tag, @seealso tag, used to sorting items inside (`Sort`(12.4)) and some more.

Properties

IndexDecl `public property IndexDecl: string read FIndexDecl write FIndexDecl;`

contains the optional index declaration, including brackets

Proptype `public property Proptype: string read FPropType write FPropType;`

contains the type of the property

Reader `public property Reader: string read FReader write FReader;`

read specifier

Writer `public property Writer: string read FWriter write FWriter;`

write specifier

DefaultInClass `public property DefaultInClass: Boolean read FDefaultInClass write FDefaultInClass;`

Is it the default property in class. For example "property Items[I: Integer]: String read GetItems; default;".

DefaultValue `public property DefaultValue: string read FDefaultValue write FDefaultValue;`

Default value. For example it will be 123 for declaration like this "property Xxx: Integer default 123;".

NoDefault `public property NoDefault: Boolean read FNoDefault write FNoDefault;`
true if Nodefault property

Stored `public property Stored: string read FStored write FStored;`
keeps Stored specifier

Fields

FDefaultInClass `protected FDefaultInClass: Boolean;`

FNoDefault `protected FNoDefault: Boolean;`

FIndexDecl `protected FIndexDecl: string;`

FStored `protected FStored: string;`

FDefaultValue `protected FDefaultValue: string;`

FWriter `protected FWriter: string;`

FPropType `protected FPropType: string;`

FReader `protected FReader: string;`

Methods

Serialize

Declaration `protected procedure Serialize(const ADestination: TStream); override;`

Deserialize

Declaration `protected procedure Deserialize(const ASource: TStream); override;`

InheritedItem

Declaration `public function InheritedItem: TPasItem; override;`

Description Get the closest item that this item inherits from. Returns `Nil` if the property does not override.

IsOverride

Declaration `public function IsOverride: Boolean; override;`

TPasCio Class

Hierarchy

TPasCio > `TPasType`(12.4) > `TPasItem`(12.4) > `TBaseItem`(12.4) > `TSerializable`(21.4) > TObject

Description

Extends `TPasItem`(12.4) to store all items in a class / an object, e.g. fields.

TODO: Rename to TPasStructure, most general term.

Properties

Ancestors `public property Ancestors: TStringPairVector read FAncestors;`

Name of the ancestor (class, object, interface). Each item is a TStringPair, with

- `Name` is the name (single Pascal identifier) of this ancestor,
- `Value` is the full declaration of this ancestor. For example, in addition to Name, this may include "specialize" directive (for FPC generic specialization) at the beginning. And "<foo,bar>" section at the end (for FPC or Delphi generic specialization).
- `Data` is a TPasItem reference to this ancestor, or `Nil` if not found. This is assigned only in TDocGenerator.BuildLinks.

Note that each ancestor is a TPasItem, *not necessarily* TPasCio. Consider e.g. the case

```
TMyStringList = Classes.TStringList;
TMyExtendedStringList = class(TMyStringList)
  ...
end;
```

At least for now, such declaration will result in TPasType (not TPasCio!) with Name = 'TMyStringList', which means that ancestor of TMyExtendedStringList will be a TPasType instance.

Note that the PasDoc_Parser already takes care of correctly setting Ancestors when user didn't specify any ancestor name at cio declaration. E.g. if this cio is a class, and user didn't specify ancestor name at class declaration, and this class name is not 'TObject' (in case pasdoc parses the RTL), the Ancestors[0] will be set to 'TObject'.

Cios `public property Cios: TPasNestedCios read FCios;`

Nested classes (and records, interfaces...).

ClassDirective `public property ClassDirective: TClassDirective read FClassDirective write FClassDirective;`

`ClassDirective` is used to indicate whether a class is sealed or abstract.

Fields `public property Fields: TPasItems read FFields;`

list of all fields

HelperTypeIdentifier `public property HelperTypeIdentifier: string read FHelperTypeIdentifier write FHelperTypeIdentifier;`

Class or record helper type identifier

Methods `public property Methods: TPasRoutines read FMethods;`

list of all methods

Properties `public property Properties: TPasProperties read FProperties;`

list of properties

MyType `public property MyType: TCIOType read FMyType write FMyType;`

determines if this is a class, an interface or an object

OutputFileName `public property OutputFileName: string read FOutputFileName write FOutputFileName;`

name of documentation output file (if each class / object gets its own file, that's the case for HTML, but not for TeX)

Types `public property Types: TPasTypes read FTypes;`

Simple nested types (that don't fall into `Cios`(12.4)).

NameWithGeneric `public property NameWithGeneric: string read FNameWithGeneric write FNameWithGeneric;`

Name, with optional "generic" directive before (for FPC generics) and generic type identifiers list "<foo,bar>" after (for FPC and Delphi generics).

Fields

FClassDirective `protected FClassDirective: TClassDirective;`

FFields `protected FFields: TPasItems;`

FMethods `protected FMethods: TPasRoutines;`

FProperties `protected FProperties: TPasProperties;`

FAncestors `protected FAncestors: TStringPairVector;`

FOutputFileName `protected FOutputFileName: string;`

FMyType `protected FMyType: TCIOType;`

FHelperTypeIdentifier `protected FHelperTypeIdentifier: string;`

FCios `protected FCios: TPasNestedCios;`

FTypes `protected FTypes: TPasTypes;`

FNameWithGeneric `protected FNameWithGeneric: string;`

Methods

Serialize

Declaration `protected procedure Serialize(const ADestination: TStream); override;`

Deserialize

Declaration `protected procedure Deserialize(const ASource: TStream); override;`

StoreMemberTag

Declaration `protected procedure StoreMemberTag(ThisTag: TTag; var ThisTagData: TObject; EnclosingTag: TTag; var EnclosingTagData: TObject; const TagParameter: string; var ReplaceStr: string);`

Create

Declaration `public constructor Create; override;`

Destroy

Declaration `public destructor Destroy; override;`

FindItem

Declaration `public function FindItem(const ItemName: string): TBaseItem; override;`

Description If this class (or interface or object) contains a field, method or property with the name of ItemName, the corresponding item pointer is returned.

FindItemMaybeInAncestors

Declaration `public function FindItemMaybeInAncestors(const ItemName: string): TBaseItem; override;`

FindItemInAncestors

Declaration `public function FindItemInAncestors(const ItemName: string): TPasItem;`

Description This searches for item (field, method or property) defined in ancestor of this cio. I.e. searches within the FirstAncestor, then within FirstAncestor.FirstAncestor, and so on. Returns nil if not found.

Sort

Declaration `public procedure Sort(const SortSettings: TSortSettings); override;`

RegisterTags

Declaration `public procedure RegisterTags(TagManager: TTagManager); override;`

FirstAncestor

Declaration `public function FirstAncestor: TPasItem;`

Description This returns Ancestors[0].Data, i.e. instance of the first ancestor of this Cio (or nil if it couldn't be found), or nil if Ancestors.Count = 0.

InheritedItem

Declaration `public function InheritedItem: TPasItem; override;`

Description Get the closest item that this item inherits from. Returns the value of `FirstAncestor`(12.4).

GetInheritedItemDescriptions

Declaration `public function GetInheritedItemDescriptions: TStringPairVector; override;`

FirstAncestorName

Declaration `public function FirstAncestorName: string;`

Description This returns the name of first ancestor of this Cio.

If Ancestor.Count > 0 then it simply returns Ancestors[0], i.e. the name of the first ancestor as was specified at class declaration, else it returns ''.

So this method is *roughly* something like `FirstAncestor.Name`, but with a few notable differences:

- FirstAncestor is nil if the ancestor was not found in items parsed by pasdoc. But this method will still return in this case name of ancestor.
- `FirstAncestor.Name` is the name of ancestor as specified at declaration of an ancestor. But this method is the name of ancestor as specified at declaration of this cio — with the same letter case, with optional unit specifier.

If this function returns '', then you can be sure that FirstAncestor returns nil. The other way around is not necessarily true — FirstAncestor may be nil, but still this function may return something <> ''.

ShowVisibility

Declaration `public function ShowVisibility: boolean;`

Description Is Visibility of items (Fields, Methods, Properties) important ?

EAnchorAlreadyExists Class

Hierarchy

EAnchorAlreadyExists > Exception

TExternalItem Class

Hierarchy

TExternalItem > `TBaseItem`(12.4) > `TSerializable`(21.4) > TObject

Description

`TExternalItem` extends `TBaseItem`(12.4) to store extra information about a project. `TExternalItem` is used to hold an introduction and conclusion to the project.

Properties

OutputFileName `public property OutputFileName: string read FOutputFileName write SetOutputFileName;`

name of documentation output file

ShortTitle `public property ShortTitle: string read FShortTitle write FShortTitle;`

SourceFileName `public property SourceFileName: string read FSourceFilename write FSourceFilename;`

Title `public property Title: string read FTitle write FTitle;`

Anchors `public property Anchors: TBaseItems read FAnchors;`

`Anchors` holds a list of `TAnchorItem`(12.4)s that represent anchors and sections within the `TExternalItem`. The `TAnchorItem`(12.4)s have no content so, they should not be indexed separately.

Methods

HandleTitleTag

Declaration
```
protected procedure HandleTitleTag(ThisTag: TTag; var ThisTagData:
TObject; EnclosingTag: TTag; var EnclosingTagData: TObject; const
TagParameter: string; var ReplaceStr: string);
```

HandleShortTitleTag

Declaration
```
protected procedure HandleShortTitleTag(ThisTag: TTag; var ThisTagData:
TObject; EnclosingTag: TTag; var EnclosingTagData: TObject; const
TagParameter: string; var ReplaceStr: string);
```

Create

Declaration `public Constructor Create; override;`

Destroy

Declaration `public destructor Destroy; override;`

RegisterTags

Declaration `public procedure RegisterTags(TagManager: TTagManager); override;`

FindItem

Declaration `public function FindItem(const ItemName: string): TBaseItem; override;`

AddAnchor

Declaration `public procedure AddAnchor(const AnchorItem: TAnchorItem); overload;`

AddAnchor

Declaration `public function AddAnchor(const AnchorName: string): TAnchorItem; overload;`

Description If item with Name (case ignored) already exists, this raises exception EAnchorAlreadyExists. Otherwise it adds TAnchorItem with given name to Anchors. It also returns created TAnchorItem.

BasePath

Declaration `public function BasePath: string; override;`

TExternalItemList Class

Hierarchy

TExternalItemList > `TObjectVector`(15.4) > TObjectList

Description

`TExternalItemList` extends `TObjectVector`(15.4) to store non-nil instances of `TExternalItem`(12.4)

Methods

Get

Declaration `public function Get(Index: Integer): TExternalItem;`

TAnchorItem Class

Hierarchy

TAnchorItem > `TBaseItem`(12.4) > `TSerializable`(21.4) > TObject

Description

no description available, TBaseItem description followsThis is a basic item class, that is linkable, and has some `RawDescription`(12.4).

Properties

ExternalItem `public property ExternalItem: TExternalItem read FExternalItem write FExternalItem;`

SectionLevel `public property SectionLevel: Integer read FSectionLevel write FSectionLevel default 0;`

If this is an anchor for a section, this tells section level (as was specified in the @section tag). Otherwise this is 0.

SectionCaption `public property SectionCaption: string read FSectionCaption write FSectionCaption;`

If this is an anchor for a section, this tells section caption (as was specified in the @section tag).

TPasUnit Class

Hierarchy

TPasUnit > `TPasItem`(12.4) > `TBaseItem`(12.4) > `TSerializable`(21.4) > TObject

Description

extends `TPasItem`(12.4) to store anything about a unit, its constants, types etc.; also provides methods for parsing a complete unit.
Note: Remember to always set `CacheDateTime`(12.4) after deserializing this unit.

Properties

CIOs `public property CIOs: TPasItems read FCIOs;`

list of classes, interfaces, objects, and records defined in this unit

Constants `public property Constants: TPasItems read FConstants;`

list of constants defined in this unit

FuncsProcs `public property FuncsProcs: TPasRoutines read FFuncsProcs;`

list of functions and procedures defined in this unit

UsesUnits `public property UsesUnits: TStringVector read FUsesUnits;`

The names of all units mentioned in a uses clause in the interface section of this unit.

This is never nil.

After `TDocGenerator.BuildLinks`(4.4), for every i: UsesUnits.Objects[i] will point to TPasUnit object with Name = UsesUnits[i] (or nil, if pasdoc's didn't parse such unit). In other words, you will be able to use UsesUnits.Objects[i] to obtain given unit's instance, as parsed by pasdoc.

Types `public property Types: TPasTypes read FTypes;`

list of types defined in this unit

Variables `public property Variables: TPasItems read FVariables;`

list of variables defined in this unit

OutputFileName `public property OutputFileName: string read FOutputFileName write FOutputFileName;`

name of documentation output file THIS SHOULD NOT BE HERE!

SourceFileName `public property SourceFileName: string read FSourceFilename write FSourceFilename;`

SourceFileDateTime `public property SourceFileDateTime: TDateTime read FSourceFileDateTime write FSourceFileDateTime;`

CacheDateTime `public property CacheDateTime: TDateTime read FCacheDateTime write FCacheDateTime;`

If WasDeserialized then this specifies the datetime of a cache data of this unit, i.e. when cache data was generated. If cache was obtained from a file then this is just the cache file modification date/time.

If not WasDeserialized then this property has undefined value – don't use it.

IsUnit `public property IsUnit: boolean read FIsUnit write FIsUnit;`

If `False`, then this is a program or library file, not a regular unit (though it's treated by pasdoc almost like a unit, so we use TPasUnit class for this).

IsProgram `public property IsProgram: boolean read FIsProgram write FIsProgram;`

Fields

FTypes `protected FTypes: TPasTypes;`

FVariables `protected FVariables: TPasItems;`

FCIOs `protected FCIOs: TPasItems;`

FConstants `protected FConstants: TPasItems;`

FFuncsProcs `protected FFuncsProcs: TPasRoutines;`

FUsesUnits `protected FUsesUnits: TStringVector;`

FSourceFilename `protected FSourceFilename: string;`

FOutputFileName `protected FOutputFileName: string;`

FCacheDateTime `protected FCacheDateTime: TDateTime;`

FSourceFileDateTime `protected FSourceFileDateTime: TDateTime;`

FIsUnit `protected FIsUnit: boolean;`

FIsProgram `protected FIsProgram: boolean;`

Methods

Serialize

Declaration `protected procedure Serialize(const ADestination: TStream); override;`

Deserialize

Declaration `protected procedure Deserialize(const ASource: TStream); override;`

Create

Declaration `public constructor Create; override;`

Destroy

Declaration `public destructor Destroy; override;`

AddCIO

Declaration `public procedure AddCIO(const i: TPasCio);`

AddConstant

Declaration `public procedure AddConstant(const i: TPasItem);`

AddType

Declaration `public procedure AddType(const i: TPasItem);`

AddVariable

Declaration `public procedure AddVariable(const i: TPasItem);`

FindInsideSomeClass

Declaration
```
public function FindInsideSomeClass(const AClassName, ItemInsideClass:
string): TPasItem;
```

FindInsideSomeEnum

Declaration `public function FindInsideSomeEnum(const EnumName, EnumMember: string): TPasItem;`

FindItem

Declaration `public function FindItem(const ItemName: string): TBaseItem; override;`

Sort

Declaration `public procedure Sort(const SortSettings: TSortSettings); override;`

FileNewerThanCache

Declaration `public function FileNewerThanCache(const FileName: string): boolean;`

Description Returns if unit WasDeserialized, and file FileName exists, and file FileName is newer than CacheDateTime.

So if FileName contains some info generated from information of this unit, then we can somehow assume that FileName still contains valid information and we don't have to write it once again.

Sure, we're not really 100% sure that FileName still contains valid information, but that's how current approach to cache works.

BasePath

Declaration `public function BasePath: string; override;`

TBaseItems Class

Hierarchy

TBaseItems > `TObjectVector`(15.4) > TObjectList

Description

Container class to store a list of `TBaseItem`(12.4)s.

Methods

Serialize

Declaration `protected procedure Serialize(const ADestination: TStream); virtual;`

Deserialize

Declaration `protected procedure Deserialize(const ASource: TStream); virtual;`

Create

Declaration `public constructor Create(const AOwnsObject: Boolean); override;`

Destroy

Declaration `public destructor Destroy; override;`

FindListItem

Declaration `public function FindListItem(const ASignature: string): TBaseItem;`

Description Find a given item name on a list. In the base class (TBaseItems), this simply searches the items (not recursively).

In some cases, it may look within the items (recursively), when the identifiers inside the item are in same namespace as the items themselves. Example: it will look also inside enumerated types members, because (when "scoped enums" are off) the enumerated members are in the same namespace as the enumerated type name.

Returns `Nil` if nothing can be found.

InsertItems

Declaration `public procedure InsertItems(const c: TBaseItems);`

Description Inserts all items of C into this collection. Disposes C and sets it to nil.

Add

Declaration `public procedure Add(const AObject: TBaseItem); virtual;`

Description During Add, AObject is associated with AObject.Name using hash table, so remember to set AObject.Name *before* calling Add(AObject).

ClearAndAdd

Declaration `public procedure ClearAndAdd(const AObject: TBaseItem);`

Description This is a shortcut for doing `Clear`(12.4) and then `Add(AObject)`(12.4). Useful when you want the list to contain exactly the one given AObject.

Delete

Declaration `public procedure Delete(const AIndex: Integer);`

Clear

Declaration `public procedure Clear; override;`

TPasItems Class

Hierarchy

TPasItems > `TBaseItems`(12.4) > `TObjectVector`(15.4) > TObjectList

Description

Container class to store a list of `TPasItem`(12.4)s.

Properties

PasItemAt `public property PasItemAt[const AIndex: Integer]: TPasItem read GetPasItemAt write SetPasItemAt;`

Methods

FindListItem

Declaration `public function FindListItem(const ASignature: string): TPasItem;`

Description A comfortable routine that just calls inherited and casts result to TPasItem, since every item on this list must be always TPasItem.

CopyItems

Declaration `public procedure CopyItems(const c: TPasItems);`

Description Copies all Items from c to this object, not changing c at all.

CountCIO

Declaration `public procedure CountCIO(var c, i, o: Integer);`

Description Counts classes, interfaces and objects within this collection.

RemovePrivateItems

Declaration `public procedure RemovePrivateItems;`

Description Checks each element's Visibility field and removes all elements with a value of viPrivate.

SortDeep

Declaration `public procedure SortDeep(const SortSettings: TSortSettings);`

Description This sorts all items on this list by their name, and also calls `Sort(SortSettings)`(12.4) for each of these items. This way it sorts recursively everything in this list.

This is equivalent to doing both `SortShallow`(12.4) and `SortOnlyInsideItems`(12.4).

SortOnlyInsideItems

Declaration `public procedure SortOnlyInsideItems(const SortSettings: TSortSettings);`

Description This calls `Sort(SortSettings)`(12.4) for each of items on the list. It does *not* sort the items on this list.

SortShallow

Declaration `public procedure SortShallow;`

Description This sorts all items on this list by their name. Unlike `SortDeep`(12.4), it does *not* call `Sort`(12.4) for each of these items. So "items inside items" (e.g. class methods, if this list contains TPasCio objects) remain unsorted.

SetFullDeclaration

Declaration `public procedure SetFullDeclaration(PrefixName: boolean; const Suffix: string);`

Description Sets FullDeclaration of every item to

1. Name of this item (only if PrefixName)
2. + Suffix.

Very useful if you have a couple of items that share a common declaration in source file, e.g. variables or fields declared like
`A, B: Integer;`

TPasRoutines Class

Hierarchy

TPasRoutines > `TPasItems`(12.4) > `TBaseItems`(12.4) > `TObjectVector`(15.4) > TObjectList

Description

Collection of methods.

Methods

Serialize

Declaration `protected procedure Serialize(const ADestination: TStream); override;`

Deserialize

Declaration `protected procedure Deserialize(const ASource: TStream); override;`

FindListItem

Declaration `public function FindListItem(const AName: string; Index: Integer): TPasRoutine; overload;`

Description Find an Index-th item with given name on a list. Index is 0-based. There could be multiple items sharing the same name (overloads) while method of base class returns only the one most recently added item.

Returns `Nil` if nothing can be found.

FindListItem

Declaration `public function FindListItem(const ANameOrSignature: string): TPasRoutine; overload;`

Add

Declaration `public procedure Add(const AItem: TBaseItem); override;`

Create

Declaration `public constructor Create(const AOwnsObject: Boolean); override;`

Destroy

Declaration `public destructor Destroy; override;`

TPasProperties Class

Hierarchy

TPasProperties > `TPasItems`(12.4) > `TBaseItems`(12.4) > `TObjectVector`(15.4) > TObjectList

Description

Collection of properties.

TPasNestedCios Class

Hierarchy

TPasNestedCios > `TPasItems`(12.4) > `TBaseItems`(12.4) > `TObjectVector`(15.4) > TObjectList

Description

Collection of classes / records / interfaces.

Methods

Create

Declaration `public constructor Create; reintroduce;`

TPasTypes Class

Hierarchy

TPasTypes > `TPasItems`(12.4) > `TBaseItems`(12.4) > `TObjectVector`(15.4) > TObjectList

Description

Collection of types.

Methods

FindListItem

Declaration `public function FindListItem(const AName: string): TPasItem;`

TPasUnits Class

Hierarchy

TPasUnits > `TPasItems`(12.4) > `TBaseItems`(12.4) > `TObjectVector`(15.4) > TObjectList

Description

Collection of units.

Properties

UnitAt `public property UnitAt[const AIndex: Integer]: TPasUnit read GetUnitAt write SetUnitAt;`

Methods

ExistsUnit

Declaration `public function ExistsUnit(const AUnit: TPasUnit): Boolean;`

12.5 Functions and Procedures

RoutineTypeToString

Declaration `function RoutineTypeToString(const RoutineType: TRoutineType): string;`

Description Returns lowercased keyword associated with given method type.

CioTypeToString

Declaration `function CioTypeToString(const CioType: TCIOType): String;`

Description Returns lowercased keyword(s) associated with given structure type.

VisibilitiesToStr

Declaration `function VisibilitiesToStr(const Visibilities: TVisibilities): string;`

Description Returns VisibilityStr for each value in Visibilities, delimited by commas.

VisToStr

Declaration `function VisToStr(const Vis: TVisibility): string;`

12.6 Types

TVisibility

Declaration `TVisibility = (...);`

Description Visibility of a field/method.

Values
- `viPublished` indicates field or method is published
- `viPublic` indicates field or method is public
- `viProtected` indicates field or method is protected
- `viStrictProtected` indicates field or method is strict protected
- `viPrivate` indicates field or method is private
- `viStrictPrivate` indicates field or method is strict private
- `viAutomated` indicates field or method is automated
- `viImplicit` implicit visibility, marks the implicit members if user used --implicit-visibility=implicit command-line option.

TVisibilities

Declaration `TVisibilities = set of TVisibility;`

TInfoMergeType

Declaration `TInfoMergeType = (...);`

Description Type of merging interface and implementation comments. See –implementation-comments documentation.

Values
- `imtNone` Implementation not parsed.
- `imtPreferIntf` Read both, prefer interface.
- `imtPreferImpl` Read both, prefer implementation.
- `imtJoin` Read both, concatenate.

PRawDescriptionInfo

Declaration `PRawDescriptionInfo = ^TRawDescriptionInfo;`

THintDirective

Declaration `THintDirective = (...);`

Description

Values `hdDeprecated`
`hdPlatform`
`hdLibrary`
`hdExperimental`

THintDirectives

Declaration `THintDirectives = set of THintDirective;`

TRoutineType

Declaration `TRoutineType = (...);`

Description Routine type for `TPasRoutine.What`(12.4)

Values `ROUTINE_CONSTRUCTOR`
`ROUTINE_DESTRUCTOR`
`ROUTINE_FUNCTION`
`ROUTINE_PROCEDURE`
`ROUTINE_OPERATOR`

TCIOType

Declaration `TCIOType = (...);`

Description Determine type of `TPasCio`(12.4) item, like a class or record.

Values `CIO_CLASS`
`CIO_PACKEDCLASS`
`CIO_OBJCCLASS`
`CIO_PACKEDOBJCCLASS`
`CIO_DISPINTERFACE`
`CIO_INTERFACE`
`CIO_OBJECT`
`CIO_PACKEDOBJECT`

CIO_RECORD

CIO_PACKEDRECORD

CIO_TYPE CIO_TYPE is used only when CIO is a type helper, designed by CIO.ClassDirective = CT_HELPER.

TClassDirective

Declaration `TClassDirective = (...);`

Description

Values CT_NONE

CT_ABSTRACT

CT_SEALED

CT_HELPER

CT_EXTERNAL external can be used with objcclass (see FPC PasCocoa)

12.7 Constants

VisibilityStr

Declaration
```
VisibilityStr: array[TVisibility] of string[16] = ( 'published', 'public',
'protected', 'strict protected', 'private', 'strict private', 'automated',
'implicit' );
```

AllVisibilities

Declaration
```
AllVisibilities: TVisibilities = [Low(TVisibility) .. High(TVisibility)];
```

DefaultVisibilities

Declaration
```
DefaultVisibilities: TVisibilities = [viProtected, viPublic, viPublished,
viAutomated];
```

InfoMergeTypeStr

Declaration
```
InfoMergeTypeStr: array[TInfoMergeType] of string = ( 'none',
'prefer-interface', 'prefer-implementation', 'join' );
```

CIORecordType

Declaration
```
CIORecordType = [CIO_RECORD, CIO_PACKEDRECORD];
```

CIONonHierarchy

Declaration
```
CIONonHierarchy = CIORecordType;
```

EmptyRawDescriptionInfo

Declaration `EmptyRawDescriptionInfo: TRawDescriptionInfo = ( Content: ''; StreamName: ''; BeginPosition: -1; EndPosition: -1; );`

12.8 Authors

Johannes Berg <johannes@sipsolutions.de>
Ralf Junker (delphi@zeitungsjunge.de)
Marco Schmidt (marcoschmidt@geocities.com)
Michalis Kamburelis
Richard B. Winston <rbwinst@usgs.gov>
Damien Honeyford
Arno Garrels <first name.name@nospamgmx.de>

12.9 Created

11 Mar 1999

Chapter 13

Unit PasDoc_Languages

13.1 Description

Language definitions and translations.

13.2 Overview

`TLanguageRecord Record`

`TPasDocLanguages Class` Language class to hold all translated strings

`LanguageFromIndex` Full language name

`LanguageFromID`

`SyntaxFromIndex` Language abbreviation

`SyntaxFromID`

`IDfromLanguage` Search for language by short or long name

`Translation` Manual translation of id into lang

`LanguageFromStr` Find a language with Syntax = S (case ignored).

`LanguageDescriptor` access LANGUAGE_ARRAY

`LanguageCode` Language code, using an official standardardized language names, suitable for Aspell or HTML.

13.3 Classes, Interfaces, Objects and Records

TLanguageRecord Record

Fields

Table `public Table: PTransTable;`

Name `public Name: string;`

Syntax `public Syntax: string;`

CharSet `public CharSet: string;`

AspellLanguage `public AspellLanguage: string;`

Name of this language as used by Aspell, see Aspell supported languages.

Set this to empty string if it's the same as our Syntax up to a dot. So a Syntax = 'pl' or Syntax = 'pl.iso-8859-2' already indicates AspellLanguage = 'pl'.

TODO: In the future, it would be nice if all language names used by PasDoc and Aspell matched. Aspell language naming follows the standard ISO 639-1 as far as I see, and we should probably follow it too (currently, we deviate for some languages).

So in the future, we'll probably replace Syntax and AspellLanguage by LanguageCode and CharsetCode. LanguageCode = code (suitable for both PasDoc and Aspell command-line; the thing currently up to a dot in Syntax), CharsetCode = the short representation of CharSet (the thing currently after a dot in Syntax).

TPasDocLanguages Class

Hierarchy

TPasDocLanguages > TObject

Description

Language class to hold all translated strings

Properties

CharSet `public property CharSet: string read FCharSet;`

Charset for current language.

Translation `public property Translation[ATranslationID: TTranslationID]: string read GetTranslation;`

Language `public property Language: TLanguageID read FLanguage write SetLanguage default DEFAULT_LANGUAGE;`

Fields

FCharSet `protected FCharSet: string;`

Methods

GetTranslation

Declaration `protected function GetTranslation(ATranslationID: TTranslationID): string;`

Description Translation for given ATranslationID.

Create

Declaration `public constructor Create;`

13.4 Functions and Procedures

LanguageFromIndex

Declaration `function LanguageFromIndex(i: integer): string;`

Description Full language name

LanguageFromID

Declaration `function LanguageFromID(i: TLanguageID): string;`

SyntaxFromIndex

Declaration `function SyntaxFromIndex(i: integer): string;`

Description Language abbreviation

SyntaxFromID

Declaration `function SyntaxFromID(i: TLanguageID): string;`

IDfromLanguage

Declaration `function IDfromLanguage(const s: string): TLanguageID;`

Description Search for language by short or long name

Translation

Declaration `function Translation(id: TTranslationID; lang: TLanguageID): string;`

Description Manual translation of id into lang

LanguageFromStr

Declaration `function LanguageFromStr(S: string; out LanguageId: TLanguageID): boolean;`

Description Find a language with Syntax = S (case ignored). Returns `True` and sets LanguageId if found, otherwise returns `False`.

LanguageDescriptor

Declaration `function LanguageDescriptor(id: TLanguageID): PLanguageRecord;`

Description access LANGUAGE_ARRAY

LanguageCode

Declaration `function LanguageCode(const Language: TLanguageID): string;`

Description Language code, using an official standardardized language names, suitable for Aspell or HTML.

13.5 Types

TLanguageID

Declaration `TLanguageID = (...);`

Description An enumeration type of all supported languages

Values
`lgBosnian`
`lgBrazilian_1252`
`lgBrazilian_utf8`
`lgBulgarian`
`lgCatalan`
`lgChinese_gb2312`
`lgCroatian`
`lgDanish`
`lgDutch`
`lgEnglish`
`lgFrench_ISO_8859_15`
`lgFrench_UTF_8`
`lgGerman_ISO_8859_15`
`lgGerman_UTF_8`
`lgIndonesian`
`lgItalian`

lgJavanese
lgPolish_CP1250
lgPolish_ISO_8859_2
lgRussian_1251
lgRussian_utf8
lgRussian_866
lgRussian_koi8
lgSlovak
lgSpanish
lgSwedish
lgHungarian_1250
lgCzech_CP1250
lgCzech_ISO_8859_2

TTranslationID

Declaration `TTranslationID = (...);`

Description An enumeration type of all static output texts. Warning: count and order changed!

Values
`trNoTrans` no translation ID assigned, so far
`trLanguage` the language name (English, ASCII), e.g. for file names.
`trUnits` map
`trClassHierarchy`
`trCio`
`trNestedCR`
`trNestedTypes`
`trIdentifiers`
`trGvUses`
`trGvClasses`
`trClasses` tables and members
`trClass`
`trObjcClass`
`trDispInterface`
`trInterface`
`trObjects`
`trObject`
`trRecord`

trPacked

trHierarchy

trFields

trMethods

trProperties

trLibrary

trPackage

trProgram

trUnit

trUses

trConstants

trFunctionsAndProcedures

trTypes

trType

trVariables

trAuthors

trAuthor

trCreated

trLastModified

trSubroutine

trParameters

trReturns

trExceptionsRaised

trExceptions

trException

trEnum

trVisibility visibilities

trPrivate

trStrictPrivate

trProtected

trStrictProtected

trPublic

trPublished

trAutomated

trImplicit

`trDeprecated` hints

`trPlatformSpecific`

`trLibrarySpecific`

`trExperimental`

`trOverview` headings

`trIntroduction`

`trConclusion`

`trAdditionalFile`

`trEnclosingClass`

`trHeadlineCio`

`trHeadlineConstants`

`trHeadlineFunctionsAndProcedures`

`trHeadlineIdentifiers`

`trHeadlineTypes`

`trHeadlineUnits`

`trHeadlineVariables`

`trSummaryCio`

`trDeclaration` column headings

`trDescription` as column OR section heading!

`trDescriptions` section heading for detailed descriptions

`trName`

`trValues`

`trWarningTag` tags with inbuilt heading

`trNoteTag`

`trNone` empty tables

`trNoCIOs`

`trNoCIOsForHierarchy`

`trNoTypes`

`trNoVariables`

`trNoConstants`

`trNoFunctions`

`trNoIdentifiers`

`trHelp` misc

`trLegend`

`trMarker`

trWarningOverwrite

trWarning

trGeneratedBy

trGeneratedOn

trOnDateTime

trSearch

trSeeAlso

trNested

trAttributes add more here

trSourcePosition

trDummy

RTransTable

Declaration `RTransTable = array[TTranslationID] of string;`

Description array holding the translated strings, or empty for default (English) text.

PTransTable

Declaration `PTransTable = ^RTransTable;`

PLanguageRecord

Declaration `PLanguageRecord = ^TLanguageRecord;`

Description language descriptor

13.6 Constants

DEFAULT_LANGUAGE

Declaration `DEFAULT_LANGUAGE = lgEnglish;`

lgDefault

Declaration `lgDefault = lgEnglish;`

13.7 Authors

Johannes Berg <johannes AT sipsolutions.de>
Ralf Junker <delphi AT zeitungsjunge.de>
Andrew Andreev <andrew AT alteragate.net> (Bulgarian translation)
Alexander Lisnevsky <alisnevsky AT yandex.ru> (Russian translation)
Hendy Irawan <ceefour AT gauldong.net> (Indonesian and Javanese translation)
Ivan Montes Velencoso (Catalan and Spanish translations)
Javi (Spanish translation)
Jean Dit Bailleul (Frensh translation)
Marc Weustinks (Dutch translation)
Martin Hansen <mh AT geus.dk> (Danish translation)
Michele Bersini <michele.bersini AT smartit.it> (Italian translation)
Peter Simkovic <simkovic_jr AT manal.sk> (Slovak translation)
Peter Th_rnqvist <pt AT timemetrics.se> (Swedish translation)
Rodrigo Urubatan Ferreira Jardim <rodrigo AT netscape.net> (Brasilian translation)
Alexandre da Silva <simpsomboy AT gmail.com> (Brasilian translation - Update)
Alexsander da Rosa <alex AT rednaxel.com> (Brasilian translation - UTF8)
Vitaly Kovalenko <v_l_kovalenko AT alsy.by> (Russian translation)
Grzegorz Skoczylas <gskoczylas AT rekord.pl> (corrected Polish translation)
Jonas Gergo <jonas.gergo AT ch...> (Hungarian translation)
Michalis Kamburelis
Ascanio Pressato (Some Italian translation)
JBarbero Quiter (updated Spanish translation)
Liu Chuanjun <1000copy AT gmail.com> (Chinese gb2312 translation)
Liu Da <xmacmail AT gmail.com> (Chinese gb2312 translation)
DoDi
Rene Mihula <rene.mihula@gmail.com> (Czech translation)
Yann Merignac (French translation)
Arno Garrels <first name.name@nospamgmx.de>

Chapter 14

Unit PasDoc_Main

14.1 Description

Main procedure, that does the complete job of the command-line PasDoc.

14.2 Overview

`Main` Does the complete job of the command-line PasDoc.

14.3 Functions and Procedures

Main

Declaration `procedure Main;`

Description Does the complete job of the command-line PasDoc.

- Process command-line options.
- Create TPasDoc, set it up, run `TPasDoc.Execute`(3.4) on it.

Chapter 15

Unit PasDoc_ObjectVector

15.1 Description

a simple object vector

15.2 Uses

- `Contnrs`
- `Classes`

15.3 Overview

`TObjectVector Class`

`ObjectVectorIsNilOrEmpty`

15.4 Classes, Interfaces, Objects and Records

TObjectVector Class

Hierarchy

TObjectVector > TObjectList

Methods

Create

Declaration `public constructor Create(const AOwnsObject:  boolean); virtual;`

Description This is only to make constructor virtual, while original TObjectList has a static constructor.

15.5 Functions and Procedures

ObjectVectorIsNilOrEmpty

Declaration `function ObjectVectorIsNilOrEmpty(const AOV: TObjectVector): boolean;`

15.6 Authors

Johannes Berg <johannes@sipsolutions.de>
Michalis Kamburelis

Chapter 16

Unit PasDoc_OptionParser

16.1 Description

Command line option parsing.

To use this unit, create an object of `TOptionParser`(16.4) and add options to it, each option descends from `TOption`(16.4). Then, call your object's `TOptionParser.ParseOptions`(16.4) method and options are parsed. After parsing, examine your option objects.

16.2 Uses

- `Classes`

16.3 Overview

`TOption Class` Base class for options.

`TBoolOption Class` Boolean option, "on" if was specified.

`TValueOption Class` Base class for all options that have values.

`TIntegerOption Class` Option that accepts additional Integer as a value.

`TStringOption Class` Option that accepts additional string as a value.

`TStringOptionList Class` stringlist option

`TPathListOption Class` pathlist option

`TSetOption Class` useful for making a choice of things

`TOptionParser Class` OptionParser — instantiate one of these for commandline parsing

16.4 Classes, Interfaces, Objects and Records

TOption Class

Hierarchy

TOption > TObject

Description

Base class for options.

This class implements all the basic functionality and provides abstract methods for the `TOptionParser`(16.4) class to call, which are overridden by descendants. It also provides function to write the explanation.

Properties

ShortForm `public property ShortForm: char read FShort write FShort;`

Short form of the option — single character — if #0 then not used

LongForm `public property LongForm: string read FLong write FLong;`

long form of the option — string — if empty, then not used

ShortCaseSensitive `public property ShortCaseSensitive: boolean read FShortSens write FShortSens;`

specified whether the short form should be case sensitive or not

LongCaseSensitive `public property LongCaseSensitive: boolean read FLongSens write FLongSens;`

specifies whether the long form should be case sensitive or not

WasSpecified `public property WasSpecified: boolean read FWasSpecified;`

signifies if the option was specified at least once

Explanation `public property Explanation: string read FExplanation write FExplanation;`

explanation for the option, see also `WriteExplanation`(16.4)

Fields

FShort `protected FShort: char;`

FLong `protected FLong: string;`

FShortSens `protected FShortSens: boolean;`

FLongSens `protected FLongSens: boolean;`

FExplanation `protected FExplanation: string;`

FWasSpecified `protected FWasSpecified: boolean;`

FParser `protected FParser: TOptionParser;`

Methods

ParseOption

Declaration `protected function ParseOption(const AWords: TStrings): boolean; virtual; abstract;`

Create

Declaration `public constructor Create(const AShort:char; const ALong: string);`

Description Create a new Option. Set AShort to #0 in order to have no short option. Technically you can set ALong to " to have no long option, but in practive *every* option should have long form. Don't override this in descendants (this always simply calls CreateEx). Override only CreateEx.

CreateEx

Declaration `public constructor CreateEx(const AShort:char; const ALong: string; const AShortCaseSensitive, ALongCaseSensitive: boolean); virtual;`

GetOptionWidth

Declaration `public function GetOptionWidth: Integer;`

Description returns the width of the string "-s, --long-option" where s is the short option. Removes non-existant options (longoption = " or shortoption = #0)

WriteExplanation

Declaration `public procedure WriteExplanation(const AOptWidth: Integer);`

Description writes the wrapped explanation including option format, AOptWidth determines how much it is indented & wrapped

TBoolOption Class

Hierarchy

TBoolOption > `TOption`(16.4) > TObject

Description

Boolean option, "on" if was specified.

Note: This cannot handle something like --option=false . Whether the option is `TurnedOn`(16.4) simply depends on whether it was specified.

Properties

TurnedOn `public property TurnedOn: boolean read FWasSpecified;`

Methods

ParseOption

Declaration `protected function ParseOption(const AWords: TStrings): boolean; override;`

TValueOption Class

Hierarchy

TValueOption > `TOption`(16.4) > TObject

Description

Base class for all options that have values.

These options that take one or more values of the form --option=value or --option value etc

Methods

CheckValue

Declaration
```
protected function CheckValue(const AString: String): boolean; virtual;
abstract;
```

ParseOption

Declaration `protected function ParseOption(const AWords: TStrings): boolean; override;`

TIntegerOption Class

Hierarchy

TIntegerOption > `TValueOption`(16.4) > `TOption`(16.4) > TObject

Description

Option that accepts additional Integer as a value.

Properties

Value `public property Value: Integer read FValue write FValue;`

Fields

FValue `protected FValue: Integer;`

Methods

CheckValue

Declaration `protected function CheckValue(const AString: String): boolean; override;`

TStringOption Class

Hierarchy

TStringOption > `TValueOption`(16.4) > `TOption`(16.4) > TObject

Description

Option that accepts additional string as a value.

Properties

Value `public property Value: String read FValue write FValue;`

Fields

FValue `protected FValue: String;`

Methods

CheckValue

Declaration `protected function CheckValue(const AString: String): boolean; override;`

TStringOptionList Class

Hierarchy

TStringOptionList > `TValueOption`(16.4) > `TOption`(16.4) > TObject

Description

stringlist option

accepts multiple strings and collates them even if the option itself is specified more than one time

Properties

Values `public property Values: TStringList read FValues;`

Fields

FValues `protected FValues: TStringList;`

Methods

CheckValue

Declaration `protected function CheckValue(const AString: String): Boolean; override;`

CreateEx

Declaration `public constructor CreateEx(const AShort: Char; const ALong: String; const AShortCaseSensitive, ALongCaseSensitive: Boolean); override;`

Destroy

Declaration `public destructor Destroy; override;`

TPathListOption Class

Hierarchy

TPathListOption > `TStringOptionList`(16.4) > `TValueOption`(16.4) > `TOption`(16.4) > TObject

Description

pathlist option

accepts multiple strings paths and collates them even if the option itself is specified more than one time. Paths in a single option can be separated by the DirectorySeparator

Methods

CheckValue

Declaration `public function CheckValue(const AString: String): Boolean; override;`

TSetOption Class

Hierarchy

TSetOption > `TValueOption`(16.4) > `TOption`(16.4) > TObject

Description

useful for making a choice of things

Values must not have a + or - sign as the last character as that can be used to add/remove items from the default set, specifying items without +/- at the end clears the default and uses only specified items

Properties

PossibleValues `public property PossibleValues: string read GetPossibleValues write SetPossibleValues;`

Values `public property Values: string read GetValues write SetValues;`

Fields

FPossibleValues `protected FPossibleValues: TStringList;`

FValues `protected FValues: TStringList;`

Methods

GetPossibleValues

Declaration `protected function GetPossibleValues: string;`

SetPossibleValues

Declaration `protected procedure SetPossibleValues(const Value: string);`

CheckValue

Declaration `protected function CheckValue(const AString: String): Boolean; override;`

GetValues

Declaration `protected function GetValues: string;`

SetValues

Declaration `protected procedure SetValues(const Value: string);`

CreateEx

Declaration `public constructor CreateEx(const AShort: Char; const ALong: String; const AShortCaseSensitive, ALongCaseSensitive: Boolean); override;`

Destroy

Declaration `public destructor Destroy; override;`

HasValue

Declaration `public function HasValue(const AValue: string): boolean;`

TOptionParser Class

Hierarchy

TOptionParser > TObject

Description

OptionParser — instantiate one of these for commandline parsing

This class is the main parsing class, although a lot of parsing is handled by `TOption`(16.4) and its descendants instead.

Properties

LeftList

```
public property LeftList: TStringList read FLeftList;
```

This StringList contains all the items from the command line that could not be parsed. Includes options that didn't accept their value and non-options like filenames specified on the command line

OptionsCount

```
public property OptionsCount: Integer read GetOptionsCount;
```

The number of option objects that were added to this parser

Options

```
public property Options[const AIndex: Integer]: TOption read
GetOption;
```

retrieve an option by index — you can use this and `OptionsCount`(16.4) to iterate through the options that this parser owns

ByName

```
public property ByName[const AName: string]: TOption read
GetOptionByLongName;
```

retrieve an option by its long form. Case sensitivity of the options is taken into account!

ByShortName

```
public property ByShortName[const AName: char]: TOption read
GetOptionByShortname;
```

retrieve an option by its short form. Case sensitivity of the options is taken into account!

ShortOptionStart

```
public property ShortOptionStart: Char read FShortOptionChar
write FShortOptionChar default DefShortOptionChar;
```

introductory character to be used for short options

LongOptionStart

```
public property LongOptionStart: String read FLongOptionString
write FLongOptionString;
```

introductory string to be used for long options

IncludeFileOptionName

```
public property IncludeFileOptionName: string read
FIncludeFileOptionName write FIncludeFileOptionName;
```

name of an option to include config file

IncludeFileOptionExpl `public property IncludeFileOptionExpl: string read FIncludeFileOptionExpl write FIncludeFileOptionExpl;`

explanation of an option to include config file

Fields

FParams `protected FParams: TStringList;`

FOptions `protected FOptions: TList;`

FLeftList `protected FLeftList: TStringList;`

FShortOptionChar `protected FShortOptionChar: Char;`

FLongOptionString `protected FLongOptionString: string;`

FIncludeFileOptionName `protected FIncludeFileOptionName: string;`

FIncludeFileOptionExpl `protected FIncludeFileOptionExpl: string;`

Methods

GetOption

Declaration `protected function GetOption(const AIndex: Integer): TOption;`

GetOptionsCount

Declaration `protected function GetOptionsCount: Integer;`

GetOptionByLongName

Declaration `protected function GetOptionByLongName(const AName: string): TOption;`

GetOptionByShortname

Declaration `protected function GetOptionByShortname(const AName: char): TOption;`

Create

Declaration `public constructor Create; virtual;`

Description Create without any options — this will parse the current command line

CreateParams

Declaration `public constructor CreateParams(const AParams: TStrings); virtual;`

Description Create with parameters to be used instead of command line

Destroy

Declaration `public destructor Destroy; override;`

Description destroy the option parser object and all associated `TOption`(16.4) objects

AddOption

Declaration `public function AddOption(const AOption: TOption): TOption;`

Description Add a `TOption`(16.4) descendant to be included in parsing the command line

ParseOptions

Declaration `public procedure ParseOptions;`

Description Parse the specified command line, see also `Create`(16.4)

WriteExplanations

Declaration `public procedure WriteExplanations;`

Description output explanations for all options to stdout, will nicely format the output and wrap explanations

16.5 Constants

DefShortOptionChar

Declaration `DefShortOptionChar = '-';`

Description default short option character used

DefLongOptionString

Declaration `DefLongOptionString = '--';`

Description default long option string used

OptionFileChar

Declaration `OptionFileChar = '@';`

Description Marks "include config file" option

CfgMacroCfgPath

Declaration `CfgMacroCfgPath = '$CFG_PATH';`

Description Special substitution that, if found inside a config file, will be replaced with actual path of the file

OptionIndent

Declaration `OptionIndent = ' ';`

Description Indentation of option's name from the start of console line

OptionSep

Declaration `OptionSep = ' ';`

Description Separator between option's name and explanation

ConsoleWidth

Declaration `ConsoleWidth = 80;`

Description Width of console

16.6 Author

Johannes Berg <johannes@sipsolutions.de>

Chapter 17

Unit PasDoc_Parser

17.1 Description

Parse Pascal code.

Contains the `TParser`(17.4) object, which can parse a Pascal code, and put the collected information into the TPasUnit instance.

17.2 Uses

- `SysUtils`
- `Classes`
- `Contnrs`
- `StrUtils`
- `PasDoc_Types`(29)
- `PasDoc_Items`(12)
- `PasDoc_Scanner`(20)
- `PasDoc_Tokenizer`(28)
- `PasDoc_StringPairVector`(24)
- `PasDoc_StringVector`(25)

17.3 Overview

`EInternalParserError Class` Raised when an impossible situation (indicating bug in pasdoc) occurs.

`TPasCioHelper Class` `TPasCioHelper` stores a CIO reference and current state.

`TPasCioHelperStack Class` A stack of `TPasCioHelper`(17.4) objects currently used to parse nested classes and records

`TRawDescriptionInfoList Class` `TRawDescriptionInfoList` stores a series of `TRawDescriptionInfos`(12.4).

`TParser Class` Parser class that will process a complete unit file and all of its include files, regarding directives.

17.4 Classes, Interfaces, Objects and Records

EInternalParserError Class

Hierarchy

EInternalParserError > Exception

Description

Raised when an impossible situation (indicating bug in pasdoc) occurs.

TPasCioHelper Class

Hierarchy

TPasCioHelper > TObject

Description

`TPasCioHelper` stores a CIO reference and current state.

Properties

Cio `public property Cio: TPasCio read FCio write FCio;`

CurVisibility `public property CurVisibility: TVisibility read FCurVisibility write FCurVisibility;`

Mode `public property Mode: TItemParseMode read FMode write FMode;`

SkipCioDecl `public property SkipCioDecl: Boolean read FSkipCioDecl write FSkipCioDecl;`

Methods

FreeAll

Declaration `public procedure FreeAll;`

Description Frees included objects and calls its own destructor. Objects are not owned by default.

TPasCioHelperStack Class

Hierarchy

TPasCioHelperStack > TObjectStack

Description

A stack of `TPasCioHelper`(17.4) objects currently used to parse nested classes and records

Methods

Clear

Declaration `public procedure Clear;`

Description Frees all items including their CIOs and clears the stack

Push

Declaration `public function Push(AHelper: TPasCioHelper): TPasCioHelper; inline;`

Pop

Declaration `public function Pop: TPasCioHelper; inline;`

Peek

Declaration `public function Peek: TPasCioHelper; inline;`

TRawDescriptionInfoList Class

Hierarchy

TRawDescriptionInfoList > TObject

Description

`TRawDescriptionInfoList` stores a series of `TRawDescriptionInfos`(12.4). It is modelled after TStringList but has only the minimum number of methods required for use in PasDoc.

Properties

Count `public property Count: integer read FCount;`

`Count` is the number of `TRawDescriptionInfos`(12.4) in `TRawDescriptionInfoList`.

Items `public property Items[Index: integer]: TRawDescriptionInfo read GetItems;`

`Items` provides read access to the `TRawDescriptionInfos`(12.4) in `TRawDescriptionInfoList`.

Methods

Append

Declaration `public function Append(Comment: TRawDescriptionInfo): integer;`

Description `Append` adds a new `TRawDescriptionInfo`(12.4) to `TRawDescriptionInfoList`.

Create

Declaration `public Constructor Create;`

TParser Class

Hierarchy

TParser > TObject

Description

Parser class that will process a complete unit file and all of its include files, regarding directives. When creating this object constructor `Create`(17.4) takes as an argument an input stream and a list of directives. Parsing work is done by calling `ParseUnitOrProgram`(17.4) method. If no errors appear, should return a `TPasUnit`(12.4) object with all information on the unit. Else exception is raised.
Things that parser inits in items it returns:

- Of every TPasItem : Name, RawDescription, Visibility, HintDirectives, DeprecatedNote, FullDeclararation (note: for now not all items get sensible FullDeclararation, but the intention is to improve this over time; see `TPasItem.FullDeclaration`(12.4) to know where FullDeclararation is available now).

  Note to IsDeprecated: parser inits it basing on hint directive "deprecated" presence in source file; it doesn't handle the fact that @deprecated tag may be specified inside RawDescription.

  Note to RawDescription: parser inits them from user's comments that preceded given item in source file. It doesn't handle the fact that @member and @value tags may also assign RawDescription for some item.

- Of TPasCio: Ancestors, Fields, Methods, Properties, MyType.
- Of TPasEnum: Members, FullDeclararation.
- Of TPasRoutine: What.
- Of TPasVarConst: FullDeclaration.
- Of TPasProperty: IndexDecl, FullDeclaration. PropType, NoDefault, Stored, DefaultValue, Reader, Writer. TODO: Parsing TPasProperty.DefaultInClass.
- Of TPasUnit; UsesUnits, Types, Variables, CIOs, Constants, FuncsProcs.

It doesn't init other values. E.g. AbstractDescription or DetailedDescription of TPasItem should be inited while expanding this item's tags. E.g. SourceFileDateTime and SourceFileName of TPasUnit must be set by other means.

Properties

OnMessage `public property OnMessage: TPasDocMessageEvent read FOnMessage write FOnMessage;`

CommentMarkers `public property CommentMarkers: TStringList read FCommentMarkers write SetCommentMarkers;`

MarkersOptional `public property MarkersOptional: boolean read fMarkersOptional write fMarkersOptional;`

IgnoreLeading `public property IgnoreLeading: string read FIgnoreLeading write FIgnoreLeading;`

IgnoreMarkers `public property IgnoreMarkers: TStringList read FIgnoreMarkers write SetIgnoreMarkers;`

ShowVisibilities `public property ShowVisibilities: TVisibilities read FShowVisibilities write FShowVisibilities;`

ImplicitVisibility `public property ImplicitVisibility: TImplicitVisibility read FImplicitVisibility write FImplicitVisibility;`

See command-line option --implicit-visibility documentation at –implicit-visibility documentation.

AutoBackComments `public property AutoBackComments: boolean read FAutoBackComments write FAutoBackComments;`

See command-line option --auto-back-comments documentation at –auto-back-comments documentation.

InfoMergeType `public property InfoMergeType: TInfoMergeType read FInfoMergeType write FInfoMergeType;`

Whether to read comments from the implementation, and how to merge them with the interface comments.

Methods

Create

Declaration `public constructor Create( const InputStream: TStream; const Directives: TStringVector; const IncludeFilePaths: TStringVector; const OnMessageEvent: TPasDocMessageEvent; const VerbosityLevel: Cardinal; const AStreamName, AStreamAbsoluteFileName: string; const AHandleMacros: boolean);`

Description Create a parser, initialize the scanner with input stream S. All strings in SD are defined compiler directives.

Destroy

Declaration `public destructor Destroy; override;`

Description Release all dynamically allocated memory.

ParseUnitOrProgram

Declaration `public procedure ParseUnitOrProgram(var U: TPasUnit);`

Description This does the real parsing work, creating U unit and parsing InputStream and filling all U properties.

17.5 Types

TItemParseMode

Declaration `TItemParseMode = (...);`

Description

Values `pmUndefined`

`pmConst`

`pmVar`

`pmType`

TOwnerItemType

Declaration `TOwnerItemType = (...);`

Description

Values `otUnit`

`otCio`

17.6 Authors

Ralf Junker (delphi@zeitungsjunge.de)
Marco Schmidt (marcoschmidt@geocities.com)
Johannes Berg <johannes@sipsolutions.de>
Michalis Kamburelis
Arno Garrels <first name.name@nospamgmx.de>

Chapter 18

Unit PasDoc_ProcessLineTalk

18.1 Description

Talking with another process through pipes.

18.2 Uses

- `SysUtils`
- `Classes`

18.3 Overview

`TTextReader Class` TTextReader reads given Stream line by line.

`TProcessLineTalk Class` This is a subclass of TProcess that allows to easy "talk" with executed process by pipes (read process stdout/stderr, write to process stdin) on a line-by-line basis.

18.4 Classes, Interfaces, Objects and Records

TTextReader Class

Hierarchy

TTextReader > TObject

Description

TTextReader reads given Stream line by line. Lines may be terminated in Stream with #13, #10, #13+#10 or #10+#13. This way I can treat any TStream quite like standard Pascal text files: I have simple Readln method.

After calling Readln or Eof you should STOP directly using underlying Stream (but you CAN use Stream right after creating TTextReader.Create(Stream) and before any Readln or Eof operations on this TTextReader).

Methods

CreateFromFileStream

Declaration `public constructor CreateFromFileStream(const FileName: string);`

Description This is a comfortable constructor, equivalent to TTextReader.Create(TFileStream.Create(FileName, fmOpenRead or fmShareDenyWrite), true)

Create

Declaration `public constructor Create(AStream: TStream; AOwnsStream: boolean);`

Description If AOwnsStream then in Destroy we will free Stream object.

Destroy

Declaration `public destructor Destroy; override;`

Readln

Declaration `public function Readln: string;`

Description Reads next line from Stream. Returned string does not contain any end-of-line characters.

Eof

Declaration `public function Eof: boolean;`

TProcessLineTalk Class

Hierarchy

TProcessLineTalk > TComponent

Description

This is a subclass of TProcess that allows to easy "talk" with executed process by pipes (read process stdout/stderr, write to process stdin) on a line-by-line basis.
If symbol HAS_PROCESS is not defined, this defines a junky implementation of TProcessLineTalk class that can't do anything and raises exception when you try to execute a process.

Properties

CommandLine `published property CommandLine: string read FCommandLine write FCommandLine;`

Executable `published property Executable: string read FExecutable write FExecutable;`

Parameters `published property Parameters: TStrings read FParameters;`

Methods

Execute

Declaration `public procedure Execute;`

WriteLine

Declaration `public procedure WriteLine(const S: string);`

ReadLine

Declaration `public function ReadLine: string;`

Create

Declaration `public constructor Create(AOwner: TComponent); override;`

Destroy

Declaration `public destructor Destroy; override;`

18.5 Authors

Michalis Kamburelis
Arno Garrels <first name.name@nospamgmx.de>

Chapter 19

Unit PasDoc_Reg

19.1 Description

Registers the PasDoc components into the IDE.

TODO: We have some properties in TPasDoc and generators components that should be registered with filename editors.

19.2 Overview

`Register` Registers the PasDoc components into the IDE.

19.3 Functions and Procedures

Register

Declaration `procedure Register;`

Description Registers the PasDoc components into the IDE.

19.4 Authors

Ralf Junker (delphi@zeitungsjunge.de)
Johannes Berg <johannes@sipsolutions.de>
Michalis Kamburelis

Chapter 20

Unit PasDoc_Scanner

20.1 Description

Scanner for Pascal, producing tokens and interpreting conditionals.

20.2 Uses

- SysUtils
- Classes
- Contnrs
- Types
- PasDoc_Types(29)
- PasDoc_Tokenizer(28)
- PasDoc_StringVector(25)
- PasDoc_StreamUtils(23)
- PasDoc_StringPairVector(24)
- PasDoc_ObjectVector(15)

20.3 Overview

ETokenizerStreamEnd Class

EInvalidIfCondition Class

TScanner Class Scanner for Pascal, producing tokens and interpreting conditionals
Returns tokens from a Pascal language source code input stream.

20.4 Classes, Interfaces, Objects and Records

ETokenizerStreamEnd Class

Hierarchy

ETokenizerStreamEnd > `EPasDoc`(29.4) > Exception

Description

no description available, EPasDoc description followsException raised in many situations when PasDoc encounters an error.

EInvalidIfCondition Class

Hierarchy

EInvalidIfCondition > `EPasDoc`(29.4) > Exception

Description

no description available, EPasDoc description followsException raised in many situations when PasDoc encounters an error.

TScanner Class

Hierarchy

TScanner > TObject

Description

Scanner for Pascal, producing tokens and interpreting conditionals
Returns tokens from a Pascal language source code input stream. Uses the `PasDoc_Tokenizer`(28) unit to get tokens, processes directives that might lead to

- including other files
- define / undefine symbols
- processes conditional directives
- handles FPC macros (when HandleMacros is true).

Effectively this is a combined tokenizer and pre-processor.
Single TScanner instance scans one unit using one or more `TTokenizer`(28.4) instances (to scan the unit and all nested include files).

Properties

IncludeFilePaths `public property IncludeFilePaths: TStringVector read FIncludeFilePaths write SetIncludeFilePaths;`

Paths to search for include files. When you assign something to this property it causes Assign(Value) call, not a real reference copy.

OnMessage `public property OnMessage: TPasDocMessageEvent read FOnMessage write FOnMessage;`

Verbosity `public property Verbosity: Cardinal read FVerbosity write FVerbosity;`

SwitchOptions `public property SwitchOptions: TSwitchOptions read FSwitchOptions;`

HandleMacros `public property HandleMacros: boolean read FHandleMacros;`

Methods

DoError

Declaration `protected procedure DoError(const AMessage: string; const AArguments: array of const);`

DoMessage

Declaration `protected procedure DoMessage(const AVerbosity: Cardinal; const MessageType: TPasDocMessageType; const AMessage: string; const AArguments: array of const);`

Create

Declaration `public constructor Create( const s: TStream; const OnMessageEvent: TPasDocMessageEvent; const VerbosityLevel: Cardinal; const AStreamName, AStreamAbsoluteFileName: string; const AHandleMacros: boolean);`

Description Creates a TScanner object that scans the given input stream.

Note that the stream S will be freed by this object (at destruction or when we will read all it's tokens), so after creating TScanner you should leave the stream to be managed completely by this TScanner.

Destroy

Declaration `public destructor Destroy; override;`

AddSymbol

Declaration `public procedure AddSymbol(const Name: string);`

Description Adds Name to the list of symbols (as a normal symbol, not macro).

AddSymbols

Declaration `public procedure AddSymbols(const NewSymbols: TStringVector);`

Description Adds all symbols in the NewSymbols collection by calling `AddSymbol`(20.4) for each of the strings in that collection.

AddMacro

Declaration `public procedure AddMacro(const Name, Value: string);`

Description Adds Name as a symbol that is a macro, that expands to Value.

ConsumeToken

Declaration `public procedure ConsumeToken;`

Description Gets next token and throws it away.

GetToken

Declaration `public function GetToken: TToken;`

Description Returns next token. Always non-nil (will raise exception in case of any problem).

GetStreamInfo

Declaration `public function GetStreamInfo: string;`

Description Returns the name of the file that is currently processed and the line number. Good for meaningful error messages.

PeekToken

Declaration `public function PeekToken: TToken;`

UnGetToken

Declaration `public procedure UnGetToken(var t: TToken);`

Description Place T in the buffer. Next time you will call GetToken you will get T. This also sets T to nil (because you shouldn't free T anymore after ungetting it). Note that the buffer has room only for 1 token, so you have to make sure that you will never unget more than two tokens. Practically, always call UnGetToken right after some GetToken.

20.5 Types

TUpperCaseLetter

Declaration `TUpperCaseLetter = 'A'..'Z';`

Description subrange type that has the 26 lower case letters from a to z

TSwitchOptions

Declaration `TSwitchOptions = array[TUpperCaseLetter] of Boolean;`

Description an array of boolean values, index type is `TUpperCaseLetter`(20.5)

TDirectiveType

Declaration `TDirectiveType = (...);`

Description All directives a scanner is going to regard.

Values
`DT_UNKNOWN`
`DT_DEFINE`
`DT_ELSE`
`DT_ENDIF`
`DT_IFDEF`
`DT_IFNDEF`
`DT_IFOPT`
`DT_INCLUDE_FILE`
`DT_UNDEF`
`DT_INCLUDE_FILE_2`
`DT_IF`
`DT_ELSEIF`
`DT_IFEND`

20.6 Constants

MAX_TOKENIZERS

Declaration `MAX_TOKENIZERS = 32;`

Description maximum number of streams we can recurse into; first one is the unit stream, any other stream an include file; current value is 32, increase this if you have more include files recursively including others

20.7 Authors

Johannes Berg <johannes@sipsolutions.de>
Ralf Junker (delphi@zeitungsjunge.de)
Marco Schmidt (marcoschmidt@geocities.com)
Michalis Kamburelis
Arno Garrels <first name.name@nospamgmx.de>

Chapter 21

Unit PasDoc_Serialize

21.1 Description

Serializing / deserializing cached information.

21.2 Uses

- `Classes`
- `SysUtils`
- `PasDoc_StreamUtils`(23)

21.3 Overview

`EInvalidCacheFileVersion Class`

`TSerializable Class`

`ESerializedException Class`

21.4 Classes, Interfaces, Objects and Records

EInvalidCacheFileVersion Class

Hierarchy

EInvalidCacheFileVersion > Exception

TSerializable Class

Hierarchy

TSerializable > TObject

Properties

WasDeserialized `public property WasDeserialized: boolean read FWasDeserialized;`

Methods

Serialize

Declaration `protected procedure Serialize(const ADestination: TStream); virtual;`

Deserialize

Declaration `protected procedure Deserialize(const ASource: TStream); virtual;`

Read7BitEncodedInt

Declaration `public class function Read7BitEncodedInt(const ASource: TStream): Integer;`

Write7BitEncodedInt

Declaration
```
public class procedure Write7BitEncodedInt(Value: Integer; const
ADestination: TStream);
```

LoadStringFromStream

Declaration
```
public class function LoadStringFromStream(const ASource: TStream):
string;
```

SaveStringToStream

Declaration
```
public class procedure SaveStringToStream(const AValue: string; const
ADestination: TStream);
```

LoadDoubleFromStream

Declaration
```
public class function LoadDoubleFromStream(const ASource: TStream):
double;
```

SaveDoubleToStream

Declaration
```
public class procedure SaveDoubleToStream(const AValue: double; const
ADestination: TStream);
```

LoadIntegerFromStream

Declaration `public class function LoadIntegerFromStream(const ASource: TStream): Longint;`

SaveIntegerToStream

Declaration `public class procedure SaveIntegerToStream(const AValue: Longint; const ADestination: TStream);`

Create

Declaration `public constructor Create; virtual;`

SerializeObject

Declaration `public class procedure SerializeObject(const AObject: TSerializable; const ADestination: TStream);`

DeserializeObject

Declaration `public class function DeserializeObject(const ASource: TStream): TSerializable;`

Register

Declaration `public class procedure Register(const AClass: TSerializableClass);`

SerializeToFile

Declaration `public procedure SerializeToFile(const AFileName: string);`

DeserializeFromFile

Declaration `public class function DeserializeFromFile(const AFileName: string): TSerializable;`

Description Read back from file.

Exceptions `EInvalidCacheFileVersion`**(21.4)** When the cached file contents are from an old pasdoc version (or invalid).

ESerializedException Class

Hierarchy

ESerializedException > Exception

21.5 Types

TSerializableClass

Declaration `TSerializableClass = class of TSerializable;`

21.6 Author

Arno Garrels <first name.name@nospamgmx.de>

Chapter 22

Unit PasDoc_SortSettings

22.1 Description

Sorting settings types and names.

22.2 Uses

- SysUtils

22.3 Overview

EInvalidSortSetting Class

SortSettingFromName

SortSettingsToName Comma-separated list

22.4 Classes, Interfaces, Objects and Records

EInvalidSortSetting Class

Hierarchy

EInvalidSortSetting > Exception

22.5 Functions and Procedures

SortSettingFromName

Declaration `function SortSettingFromName(const SortSettingName: string): TSortSetting;`

Description

Exceptions `EInvalidSortSetting`**(22.4)** if ASortSettingName does not match (case ignored) to any SortSettingNames.

SortSettingsToName

Declaration `function SortSettingsToName(const SortSettings: TSortSettings): string;`

Description Comma-separated list

22.6 Types

TSortSetting

Declaration `TSortSetting = (...);`

Description

Values
`ssCIOs`
`ssConstants`
`ssFuncsProcs`
`ssTypes`
`ssVariables`
`ssUsesClauses`
`ssRecordFields`
`ssNonRecordFields`
`ssMethods`
`ssProperties`

TSortSettings

Declaration `TSortSettings = set of TSortSetting;`

22.7 Constants

AllSortSettings

Declaration
```
AllSortSettings: TSortSettings = [Low(TSortSetting) ..
High(TSortSetting)];
```

SortSettingNames

Declaration
```
SortSettingNames: array[TSortSetting] of string = ( 'structures',
'constants', 'functions', 'types', 'variables', 'uses-clauses',
'record-fields', 'non-record-fields', 'methods', 'properties' );
```

Description Must be lowercase. Used in `SortSettingsToName`(22.5), `SortSettingFromName`(22.5).

Chapter 23

Unit PasDoc_StreamUtils

23.1 Description

A few stream utility functions.
TBufferedStream, TStreamReader and TStreamWriter by Arno Garrels.

23.2 Uses

- `SysUtils`
- `Classes`
- `PasDoc_Types`(29)

23.3 Overview

`TBufferedStream Class`

`StreamReadLine`

`StreamWriteLine` Write AString contents, then LineEnding to AStream

`StreamWriteString` Just write AString contents to AStream

23.4 Classes, Interfaces, Objects and Records

TBufferedStream Class

Hierarchy

TBufferedStream > TStream

Properties

IsReadOnly `public property IsReadOnly: Boolean read FIsReadOnly write SetIsReadOnly;`

Set IsReadOnly if you are sure you will never write to the stream and nobody else will do, this speeds up getter Size and in turn Seeks as well. IsReadOnly is set to TRUE if a constructor with filename is called with a read only mode and a share lock.

FastSize `public property FastSize: Int64 read GetSize;`

Methods

SetIsReadOnly

Declaration `protected procedure SetIsReadOnly(const Value: Boolean);`

Description See property IsReadOnly below

SetSize

Declaration `protected procedure SetSize(NewSize: Integer); override;`

SetSize

Declaration `protected procedure SetSize(const NewSize: Int64); override;`

InternalGetSize

Declaration `protected function InternalGetSize: Int64; inline;`

GetSize

Declaration `protected function GetSize: Int64; override;`

Init

Declaration `protected procedure Init; virtual;`

FillBuffer

Declaration `protected function FillBuffer: Boolean; inline;`

Create

Declaration `public constructor Create; overload;`

Create

Declaration `public constructor Create(Stream : TStream; BufferSize : Integer = DEFAULT_BUFSIZE; OwnsStream : Boolean = FALSE); overload; virtual;`

Description Dummy, don't call!

Create

Declaration `public constructor Create(const FileName : String; Mode : Word; BufferSize : Integer = DEFAULT_BUFSIZE); overload; virtual;`

Destroy

Declaration `public destructor Destroy; override;`

Flush

Declaration `public procedure Flush; inline;`

Read

Declaration `public function Read(var Buffer; Count: Integer): Integer; override;`

Seek

Declaration `public function Seek(Offset: Integer; Origin: Word): Integer; override;`

Seek

Declaration `public function Seek(const Offset: Int64; Origin: TSeekOrigin): Int64; override;`

Write

Declaration `public function Write(const Buffer; Count: Integer): Integer; override;`

23.5 Functions and Procedures

StreamReadLine

Declaration `function StreamReadLine(const AStream: TStream): AnsiString;`

StreamWriteLine

Declaration `procedure StreamWriteLine(const AStream: TStream; const AString: AnsiString);`

Description Write AString contents, then LineEnding to AStream

StreamWriteString

Declaration `procedure StreamWriteString(const AStream: TStream; const AString: AnsiString);`

Description Just write AString contents to AStream

23.6 Constants

DEFAULT_BUFSIZE

Declaration `DEFAULT_BUFSIZE = 4096;`

MIN_BUFSIZE

Declaration `MIN_BUFSIZE = 128;`

MAX_BUFSIZE

Declaration `MAX_BUFSIZE = 1024 * 64;`

23.7 Authors

Johannes Berg <johannes@sipsolutions.de>
Arno Garrels <first name.name@nospamgmx.de>

Chapter 24

Unit PasDoc_StringPairVector

24.1 Description

Simple container for a pair of strings.

24.2 Uses

- Classes
- Contnrs
- PasDoc_ObjectVector(15)

24.3 Overview

TStringPair Class

TStringPairVector Class List of string pairs.

24.4 Classes, Interfaces, Objects and Records

TStringPair Class

Hierarchy

TStringPair > TObject

Fields

Name public Name: string;

Value public Value: string;

Data public Data: Pointer;

Methods

CreateExtractFirstWord

Declaration `public constructor CreateExtractFirstWord(const S: string);`

Description Init Name and Value by `ExtractFirstWord`(30.5) from S.

Create

Declaration `public constructor Create; overload;`

Create

Declaration `public constructor Create(const AName, AValue: string; AData: Pointer = nil); overload;`

TStringPairVector Class

Hierarchy

TStringPairVector > `TObjectVector`(15.4) > TObjectList

Description

List of string pairs. This class contains only non-nil objects of class TStringPair.
Using this class instead of TStringList (with it's Name and Value properties) is often better, because this allows both Name and Value of each pair to safely contain any special characters (including '=' and newline markers). It's also faster, since it doesn't try to encode Name and Value into one string.

Properties

Items `public property Items[i: Integer]: TStringPair read GetItems write SetItems;`

Methods

Text

Declaration `public function Text(const NameValueSepapator, ItemSeparator: string): string;`

Description Returns all items Names and Values glued together. For every item, string Name + NameValueSepapator + Value is constructed. Then all such strings for every items all concatenated with ItemSeparator.

Remember that the very idea of `TStringPair`(24.4) and `TStringPairVector`(24.4) is that Name and Value strings may contain any special characters, including things you give here as NameValueSepapator and ItemSeparator. So it's practically impossible to later convert such Text back to items and Names/Value pairs.

FindName

Declaration `public function FindName(const Name: string; IgnoreCase: boolean = true): Integer;`

Description Finds a string pair with given Name. Returns -1 if not found.

DeleteName

Declaration `public function DeleteName(const Name: string; IgnoreCase: boolean = true): boolean;`

Description Removes first string pair with given Name. Returns if some pair was removed.

LoadFromBinaryStream

Declaration `public procedure LoadFromBinaryStream(Stream: TStream);`

Description Load from a stream using the binary format. For each item, it's Name and Value are saved. (TStringPair.Data pointers are *not* saved.)

SaveToBinaryStream

Declaration `public procedure SaveToBinaryStream(Stream: TStream);`

Description Save to a stream, in a format readable by `LoadFromBinaryStream`(24.4).

FirstName

Declaration `public function FirstName: string;`

Description Name of first item, or '' if list empty.

Chapter 25

Unit PasDoc_StringVector

25.1 Description

String vector based on TStringList.

The string vector is based on TStringList and simply exports a few extra functions - I did this so I didn't have to change so much old code, this has only little additional functionality

25.2 Uses

- `Classes`

25.3 Overview

`TStringVector Class`

`NewStringVector`

`IsEmpty`

25.4 Classes, Interfaces, Objects and Records

TStringVector Class

Hierarchy

TStringVector > TStringList

Methods

FirstName

Declaration `public function FirstName:  string;`

Description This is the same thing as Items[0]

LoadFromTextFileAdd

Declaration `public procedure LoadFromTextFileAdd(const AFilename: string); overload;`

LoadFromTextFileAdd

Declaration `public procedure LoadFromTextFileAdd(var ATextFile: TextFile); overload;`

RemoveAllNamesCI

Declaration `public procedure RemoveAllNamesCI(const AName: string);`

ExistsNameCI

Declaration `public function ExistsNameCI(const AName: string): boolean;`

IsEmpty

Declaration `public function IsEmpty: boolean;`

AddNotExisting

Declaration `public function AddNotExisting(const AString: string): Integer;`

LoadFromBinaryStream

Declaration `public procedure LoadFromBinaryStream(Stream: TStream);`

Description Load from a stream using the binary format.

The binary format is

- Count
- followed by each string, loaded using `TSerializable.LoadStringFromStream`(21.4).

Note that you should never use our Text value to load/save this object from/into a stream, like `Text := TSerializable.LoadStringFromStream(Stream)`. Using and assigning to the Text value breaks when some strings have newlines inside that should be preserved.

SaveToBinaryStream

Declaration `public procedure SaveToBinaryStream(Stream: TStream);`

Description Save to a stream, in a format readable by `LoadFromBinaryStream`(25.4).

25.5 Functions and Procedures

NewStringVector

Declaration `function NewStringVector: TStringVector;`

IsEmpty

Declaration `function IsEmpty(const AOV: TStringVector): boolean; overload;`

25.6 Authors

Johannes Berg <johannes@sipsolutions.de>
Michalis Kamburelis

Chapter 26

Unit PasDoc_TagManager

26.1 Description

Collects information about available @-tags and can parse text with tags.

26.2 Uses

- SysUtils
- Classes
- Contnrs
- PasDoc_Types(29)
- PasDoc_ObjectVector(15)

26.3 Overview

TTag Class

TTopLevelTag Class

TNonSelfTag Class

TTagVector Class All Items of this list must be non-nil TTag objects.

TTagManager Class

26.4 Classes, Interfaces, Objects and Records

TTag Class

Hierarchy

TTag > TObject

Properties

TagOptions `public property TagOptions: TTagOptions read FTagOptions write FTagOptions;`

TagManager `public property TagManager: TTagManager read FTagManager;`

TagManager that will recognize and handle this tag. Note that the tag instance is owned by this tag manager (i.e. it will be freed inside this tag manager). It can be nil if no tag manager currently owns this tag.

Note that it's very useful in `Execute`(26.4) or `OnExecute`(26.4) implementations.

E.g. you can use it to report a message by `TagManager.DoMessage(...)`, this is e.g. used by implementation of TPasItem.StoreAbstractTag.

You could also use this to manually force recursive behavior of a given tag. I.e let's suppose that you have a tag with TagOptions = [toParameterRequired], so the TagParameter parameter passed to handler was not recursively expanded. Then you can do inside your handler `NewTagParameter := TagManager.Execute(TagParameter, ...)` and this way you have explicitly recursively expanded the tag.

Scenario above is actually used in implementation of @noAutoLink tag. There I call TagManager.Execute with parameter `AutoLink` set to false thus preventing auto-linking inside text within @noAutoLink.

Name `public property Name: string read FName write FName;`

Name of the tag, that must be specified by user after the "@" sign. Value of this property must always be lowercase.

OnPreExecute `public property OnPreExecute: TTagExecuteEvent read FOnPreExecute write FOnPreExecute;`

OnExecute `public property OnExecute: TTagExecuteEvent read FOnExecute write FOnExecute;`

OnAllowedInside `public property OnAllowedInside: TTagAllowedInsideEvent read FOnAllowedInside write FOnAllowedInside;`

Methods

Create

Declaration `public constructor Create(ATagManager: TTagManager; const AName: string; AOnPreExecute: TTagExecuteEvent; AOnExecute: TTagExecuteEvent; const ATagOptions: TTagOptions);`

Description Note that AName will be converted to lowercase before assigning to Name.

PreExecute

Declaration `public procedure PreExecute(var ThisTagData: TObject; EnclosingTag: TTag; var EnclosingTagData: TObject; const TagParameter: string; var ReplaceStr: string); virtual;`

Description This is completely analogous to `Execute`(26.4) but used when `TTagManager.PreExecute`(26.4) is `True`. In this class this simply calls `OnPreExecute`(26.4).

Execute

Declaration `public procedure Execute(var ThisTagData: TObject; EnclosingTag: TTag; var EnclosingTagData: TObject; const TagParameter: string; var ReplaceStr: string); virtual;`

Description This will be used to do main work when this @-tag occured in description.

EnclosingTag parameter specifies enclosing tag. This is useful for tags that must behave differently in different contexts, e.g. in plain-text output @item tag will behave differently inside @orderedList and @unorderedList. EnclosingTag is nil when the tag occured at top level of the description.

ThisTagData and EnclosingTagData form a mechanism to pass arbitraty data between child tags enclosed within one parent tag. Example uses:

- This is the way for multiple @item tags inside @orderedList tag to count themselves (to provide list item numbers, for pasdoc output formats that can't automatically number list items).
- This is the way for @itemSpacing tag to communicate with enclosing @orderedList tag to specify list style.
- And this is the way for @cell tags to be collected inside rows data and then @rows tags to be collected inside table data. Thanks to such collecting `TDocGenerator.FormatTable`(4.4) receives at once all information about given table, and can use it to format table.

How does this XxxTagData mechanism work:

When we start parsing parameter of some tag with toRecursiveTags, we create a new pointer inited to `CreateOccurenceData`(26.4). When @-tags occur inside this parameter, we pass them this pointer as EnclosingTagData (this way all @-tags with the same parent can use this pointer to communicate with each other). At the end, when parameter was parsed, we call given tag's Execute method passing the resulting pointer as ThisTagData (this way @-tags with the same parent can use this pointer to pass some data to their parent).

In this class this method simply calls `OnExecute`(26.4) (if assigned).

AllowedInside

Declaration `public function AllowedInside(EnclosingTag: TTag): boolean; virtual;`

Description This will be checked always when this tag occurs within description. Given EnclosingTag is enclosing tag, nil if we're in top level. If this returns false then this tag will not be allowed inside EnclosingTag.

In this class this method

1. Assumes that Result = true if we're at top level or EnclosingTag.TagOptions contains toAllowOtherTagsInsideByDefault. Else it assumes Result = false.
2. Then it calls `OnAllowedInside(Self, EnclosingTag, Result)`(26.4) (if OnAllowedInside is assigned).

CreateOccurenceData

Declaration `public function CreateOccurenceData: TObject; virtual;`

Description In this class this simply returns `Nil`.

DestroyOccurenceData

Declaration `public procedure DestroyOccurenceData(Value: TObject); virtual;`

Description In this class this simply does `Value.Free`.

TTopLevelTag Class

Hierarchy

TTopLevelTag > `TTag`(26.4) > TObject

Methods

AllowedInside

Declaration `public function AllowedInside(EnclosingTag: TTag): boolean; override;`

Description This returns just `EnclosingTag = nil`.

Which means that this tag is allowed only at top level of description, never inside parameter of some tag.

TNonSelfTag Class

Hierarchy

TNonSelfTag > `TTag`(26.4) > TObject

Methods

AllowedInside

Declaration `public function AllowedInside(EnclosingTag: TTag): boolean; override;`

Description This returns just `inherited and (EnclosingTag <> Self)`.

Which means that (assuming that `OnAllowedInside`(26.4) is not assigned) this tag is allowed at top level of description and inside parameter of any tag *but not within itself and not within tags without toAllowOtherTagsInsideByDefault*.

This is currently not used by any tag.

TTagVector Class

Hierarchy

TTagVector > `TObjectVector`(15.4) > TObjectList

Description

All Items of this list must be non-nil TTag objects.

Methods

FindByName

Declaration `public function FindByName(const Name: string): TTag;`

Description Case of Name does *not* matter (so don't bother converting it to lowercase or something like that before using this method). Returns nil if not found.

Maybe in the future it will use hashlist, for now it's not needed.

TTagManager Class

Hierarchy

TTagManager > TObject

Properties

OnMessage `public property OnMessage: TPasDocMessageEvent read FOnMessage write FOnMessage;`

This will be used to print messages from within `Execute`(26.4).

Note that in this unit we essentialy "don't know" that parsed Description string is probably attached to some TPasItem. It's good that we don't know it (because it makes this class more flexible). But it also means that OnMessage that you assign here may want to add to passed AMessage something like + ' (Expanded_TPasItem_Name)', see e.g. TDocGenerator.DoMessageFromExpandDescription. Maybe in the future we will do some descendant of this class, like TTagManagerForPasItem.

Paragraph `public property Paragraph: string read FParagraph write FParagraph;`

This will be inserted on paragraph marker (two consecutive newlines, see wiki page WritingDocumentation) in the text. This should specify how paragraphs are marked in particular output format, e.g. html generator may set this to '<p>'.

Default value is ' ' (one space).

Space `public property Space: string read FSpace write FSpace;`

This will be inserted on each whitespace sequence (but not on paragraph break). This is consistent with WritingDocumentation that clearly says that "amount of whitespace does not matter".

Although in some pasdoc output formats amount of whitespace also does not matter (e.g. HTML and LaTeX) but in other (e.g. plain text) it matters, so such space compression is needed. In other output formats (no examples yet) it may need to be expressed by something else than simple space, that's why this property is exposed.

Default value is ' ' (one space).

ShortDash `public property ShortDash: string read FShortDash write FShortDash;`

This will be inserted on `@-` in description, and on a normal single dash in description that is not a part of en-dash or em-dash. This should produce just a short dash.

Default value is '-'.

You will never get any '-' character to be converted by ConvertString. Convertion of '-' is controlled solely by XxxDash properties of tag manager.

See also `EnDash`**(26.4)** This will be inserted on `--` in description.

`EmDash`**(26.4)** This will be inserted on `---` in description.

EnDash `public property EnDash: string read FEnDash write FEnDash;`

This will be inserted on `--` in description. This should produce en-dash (as in LaTeX). Default value is '--'.

EmDash `public property EmDash: string read FEmDash write FEmDash;`

This will be inserted on `---` in description. This should produce em-dash (as in LaTeX). Default value is '---'.

URLLink `public property URLLink: TStringConverter read FURLLink write FURLLink;`

This will be called from `Execute`(26.4) when URL will be found in Description. Note that passed here URL will *not* be processed by `ConvertString`(26.4).

This tells what to put in result on URL. If this is not assigned, then ConvertString(URL) will be appended to Result in `Execute`(26.4).

OnTryAutoLink `public property OnTryAutoLink: TTryAutoLinkEvent read FOnTryAutoLink write FOnTryAutoLink;`

This should check does QualifiedIdentifier looks like a name of some existing identifier. If yes, sets AutoLinked to true and sets QualifiedIdentifierReplacement to a link to

QualifiedIdentifier (QualifiedIdentifierReplacement should be ready to be put in final documentation, i.e. already in the final output format). By default AutoLinked is false.

ConvertString `public property ConvertString: TStringConverter read FConvertString write FConvertString;`

Abbreviations `public property Abbreviations: TStringList read FAbbreviations write FAbbreviations;`

PreExecute `public property PreExecute: boolean read FPreExecute write FPreExecute;`

When `PreExecute` is `True`, tag manager will work a little differently than usual:

- Instead of `TTag.Execute`(26.4), `TTag.PreExecute`(26.4) will be called.
- Various warnings will *not* be reported.
  Assumption is that you will later process the same text with `PreExecute` set to `False` to get all the warnings.
- AutoLink will not be used (like it was always false). Also the result of `Execute`(26.4) will be pretty much random and meaningless (so you should ignore it). Also this means that the TagParameter for tags with toRecursiveTags should be ignored, because it will be something incorrect. This means that only tags without toRecursiveTags should actually use TagParameter in their OnPreExecute handlers.
  Assumption is that you actually don't care about the result of `Execute`(26.4) methods, and you will later process the same text with `PreExecute` set to `False` to get the proper output.
  The goal is to make execution with PreExecute set to `True` as fast as possible.

Markdown `public property Markdown: boolean read FMarkdown write FMarkdown default false;`

When `Markdown` is `True`, Markdown syntax is considered

Methods

Create

Declaration `public constructor Create;`

Destroy

Declaration `public destructor Destroy; override;`

DoMessage

Declaration `public procedure DoMessage(const AVerbosity: Cardinal; const MessageType: TPasDocMessageType; const AMessage: string; const AArguments: array of const);`

Description Call OnMessage (if assigned) with given params.

DoMessageNonPre

Declaration `public procedure DoMessageNonPre(const AVerbosity: Cardinal; const MessageType: TPasDocMessageType; const AMessage: string; const AArguments: array of const);`

Description Call `DoMessage`(26.4) only if `PreExecute`(26.4) is `False`.

Execute

Declaration `public function Execute(const Description: string; AutoLink: boolean; WantFirstSentenceEnd: boolean; out FirstSentenceEnd: Integer): string; overload;`

Description This method is the very essence of this class and this unit. It expands Description, which means that it processes Description (text supplied by user in some comment in parsed unit) into something ready to be included in output documentation. This means that this handles parsing @-tags, inserting paragraph markers, recognizing URLs in Description and correctly translating it, and translating rest of the "normal" text via ConvertString.

If WantFirstSentenceEnd then we will look for '.' char followed by any whitespace in Description. Moreover, this '.' must be outside of any @-tags parameter. Under FirstSentenceEnd we will return the number of beginning characters *in the output string* that will include correspong '.' character (note that this definition takes into account that ConvertString may translate '.' into something longer). If no such character exists in Description, FirstSentenceEnd will be set to Length(Result), so the whole Description will be treated as it's first sentence.

If WantFirstSentenceEnd, FirstSentenceEnd will not be set.

Execute

Declaration `public function Execute(const Description: string; AutoLink: boolean): string; overload;`

Description This is equivalent to Execute(Description, AutoLink, false, Dummy)

CoreExecute

Declaration `public function CoreExecute(const Description: string; AutoLink: boolean; EnclosingTag: TTag; var EnclosingTagData: TObject; WantFirstSentenceEnd: boolean; out FirstSentenceEnd: Integer): string; overload;`

Description This is the underlying version of Execute. Use with caution!

If EnclosingTag = nil then this is understood to be toplevel of description, which means that all tags are allowed inside.

If EnclosingTag <> nil then this is not toplevel.

EnclosingTagData returns collected data for given EnclosingTag. You should init it to EnclosingTag.CreateOccurenceData. It will be passed as EnclosingTagData to each of @-tags found inside Description.

CoreExecute

Declaration `public function CoreExecute(const Description: string; AutoLink: boolean; EnclosingTag: TTag; var EnclosingTagData: TObject): string; overload;`

26.5 Types

TTagExecuteEvent

Declaration `TTagExecuteEvent = procedure(ThisTag: TTag; var ThisTagData: TObject; EnclosingTag: TTag; var EnclosingTagData: TObject; const TagParameter: string; var ReplaceStr: string) of object;`

Description

See also `TTag.Execute`**(26.4)** This will be used to do main work when this @-tag occured in description.

TTagAllowedInsideEvent

Declaration `TTagAllowedInsideEvent = procedure( ThisTag: TTag; EnclosingTag: TTag; var Allowed: boolean) of object;`

Description

See also `TTag.AllowedInside`**(26.4)** This will be checked always when this tag occurs within description.

TStringConverter

Declaration `TStringConverter = function(const s: string): string of object;`

TTagOption

Declaration `TTagOption = (...);`

Description

Values `toParameterRequired` This means that tag expects parameters. If this is not included in TagOptions then tag should not be given any parameters, i.e. TagParameter passed to `TTag.Execute`(26.4) should be ". We will display a warning if user will try to give some parameters for such tag.

`toRecursiveTags` This means that parameters of this tag will be expanded before passing them to `TTag.Execute`(26.4). This means that we will expand recursive tags inside parameters, that we will ConvertString inside parameters, that we will handle paragraphs inside parameters etc. — all that does `TTagManager.Execute`(26.4).

If toParameterRequired is not present in TTagOptions then it's not important whether you included toRecursiveTags.

It's useful for some tags to include toParameterRequired without including toRecursiveTags, e.g. @longcode or @html, that want to get their parameters "verbatim", not processed.

If toRecursiveTags is not included in tag options: Then *everything* is allowed within parameter of this tag, but nothing is interpreted. E.g. you can freely use @ char, and even write various @-tags inside @html tag — this doesn't matter, because @-tags will not be interpreted (they will not be even searched !) inside @html tag. In other words, @ character means literally "@" inside @html, nothing more. The only exception are double @@, @(and @): we still treat them specially, to allow escaping the default parenthesis matching rules. Unless toRecursiveTagsManually is present.

`toRecursiveTagsManually` Use this, instead of toRecursiveTags, if the implementation of your tag calls (always!) TagManager.CoreExecute on given TagParameter. This means that your tag is expanded recursively (it handles -tags inside), but you do it manually (instead of allowing toRecursiveTags to do the job). In this case, TagParameter given will be really absolutely unmodified (even the special @@, @(and @) will not be handled), because we know that it will be handled later by special CoreExecute call.

Never use both flags toRecursiveTags and toRecursiveTagsManually.

`toAllowOtherTagsInsideByDefault` This is meaningful only if toRecursiveTags is included. Then toAllowOtherTagsInsideByDefault determines are other tags allowed by the default implementation of `TTag.AllowedInside`(26.4).

`toAllowNormalTextInside` This is meaningful only if toRecursiveTags is included. Then `toAllowNormalTextInside` says that normal text is allowed inside parameter of this tag. *"Normal text"* is anything except other @-tags: normal text, paragraph breaks, various dashes, URLs, and literal @ character (expressed by @@ in descriptions).

If `toAllowNormalTextInside` will not be included, then normal text (not enclosed within other @-tags) will not be allowed inside. Only whitespace will be allowed, and it will be ignored anyway (i.e. will not be passed to ConvertString, empty line will not produce any Paragraph etc.). This is useful for tags like @orderedList that should only contain other @item tags inside.

`toFirstWordVerbatim` This is useful for tags like @raises and @param that treat 1st word of their descriptions very specially (where "what exactly is the 1st word" is defined by the `ExtractFirstWord`(30.5) function). This tells pasdoc to leave the beginning of tag parameter (the first word and the eventual whitespace before it) as it is in the parameter. Don't search there for @-tags, URLs, -- or other special dashes, don't insert paragraphs, don't try to auto-link it.

This is meaningful only if toRecursiveTags is included (otherwise the whole tag parameters are always preserved "verbatim").

TODO: in the future TTagExecuteEvent should just get this "first word" as a separate parameter, separated from TagParameters. Also, this word should not be converted by ConvertString.

TTagOptions

Declaration `TTagOptions = set of TTagOption;`

TTryAutoLinkEvent

Declaration

```
TTryAutoLinkEvent = procedure(TagManager:  TTagManager; const
QualifiedIdentifier:  TNameParts; out QualifiedIdentifierReplacement:
string; var AutoLinked:  boolean) of object;
```

Chapter 27

Unit PasDoc_Tipue

27.1 Description

Integrate Tipue search.

27.2 Uses

- `PasDoc_Utils`(30)
- `PasDoc_Items`(12)
- `Contnrs`

27.3 Overview

`TipueSearchButtonHead` Put this in <head> of every page with search button.

`TipueSearchButton` Put this at a place where Tipue button should appear.

`TipueAddFiles` Adds some additional files to html documentation, needed for tipue engine.

27.4 Functions and Procedures

TipueSearchButtonHead

Declaration `function TipueSearchButtonHead: string;`

Description Put this in <head> of every page with search button.

TipueSearchButton

Declaration `function TipueSearchButton: string;`

Description Put this at a place where Tipue button should appear. It will make a form with search button. You will need to use Format to insert the localized word for "Search", e.g.: Format(TipueSearchButton, ['Search']) for English.

TipueAddFiles

Declaration `procedure TipueAddFiles(Units: TPasUnits; const Introduction, Conclusion: TExternalItem; const AdditionalFiles: TExternalItemList; const Head, BodyBegin, BodyEnd: string; const LanguageCode: string; const OutputPath: string);`

Description Adds some additional files to html documentation, needed for tipue engine.

OutputPath is our output path, where html output must be placed. Must end with PathDelim.

Units must be non-nil. It will be used to generate index data for tipue.

Chapter 28

Unit PasDoc_Tokenizer

28.1 Description

Simple Pascal tokenizer.

The `TTokenizer`(28.4) object creates `TToken`(28.4) objects (tokens) for the Pascal programming language from a character input stream.
The `PasDoc_Scanner`(20) unit does the same (it actually uses this unit's tokenizer), with the exception that it evaluates compiler directives, which are comments that start with a dollar sign.

28.2 Uses

- `Classes`
- `PasDoc_Utils`(30)
- `PasDoc_Types`(29)
- `PasDoc_StreamUtils`(23)

28.3 Overview

`TToken Class` Stores the exact type and additional information on one token.

`TTokenizer Class` Converts an input TStream to a sequence of `TToken`(28.4) objects.

`StandardDirectiveByName` Checks is Name (case ignored) some Pascal keyword.

`KeyWordByName` Checks is Name (case ignored) some Pascal standard directive.

28.4 Classes, Interfaces, Objects and Records

TToken Class

Hierarchy

TToken > TObject

Description

Stores the exact type and additional information on one token.

Properties

StreamName `public property StreamName: string read FStreamName;`

Informative to user name of the stream from which this token was read. This can be a filename (relative or absolute, however user specified it), but it also can be something arbitrary like "$if / $elseif condition". So don't treat it like a reliable filename.

It is currently used to set `TRawDescriptionInfo.StreamName`(12.4).

StreamAbsoluteFileName `public property StreamAbsoluteFileName: string read FStreamAbsoluteFileName;`

Filename, always absolute, of the underlying file of this stream. Empty (") if this is not a file stream.

BeginPosition `public property BeginPosition: Int64 read FBeginPosition;`

`BeginPosition` is the position in the stream of the start of the token. It is currently used to set `TRawDescriptionInfo.BeginPosition`(12.4).

EndPosition `public property EndPosition: Int64 read FEndPosition;`

`EndPosition` is the position in the stream of the character immediately after the end of the token. It is currently used to set `TRawDescriptionInfo.EndPosition`(12.4).

Line `public property Line: Integer read FLine;`

Line number (1-based) in the stream where this token starts.

Fields

Data `public Data: string;`

the exact character representation of this token as it was found in the input file

MyType `public MyType: TTokenType;`

the type of this token as `TTokenType`(28.6)

Info `public Info: record`

additional information on this token as a variant record depending on the token's MyType

CommentContent `public CommentContent: string;`

Contents of a comment token. This is defined only when MyType is in TokenCommentTypes or is TOK_DIRECTIVE. This is the text within the comment *without* comment delimiters. For TOK_DIRECTIVE you can safely assume that CommentContent[1] = '$'.

StringContent `public StringContent: string;`

Contents of the string token, that is: the value of the string literal. D only when MyType is TOK_STRING.

Methods

Create

Declaration `public constructor Create(const TT: TTokenType);`

Description Create a token of and assign the argument token type to `MyType`(28.4)

GetTypeName

Declaration `public function GetTypeName: string;`

IsSymbol

Declaration `public function IsSymbol(const ASymbolType: TSymbolType): Boolean;`

Description Does `MyType`(28.4) is TOK_SYMBOL and Info.SymbolType is ASymbolType ?

IsKeyWord

Declaration `public function IsKeyWord(const AKeyWord: TKeyWord): Boolean;`

Description Does `MyType`(28.4) is TOK_KEYWORD and Info.KeyWord is AKeyWord ?

IsStandardDirective

Declaration `public function IsStandardDirective( const AStandardDirective: TStandardDirective): Boolean;`

Description Does `MyType`(28.4) is TOK_IDENTIFIER and Info.StandardDirective is AStandardDirective ?

Description

Declaration `public function Description: string;`

Description Few words long description of this token. Describes MyType and Data (for those tokens that tend to have short Data). Starts with lower letter.

TTokenizer Class

Hierarchy

TTokenizer > TObject

Description

Converts an input TStream to a sequence of `TToken`(28.4) objects.

Properties

OnMessage `public property OnMessage: TPasDocMessageEvent read FOnMessage write FOnMessage;`

Verbosity `public property Verbosity: Cardinal read FVerbosity write FVerbosity;`

StreamName `public property StreamName: string read FStreamName;`

Informative to user name of the stream from which this token was read. This can be a filename (relative or absolute, however user specified it), but it also can be something arbitrary like "$if / $elseif condition".

So don't treat it like a reliable filename, for this use StreamAbsoluteFileName.

StreamAbsoluteFileName `public property StreamAbsoluteFileName: string read FStreamAbsoluteFileName;`

Filename, always absolute, of the underlying file of this stream. Empty (") if this is not a file stream.

Fields

FOnMessage `protected FOnMessage: TPasDocMessageEvent;`

FVerbosity `protected FVerbosity: Cardinal;`

BufferedChar `protected BufferedChar: Char;`

if `IsCharBuffered`(28.4) is true, this field contains the buffered character

EOS `protected EOS: Boolean;`

true if end of stream `Stream`(28.4) has been reached, false otherwise

IsCharBuffered `protected IsCharBuffered: Boolean;`

if this is true, `BufferedChar`(28.4) contains a buffered character; the next call to `GetChar`(28.4) or `PeekChar`(28.4) will return this character, not the next in the associated stream `Stream`(28.4)

Line `protected Line: Integer;`

current line number in stream `Stream`(28.4); useful when giving error messages

Stream `protected Stream: TStream;`

the input stream this tokenizer is working on

FStreamName `protected FStreamName: string;`

FStreamAbsoluteFileName `protected FStreamAbsoluteFileName: string;`

Methods

DoError

Declaration
```
protected procedure DoError(const AMessage: string; const AArguments:
array of const);
```

DoMessage

Declaration
```
protected procedure DoMessage(const AVerbosity: Cardinal; const
MessageType: TPasDocMessageType; const AMessage: string; const AArguments:
array of const);
```

CheckForDirective

Declaration
```
protected procedure CheckForDirective(const t: TToken);
```

ConsumeChar

Declaration
```
protected procedure ConsumeChar;
```

CreateSymbolToken

Declaration
```
protected function CreateSymbolToken(const st: TSymbolType; const s:
string): TToken; overload;
```

CreateSymbolToken

Declaration
```
protected function CreateSymbolToken(const st: TSymbolType): TToken;
overload;
```

Description Uses default symbol representation, from SymbolNames[st]

GetChar

Declaration
```
protected function GetChar(out c: AnsiChar): Integer;
```

Description Returns 1 on success or 0 on failure

PeekChar

Declaration
```
protected function PeekChar(out c: Char): Boolean;
```

ReadCommentType1

Declaration `protected function ReadCommentType1: TToken;`

ReadCommentType2

Declaration `protected function ReadCommentType2: TToken;`

ReadCommentType3

Declaration `protected function ReadCommentType3: TToken;`

ReadAttAssemblerRegister

Declaration `protected function ReadAttAssemblerRegister: TToken;`

ReadLiteralString

Declaration `protected function ReadLiteralString(var t: TToken): Boolean;`

ReadToken

Declaration `protected function ReadToken(c: Char; const s: TCharSet; const TT: TTokenType; var t: TToken): Boolean;`

Create

Declaration `public constructor Create( const AStream: TStream; const OnMessageEvent: TPasDocMessageEvent; const VerbosityLevel: Cardinal; const AStreamName, AStreamAbsoluteFileName: string);`

Description Creates a TTokenizer and associates it with given input TStream. Note that AStream will be freed when this object will be freed.

Destroy

Declaration `public destructor Destroy; override;`

Description Releases all dynamically allocated memory.

HasData

Declaration `public function HasData: Boolean;`

GetStreamInfo

Declaration `public function GetStreamInfo: string;`

GetToken

Declaration `public function GetToken(const NilOnEnd: Boolean = false; const NilOnInvalidContent: Boolean = false): TToken;`

Description Get next token from stream.

Parameters **NilOnEnd** If `True`, return `Nil` once stream ends. Otherwise we make exception about it.

NilOnInvalidContent If `True`, return `Nil` on some invalid content, like "!" which is not Pascal token at all. This parameter is independent from NilOnEnd. The "invalid content" affected by this parameter is still something that "we can read to advance our position within the stream".

UnGetToken

Declaration `public procedure UnGetToken(var T: TToken);`

Description Makes the token T next to be returned by GetToken. Also sets T to `Nil`, to prevent you from freeing it accidentally.

You cannot have more than one "unget" token. If you only call UnGetToken after some GetToken, you are safe.

SkipUntilCompilerDirective

Declaration `public function SkipUntilCompilerDirective: TToken;`

Description Skip all chars until it encounters some compiler directive, like $ELSE or $ENDIF. Returns either `Nil` or a token with MyType = TOK_DIRECTIVE.

28.5 Functions and Procedures

StandardDirectiveByName

Declaration `function StandardDirectiveByName(const Name: string): TStandardDirective;`

Description Checks is Name (case ignored) some Pascal keyword. Returns SD_INVALIDSTANDARDDIRECTIVE if not.

KeyWordByName

Declaration `function KeyWordByName(const Name: string): TKeyword;`

Description Checks is Name (case ignored) some Pascal standard directive. Returns KEY_INVALIDKEYWORD if not.

28.6 Types

TTokenType

Declaration `TTokenType = (...);`

Description enumeration type that provides all types of tokens; each token's name starts with TOK_.

TOK_DIRECTIVE is a compiler directive (like $ifdef, $define).

Note that tokenizer is not able to tell whether you used standard directive (e.g. 'Register') as an identifier (e.g. you're declaring procedure named 'Register') or as a real standard directive (e.g. a calling specifier 'register'). So there is *no* value like TOK_STANDARD_DIRECTIVE here, standard directives are always reported as TOK_IDENTIFIER. You can check TToken.Info.StandardDirective to know whether this identifier is *maybe* used as real standard directive.

Values
`TOK_WHITESPACE`
`TOK_COMMENT_PAS`
`TOK_COMMENT_EXT`
`TOK_COMMENT_HELPINSIGHT`
`TOK_COMMENT_CSTYLE`
`TOK_IDENTIFIER`
`TOK_NUMBER`
`TOK_STRING`
`TOK_SYMBOL`
`TOK_DIRECTIVE`
`TOK_KEYWORD`
`TOK_ATT_ASSEMBLER_REGISTER`

TKeyword

Declaration `TKeyword = (...);`

Description

Values
`KEY_INVALIDKEYWORD`
`KEY_AND`
`KEY_ARRAY`
`KEY_AS`
`KEY_ASM`
`KEY_BEGIN`
`KEY_CASE`
`KEY_CLASS`

KEY_OBJCCLASS
KEY_CONST
KEY_CONSTRUCTOR
KEY_DESTRUCTOR
KEY_DISPINTERFACE
KEY_DIV
KEY_DO
KEY_DOWNTO
KEY_ELSE
KEY_END
KEY_EXCEPT
KEY_EXPORTS
KEY_FILE
KEY_FINALIZATION
KEY_FINALLY
KEY_FOR
KEY_FUNCTION
KEY_GOTO
KEY_IF
KEY_IMPLEMENTATION
KEY_IN
KEY_INHERITED
KEY_INITIALIZATION
KEY_INLINE
KEY_INTERFACE
KEY_IS
KEY_LABEL
KEY_LIBRARY
KEY_MOD
KEY_NIL
KEY_NOT
KEY_OBJECT
KEY_OF
KEY_ON
KEY_OR

KEY_PACKED

KEY_PROCEDURE

KEY_PROGRAM

KEY_PROPERTY

KEY_RAISE

KEY_RECORD

KEY_REPEAT

KEY_RESOURCESTRING

KEY_SET

KEY_SHL

KEY_SHR

KEY_STRING

KEY_THEN

KEY_THREADVAR

KEY_TO

KEY_TRY

KEY_TYPE

KEY_UNIT

KEY_UNTIL

KEY_USES

KEY_VAR

KEY_WHILE

KEY_WITH

KEY_XOR

KEY_OUT

TStandardDirective

Declaration `TStandardDirective = (...);`

Description

Values SD_INVALIDSTANDARDDIRECTIVE

SD_ABSOLUTE

SD_ABSTRACT

SD_APIENTRY

SD_ASSEMBLER

SD_AUTOMATED

SD_CDECL

SD_CVAR

SD_DEFAULT

SD_DISPID

SD_DYNAMIC

SD_EXPERIMENTAL

SD_EXPORT

SD_EXTERNAL

SD_FAR

SD_FORWARD

SD_GENERIC

SD_HELPER

SD_INDEX

SD_INLINE

SD_MESSAGE

SD_NAME

SD_NEAR

SD_NODEFAULT

SD_OPERATOR

SD_OUT

SD_OVERLOAD

SD_OVERRIDE

SD_PASCAL

SD_PRIVATE

SD_PROTECTED

SD_PUBLIC

SD_PUBLISHED

SD_READ

SD_REFERENCE

SD_REGISTER

SD_REINTRODUCE

SD_RESIDENT

SD_SEALED

SD_SPECIALIZE

SD_STATIC

SD_STDCALL

SD_STORED

SD_STRICT

SD_VIRTUAL

SD_WRITE

SD_DEPRECATED

SD_SAFECALL

SD_PLATFORM

SD_VARARGS

SD_FINAL

TStandardDirectives

Declaration `TStandardDirectives = set of TStandardDirective;`

TSymbolType

Declaration `TSymbolType = (...);`

Description enumeration type that provides all types of symbols; each symbol's name starts with SYM_

Values SYM_PLUS

SYM_MINUS

SYM_ASTERISK

SYM_SLASH

SYM_EQUAL

SYM_LESS_THAN

SYM_LESS_THAN_EQUAL

SYM_GREATER_THAN

SYM_GREATER_THAN_EQUAL

SYM_LEFT_BRACKET

SYM_RIGHT_BRACKET

SYM_COMMA

SYM_LEFT_PARENTHESIS

SYM_RIGHT_PARENTHESIS

SYM_COLON

SYM_SEMICOLON

SYM_DEREFERENCE

SYM_PERIOD

SYM_AT

SYM_DOLLAR

SYM_ASSIGN

SYM_RANGE

SYM_POWER

SYM_BACKSLASH SYM_BACKSLASH may occur when writing char constant "^\", see ../../tests/ok_caret_chara

28.7 Constants

TOKEN_TYPE_NAMES

Declaration `TOKEN_TYPE_NAMES: array[TTokenType] of string = ( 'whitespace', 'comment ((**)-style)', 'comment ({}-style)', 'comment (///-style)', 'comment (//-style)', 'identifier', 'number', 'string', 'symbol', 'directive', 'reserved word', 'AT&T assembler register name');`

Description Names of the token types. All start with lower letter. They should somehow describe (in a few short words) given TTokenType.

TokenCommentTypes

Declaration `TokenCommentTypes: set of TTokenType = [ TOK_COMMENT_PAS, TOK_COMMENT_EXT, TOK_COMMENT_HELPINSIGHT, TOK_COMMENT_CSTYLE ];`

SymbolNames

Declaration `SymbolNames: array[TSymbolType] of string = ( '+', '-', '*', '/', '=', '<', '<=', '>', '>=', '[', ']', ',', '(', ')', ':', ';', '^', '.', '@', '$', ':=', '..', '**', '\' );`

Description Symbols as strings. They can be useful to have some mapping TSymbolType -> string, but remember that actually some symbols in tokenizer have multiple possible representations, e.g. "right bracket" is usually given as "]" but can also be written as ".)".

KeyWordArray

Declaration `KeyWordArray: array[Low(TKeyword)..High(TKeyword)] of string = ('x', 'AND', 'ARRAY', 'AS', 'ASM', 'BEGIN', 'CASE', 'CLASS', 'OBJCCLASS', 'CONST', 'CONSTRUCTOR', 'DESTRUCTOR', 'DISPINTERFACE', 'DIV', 'DO', 'DOWNTO', 'ELSE', 'END', 'EXCEPT', 'EXPORTS', 'FILE', 'FINALIZATION', 'FINALLY', 'FOR', 'FUNCTION', 'GOTO', 'IF', 'IMPLEMENTATION', 'IN', 'INHERITED', 'INITIALIZATION', 'INLINE', 'INTERFACE', 'IS', 'LABEL', 'LIBRARY', 'MOD', 'NIL', 'NOT', 'OBJECT', 'OF', 'ON', 'OR', 'PACKED', 'PROCEDURE', 'PROGRAM', 'PROPERTY', 'RAISE', 'RECORD', 'REPEAT', 'RESOURCESTRING', 'SET', 'SHL', 'SHR', 'STRING', 'THEN', 'THREADVAR', 'TO', 'TRY', 'TYPE', 'UNIT', 'UNTIL', 'USES', 'VAR', 'WHILE', 'WITH', 'XOR', 'OUT');`

Description all Object Pascal keywords

StandardDirectiveArray

Declaration
```
StandardDirectiveArray:
array[Low(TStandardDirective)..High(TStandardDirective)] of PChar = ('x',
'ABSOLUTE', 'ABSTRACT', 'APIENTRY', 'ASSEMBLER', 'AUTOMATED', 'CDECL',
'CVAR', 'DEFAULT', 'DISPID', 'DYNAMIC', 'EXPERIMENTAL', 'EXPORT', 'EXTERNAL',
'FAR', 'FORWARD', 'GENERIC', 'HELPER', 'INDEX', 'INLINE', 'MESSAGE', 'NAME',
'NEAR', 'NODEFAULT', 'OPERATOR', 'OUT', 'OVERLOAD', 'OVERRIDE', 'PASCAL',
'PRIVATE', 'PROTECTED', 'PUBLIC', 'PUBLISHED', 'READ', 'REFERENCE',
'REGISTER', 'REINTRODUCE', 'RESIDENT', 'SEALED', 'SPECIALIZE', 'STATIC',
'STDCALL', 'STORED', 'STRICT', 'VIRTUAL', 'WRITE', 'DEPRECATED', 'SAFECALL',
'PLATFORM', 'VARARGS', 'FINAL');
```

Description Object Pascal directives

28.8 Authors

Johannes Berg <johannes@sipsolutions.de>
Ralf Junker (delphi@zeitungsjunge.de)
Marco Schmidt (marcoschmidt@geocities.com)
Michalis Kamburelis
Arno Garrels <first name.name@nospamgmx.de>

Chapter 29

Unit PasDoc_Types

29.1 Description

Basic types.

29.2 Uses

- `SysUtils`
- `StrUtils`
- `Types`

29.3 Overview

`EPasDoc Class` Exception raised in many situations when PasDoc encounters an error.

`SplitNameParts` Splits S, which can be made of any number of parts, separated by dots (Delphi namespaces, like PasDoc.Output.HTML.TWriter.Write).

`OneNamePart` Simply returns an array with Length = 1 and one item = S.

`GlueNameParts` Simply concatenates all NameParts with dot.

29.4 Classes, Interfaces, Objects and Records

EPasDoc Class

Hierarchy

EPasDoc > Exception

Description

Exception raised in many situations when PasDoc encounters an error.

Methods

Create

Declaration `public constructor Create(const AMessageFormat: string; const AArguments: array of const; const AExitCode: Word = 3); overload;`

Create

Declaration `public constructor Create(const AMessage: string; const AExitCode: Word = 3); overload;`

29.5 Functions and Procedures

SplitNameParts

Declaration `function SplitNameParts(S: string; out NameParts: TNameParts): Boolean;`

Description Splits S, which can be made of any number of parts, separated by dots (Delphi namespaces, like PasDoc.Output.HTML.TWriter.Write). If S is not a valid identifier, `False` is returned, otherwise `True` is returned and splitted name is returned as NameParts.

OneNamePart

Declaration `function OneNamePart(const S: string): TNameParts;`

Description Simply returns an array with Length = 1 and one item = S.

GlueNameParts

Declaration `function GlueNameParts(const NameParts: TNameParts): string;`

Description Simply concatenates all NameParts with dot.

29.6 Types

TBytes

Declaration `TBytes = array of Byte;`

UnicodeString

Declaration `UnicodeString = WideString;`

RawByteString

Declaration `RawByteString = AnsiString;`

TStringArray

Declaration `TStringArray = TStringDynArray;`

TNameParts

Declaration `TNameParts = TStringArray;`

Description This represents parts of a qualified name of some item.

User supplies such name by separating each part with dot, e.g. 'UnitName.ClassName.ProcedureName', then `SplitNameParts`(29.5) converts it to TNameParts like ['UnitName', 'ClassName', 'ProcedureName']. Length must be *always* between 1 and `MaxNameParts`(29.7).

TPasDocMessageType

Declaration `TPasDocMessageType = (...);`

Description

Values `pmtPlainText`

`pmtInformation`

`pmtWarning`

`pmtError`

TPasDocMessageEvent

Declaration
```
TPasDocMessageEvent = procedure(const MessageType:  TPasDocMessageType;
const AMessage:  string; const AVerbosity:  Cardinal) of object;
```

TCharSet

Declaration `TCharSet = set of AnsiChar;`

TImplicitVisibility

Declaration `TImplicitVisibility = (...);`

Description See command-line option --implicit-visibility documentation at –implicit-visibility documentation.

Values `ivPublic`

`ivPublished`

`ivImplicit`

29.7 Constants

MaxNameParts

Declaration `MaxNameParts = 3;`

CP_UTF16

Declaration `CP_UTF16 = 1200;`

Description Windows Unicode code page ID

CP_UTF16Be

Declaration `CP_UTF16Be = 1201;`

CP_UTF32

Declaration `CP_UTF32 = 12000;`

CP_UTF32Be

Declaration `CP_UTF32Be = 12001;`

29.8 Authors

Johannes Berg <johannes@sipsolutions.de>
Michalis Kamburelis
Arno Garrels <first name.name@nospamgmx.de>

Chapter 30

Unit PasDoc_Utils

30.1 Description

Utility functions.

30.2 Uses

- `Classes`
- `SysUtils`
- `PasDoc_Types`(29)

30.3 Overview

`TCharReplacement Record`

`IsStrEmptyA` string empty means it contains only whitespace

`StrCountCharA` count occurences of AChar in AString

`StrPosIA` Position of the ASub in AString.

`MakeMethod` creates a "method pointer"

`StringReplaceChars` Returns S with each char from ReplacementArray[].cChar replaced with ReplacementArray[].sSpec.

`SCharIs` Comfortable shortcut for Index <= Length(S) and S[Index] = C.

`SCharIs` Comfortable shortcut for Index <= Length(S) and S[Index] in Chars.

`ExtractFirstWord` Extracts all characters up to the first white-space encountered (ignoring white-space at the very beginning of the string) from the string specified by S.

`ExtractFirstWord` Another version of ExtractFirstWord.

`SkipBOM` Interpret and skip the BOM in the InputStream.

`FileToString` Read the given FileName contents into a String.

`StringToFile` Save the String content into a file.

`DataToFile` Save the binary Data into a file.

`SCharsReplace` Returns S with all Chars replaced by ReplacementChar

`SCharsReplace`

`CopyFile`

`IsPrefix` Checks is Prefix a prefix of S.

`RemovePrefix` If IsPrefix(Prefix, S), then remove the prefix, otherwise return unmodifed S.

`SEnding` SEnding returns S contents starting from position P.

`IsPathAbsolute` Check is the given Path absolute.

`IsPathAbsoluteOnDrive` Just like IsPathAbsolute, but on Windows accepts also paths that specify full directory tree without drive letter.

`CombinePaths` Combines BasePath with RelPath.

`DeleteFileExt` Remove from the FileName the last extension (including the dot).

`RemoveIndentation` Remove common indentation (whitespace prefix) from a multiline string.

`Swap16Buf`

`IsCharInSet`

`IsCharInSet`

`IsUtf8LeadByte`

`IsUtf8TrailByte`

`Utf8Size`

`IsLeadChar`

`StripHtml` Strip HTML elements from the string.

`SAppendPart` If S = ” then returns NextPart, else returns S + PartSeparator + NextPart.

`CharsPos` Find first occurrence of any character in Chars in string S.

`SRemoveChars` Remove all instances of a character in Chars from a string.

30.4 Classes, Interfaces, Objects and Records

TCharReplacement Record

Fields

cChar `public cChar: Char;`

sSpec `public sSpec: string;`

30.5 Functions and Procedures

IsStrEmptyA

Declaration `function IsStrEmptyA(const AString: string): boolean;`

Description string empty means it contains only whitespace

StrCountCharA

Declaration `function StrCountCharA(const AString: string; const AChar: Char): Integer;`

Description count occurences of AChar in AString

StrPosIA

Declaration `function StrPosIA(const ASub, AString: string): Integer;`

Description Position of the ASub in AString. Return 0 if not found

MakeMethod

Declaration `function MakeMethod(const AObject: Pointer; AMethod: Pointer): TMethod;`

Description creates a "method pointer"

StringReplaceChars

Declaration `function StringReplaceChars(const S: string; const ReplacementArray: array of TCharReplacement): string;`

Description Returns S with each char from ReplacementArray[].cChar replaced with ReplacementArray[].sSpec.

SCharIs

Declaration `function SCharIs(const S: string; Index: integer; C: char): boolean; overload;`

Description Comfortable shortcut for Index <= Length(S) and S[Index] = C.

SCharIs

Declaration `function SCharIs(const S: string; Index: integer; const Chars: TCharSet): boolean; overload;`

Description Comfortable shortcut for Index <= Length(S) and S[Index] in Chars.

ExtractFirstWord

Declaration `function ExtractFirstWord(var s: string): string; overload;`

Description Extracts all characters up to the first white-space encountered (ignoring white-space at the very beginning of the string) from the string specified by S.

If there is no white-space in S (or there is white-space only at the beginning of S, in which case it is ignored) then the whole S is regarded as it's first word.

Both S and result are trimmed, i.e. they don't have any excessive white-space at the beginning or end.

ExtractFirstWord

Declaration `procedure ExtractFirstWord(const S: string; out FirstWord, Rest: string); overload;`

Description Another version of ExtractFirstWord.

Splits S by it's first white-space (ignoring white-space at the very beginning of the string). No such white-space means that whole S is regarded as the FirstWord.

Both FirstWord and Rest are trimmed.

SkipBOM

Declaration `procedure SkipBOM(const InputStream: TStream);`

Description Interpret and skip the BOM in the InputStream. Assumes that initial position is at the beginning of the InputStream, and will change the position to the one immediately after BOM (or 0, if no BOM was detected).

Exceptions `EPasDoc`**(29.4)** When BOM indicates UTF encoding that we cannot handle (UTF-32, UTF-16 now).

FileToString

Declaration `function FileToString(const FileName: string): string;`

Description Read the given FileName contents into a String. Use this only with text files – it does automatic UTF BOM skipping, so it assumes it is a text file, not a binary file with random contents.

StringToFile

Declaration `procedure StringToFile(const FileName, S: string);`

Description Save the String content into a file. Overwrites the FileName, if it already exists.

DataToFile

Declaration `procedure DataToFile(const FileName: string; const Data: array of Byte);`

Description Save the binary Data into a file. Overwrites the FileName, if it already exists.

SCharsReplace

Declaration `function SCharsReplace(const S: String; const Chars: TCharSet; const ReplacementChar: Char): string; overload;`

Description Returns S with all Chars replaced by ReplacementChar

SCharsReplace

Declaration `function SCharsReplace(const S: String; const SearchChar: Char; const ReplacementChar: Char): string; overload;`

CopyFile

Declaration `procedure CopyFile(const SourceFileName, DestinationFileName: string);`

IsPrefix

Declaration `function IsPrefix(const Prefix, S: string): boolean;`

Description Checks is Prefix a prefix of S. Not case-sensitive.

RemovePrefix

Declaration `function RemovePrefix(const Prefix, S: string): string;`

Description If IsPrefix(Prefix, S), then remove the prefix, otherwise return unmodifed S.

SEnding

Declaration `function SEnding(const s: string; P: integer): string;`

Description SEnding returns S contents starting from position P. Returns ” if P > length(S). Yes, this is simply equivalent to Copy(S, P, MaxInt).

IsPathAbsolute

Declaration `function IsPathAbsolute(const Path: string): boolean;`

Description Check is the given Path absolute.

Path may point to directory or normal file, it doesn't matter. Also it doesn't matter whether Path ends with PathDelim or not.

Note for Windows: while it's obvious that `'c:\autoexec.bat'` is an absolute path, and `'autoexec.bat'` is not, there's a question whether path like `'\autoexec.bat'` is absolute? It doesn't specify drive letter, but it does specify full directory hierarchy on some drive. This function treats this as *not absolute*, on the reasoning that "not all information is contained in Path".

See also `IsPathAbsoluteOnDrive`**(30.5)** Just like IsPathAbsolute, but on Windows accepts also paths that specify full directory tree without drive letter.

IsPathAbsoluteOnDrive

Declaration `function IsPathAbsoluteOnDrive(const Path: string): boolean;`

Description Just like IsPathAbsolute, but on Windows accepts also paths that specify full directory tree without drive letter.

See also `IsPathAbsolute`**(30.5)** Check is the given Path absolute.

CombinePaths

Declaration `function CombinePaths(BasePath, RelPath: string): string;`

Description Combines BasePath with RelPath. BasePath MUST be an absolute path, on Windows it must contain at least drive specifier (like 'c:'), on Unix it must begin with "/". RelPath can be relative and can be absolute. If RelPath is absolute, result is RelPath. Else the result is an absolute path calculated by combining RelPath with BasePath.

DeleteFileExt

Declaration `function DeleteFileExt(const FileName: string): string;`

Description Remove from the FileName the last extension (including the dot). Note that if the FileName had a couple of extensions (e.g. `blah.x3d.gz`) this will remove only the last one. Will remove nothing if filename has no extension.

RemoveIndentation

Declaration `function RemoveIndentation(const Code: string): string;`

Description Remove common indentation (whitespace prefix) from a multiline string.

Swap16Buf

Declaration `procedure Swap16Buf(Src, Dst: PWord; WordCount: Integer);`

IsCharInSet

Declaration `function IsCharInSet(C: AnsiChar; const CharSet: TCharSet): Boolean; overload; inline;`

IsCharInSet

Declaration `function IsCharInSet(C: WideChar; const CharSet: TCharSet): Boolean; overload; inline;`

IsUtf8LeadByte

Declaration `function IsUtf8LeadByte(const B: Byte): Boolean; inline;`

IsUtf8TrailByte

Declaration `function IsUtf8TrailByte(const B: Byte): Boolean; inline;`

Utf8Size

Declaration `function Utf8Size(const LeadByte: Byte): Integer; inline;`

IsLeadChar

Declaration `function IsLeadChar(Ch: WideChar): Boolean; overload; inline;`

StripHtml

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

Description Strip HTML elements from the string.

Assumes that the HTML content is correct (all elements are nicely closed, all < > inside attributes are escaped to < >, all < > outside elements are escaped to < >). It doesn't try very hard to deal with incorrect HTML context (it will not crash, but results are undefined). It's designed to strip HTML from PasDoc-generated HTML, which should always be correct.

SAppendPart

Declaration `function SAppendPart(const s, PartSeparator, NextPart: String): String;`

Description If S = '' then returns NextPart, else returns S + PartSeparator + NextPart.

CharsPos

Declaration `function CharsPos(const Chars: TCharSet; const S: String): Integer;`

Description Find first occurrence of any character in Chars in string S. This is quite like FirstDelimiter but it takes parameter as TSetOfChars and has more sensible name. Returns 0 if not found.

SRemoveChars

Declaration `function SRemoveChars(const S: string; const Chars: TCharSet): string;`

Description Remove all instances of a character in Chars from a string.

30.6 Constants

AllChars

Declaration `AllChars = [Low(AnsiChar)..High(AnsiChar)];`

WhiteSpaceNotNL

Declaration `WhiteSpaceNotNL = [' ', #9];`

Description Whitespace that is not any part of newline.

WhiteSpaceNL

Declaration `WhiteSpaceNL = [#10, #13];`

Description Whitespace that is some part of newline.

WhiteSpace

Declaration `WhiteSpace = WhiteSpaceNotNL + WhiteSpaceNL;`

Description Any whitespace (that may indicate newline or not)

FlagStartSigns

Declaration `FlagStartSigns = ['['];`

Description Flag Start- and Endsigns for parameters (Feature request "direction of parameter": pasdoc issue 8)

FlagEndSigns

Declaration `FlagEndSigns = [']'];`

30.7 Authors

Johannes Berg <johannes@sipsolutions.de>
Michalis Kamburelis
Arno Garrels <first name.name@nospamgmx.de>

Chapter 31

Unit PasDoc_Versions

31.1 Description

Information about PasDoc and compilers version.

31.2 Overview

`COMPILER_NAME` Nice compiler name.

`PASDOC_FULL_INFO` Returns pasdoc name, version, used compiler version, etc.

31.3 Functions and Procedures

COMPILER_NAME

Declaration `function COMPILER_NAME: string;`

Description Nice compiler name. This is a function only because we can't nicely declare it as a constant. But this behaves like a constant, i.e. every time you call it it returns the same thing (as long as this is the same binary).

PASDOC_FULL_INFO

Declaration `function PASDOC_FULL_INFO: string;`

Description Returns pasdoc name, version, used compiler version, etc.

This is a function only because we can't nicely declare it as a constant. But this behaves like a constant, i.e. every time you call it it returns the same thing (as long as this is the same binary).

31.4 Constants

COMPILER_BITS

Declaration `COMPILER_BITS = '32' ;`

PASDOC_NAME

Declaration `PASDOC_NAME = 'PasDoc';`

PASDOC_DATE

Declaration `PASDOC_DATE = 'snapshot';`

Description PASDOC_DATE = '2021-02-07';

PASDOC_VERSION

Declaration `PASDOC_VERSION = '0.17.0.snapshot';`

PASDOC_NAME_AND_VERSION

Declaration `PASDOC_NAME_AND_VERSION = PASDOC_NAME + ' ' + PASDOC_VERSION;`

PASDOC_HOMEPAGE

Declaration `PASDOC_HOMEPAGE = 'https://pasdoc.github.io/';`