Pascal Browser

Pascal Browser

by Peganza

Pascal Browser will parse the source code it finds and gather all sorts of information. This information is used to create the document collection that describes the source code.

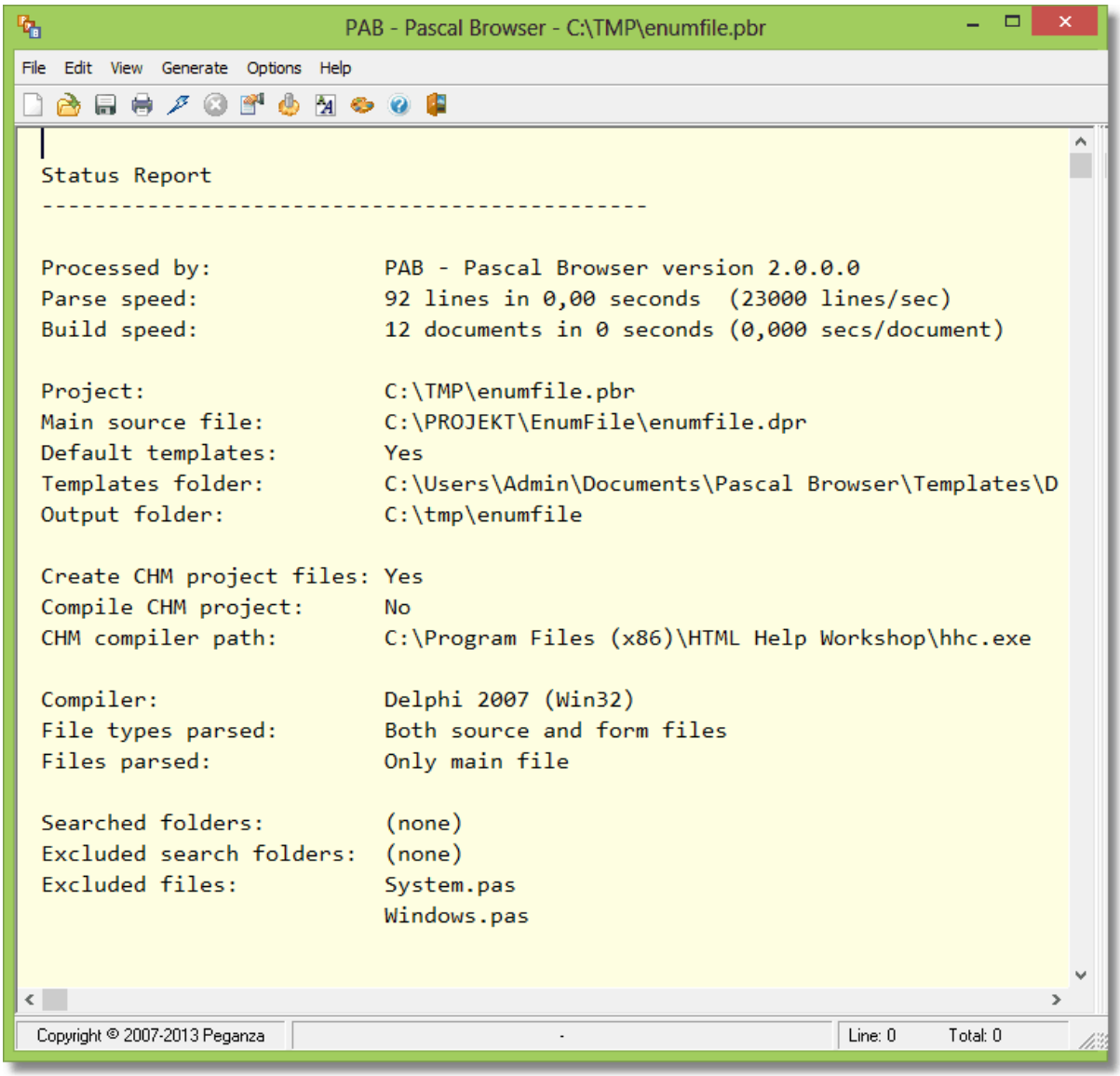
Pascal Browser

Table of Contents

1 Introduction

In short, Pascal Browser is a Windows application that can:

- create a hyperlinked collection of HTML documents for your source code
- create a full-text searchable CHM file for the hyperlinked collection of HTML documents
- help you understand and get an overview of your code
- assist you in finding errors and anomalies in the source code
- let new team members quickly get acquainted with the source code
- create your own customized documentation

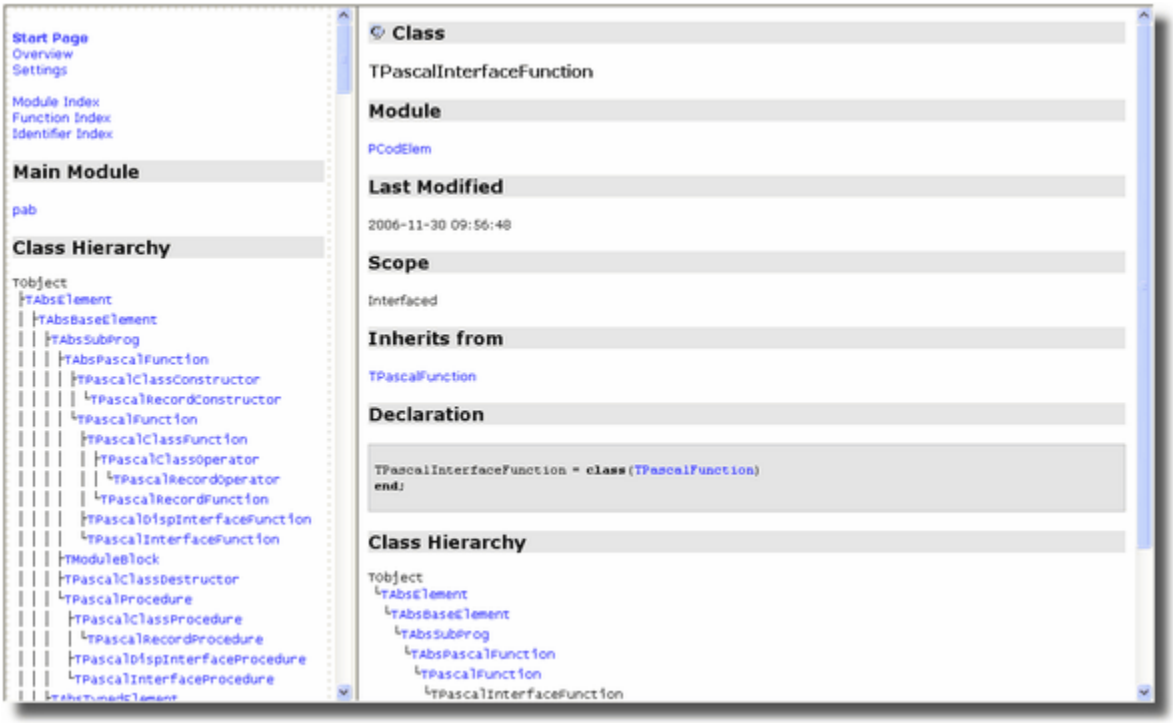


The main window in Pascal Browser

Just as its sibling product *Pascal Analyzer*, Pascal Browser will parse the source code it finds and gather all sorts of information. This information is used to create the document

collection that describes the source code. The most common use of Pascal Browser will probably be to create hyperlinked HTML documents.

This is a sample of a web page created by Pascal Browser:



Sample web page (HTML) created by Pascal Browser

Pascal Browser has deep knowledge of the source code, and understanding of how different identifiers are linked to each other. This knowledge can be used to generate a full documentation, but also to:

- display hyperlinked source code
- display class hierarchies
- display call chains
- point out unused identifiers and other anomalies

Pascal Browser functions with Pascal/Delphi Compilers from BP7 and later:

- Borland Pascal 7 (or earlier)

- Delphi 1
- Delphi 2
- Delphi 3
- Delphi 4
- Delphi 5
- Delphi 6
- Delphi 7

- Delphi 8 for .NET
- Delphi 2005 for Win32

- Delphi 2005 for .NET
- Delphi 2006 for Win32
- Delphi 2006 for .NET
- Delphi 2007 for Win32
- Delphi 2007 for .NET
- Delphi 2009 for Win32
- Delphi 2010 for Win32

- Delphi XE for Win32

- Delphi XE2 for Win32
- Delphi XE2 for Win64
- Delphi XE2 for OSX

- Delphi XE3 for Win32
- Delphi XE3 for Win64
- Delphi XE3 for OSX

- Delphi XE4 for Win32
- Delphi XE4 for Win64
- Delphi XE4 for OSX
- Delphi XE4 for iOS Device
- Delphi XE4 for iOS Simulator

- Delphi XE5 for Win32
- Delphi XE5 for Win64
- Delphi XE5 for OSX
- Delphi XE5 for iOS Device
- Delphi XE5 for iOS Simulator
- Delphi XE5 for Android

- Delphi XE6 for Win32
- Delphi XE6 for Win64
- Delphi XE6 for OSX
- Delphi XE6 for iOS Device
- Delphi XE6 for iOS Simulator
- Delphi XE6 for Android

- Delphi XE7 for Win32
- Delphi XE7 for Win64
- Delphi XE7 for OSX
- Delphi XE7 for iOS Device
- Delphi XE7 for iOS Simulator
- Delphi XE7 for Android

- Delphi XE8 for Win32
- Delphi XE8 for Win64
- Delphi XE8 for OSX
- Delphi XE8 for iOS Device 32-bits
- Delphi XE8 for iOS Device 64-bits
- Delphi XE8 for iOS Simulator
- Delphi XE8 for Android

- Delphi 10 for Win32
- Delphi 10 for Win64
- Delphi 10 for OSX
- Delphi 10 for iOS Device 32-bits
- Delphi 10 for iOS Device 64-bits
- Delphi 10 for iOS Simulator
- Delphi 10 for Android

- Delphi 10.1 for Win32
- Delphi 10.1 for Win64
- Delphi 10.1 for OSX
- Delphi 10.1 for iOS Device 32-bits
- Delphi 10.1 for iOS Device 64-bits
- Delphi 10.1 for iOS Simulator
- Delphi 10.1 for Android

- Delphi 10.2 for Win32
- Delphi 10.2 for Win64
- Delphi 10.2 for OSX
- Delphi 10.2 for iOS Device 32-bits
- Delphi 10.2 for iOS Device 64-bits
- Delphi 10.2 for iOS Simulator
- Delphi 10.2 for Android
- Delphi 10.2 for Linux 64-bits

- Delphi 10.3 for Win32
- Delphi 10.3 for Win64
- Delphi 10.3 for OSX 32-bits
- Delphi 10.3 for OSX 64-bits
- Delphi 10.3 for iOS Device 32-bits
- Delphi 10.3 for iOS Device 64-bits
- Delphi 10.3 for iOS Simulator
- Delphi 10.3 for Android 32-bits
- Delphi 10.3 for Android 64-bits
- Delphi 10.3 for Linux 64-bits

- Delphi 10.4 for Win32
- Delphi 10.4 for Win64
- Delphi 10.4 for OSX 32-bits
- Delphi 10.4 for OSX 64-bits
- Delphi 10.4 for iOS Device 32-bits
- Delphi 10.4 for iOS Device 64-bits
- Delphi 10.4 for iOS Simulator
- Delphi 10.4 for Android 32-bits
- Delphi 10.4 for Android 64-bits
- Delphi 10.4 for Linux 64-bits

- Delphi 11 for Win32
- Delphi 11 for Win64
- Delphi 11 for OSX 32-bits
- Delphi 11 for OSX 64-bits

- Delphi 11 for iOS Device 32-bits
- Delphi 11 for iOS Device 64-bits
- Delphi 11 for iOS Simulator
- Delphi 11 for Android 32-bits
- Delphi 11 for Android 64-bits
- Delphi 11 for Linux 64-bits
- Delphi 11 for OSX ARM 64-bits

- Delphi 12 for Win32
- Delphi 12 for Win64
- Delphi 12 for OSX 32-bits
- Delphi 12 for OSX 64-bits
- Delphi 12 for iOS Device 32-bits
- Delphi 12 for iOS Device 64-bits
- Delphi 12 for iOS Simulator
- Delphi 12 for Android 32-bits
- Delphi 12 for Android 64-bits
- Delphi 12 for Linux 64-bits
- Delphi 12 for OSX ARM 64-bits

Pascal Browser parses your source code in the same way as the compiler. It builds large data tables in memory and when the parsing is finished, outputs an XML file. The documentation is created by merging this XML file with XSLT style sheets.

Be forewarned that Pascal Browser sometimes needs a lot of memory. The amount of memory needed is proportional to the number of code lines and modules in the examined project.

Projects

To generate documentation for a particular set of source code with Pascal Browser, you must first create a project. Do not confuse a Pascal Browser project with a Delphi project, they are completely different things. The project holds the options for how the code is handled, and lets you conveniently use separate options for different sets of source code.

Projects are saved as text files with the extension “pbr”, like for example a file with the name MyProj.pbr. The format of the files is equivalent to that of an INI file.

How to get started

It is very easy to get started with Pascal Browser. Just create a new project and select a main file, an output root folder and some other options. Then hit the Run-button and wait for the results.

A good piece of advice is to start with a small code base. Make sure that you do not include too much source code to start with. It will be easier to navigate in the results if it is not overwhelmingly large.

Evaluation version

The evaluation version is free to download and try out. There are a few limitations in the evaluation version:

- identifiers starting with letters **a, d, g, j, m, p, s, v** (and in upper case) are not hyperlinked.
- some identifiers are obfuscated in the generated documentation
- some lines in the middle of source code listings are partly obfuscated

Special thanks to

- **Borland**, for giving us Delphi, the most productive programming environment ever
- **Embarcadero/CodeGear**, for continuing Borland's work

- **glyFX** (http://www.glyfx.com); many of our graphical elements are from their excellent glyph collections. Highly recommended!

- **Inno Setup** (http://innosetup.org/isinfo.php), a great utility to create powerful installation programs, we use it for all our applications

See also:

How to use PAB.EXE and PAB32.EXE
Main window
What's new in PAB 3.x?
What's new in PAB 2.x?

Web site: https://www.peganza.com
Email: support@peganza.com

2 What's new in PAB 3.x (January 2018, updated November 2023)?

This text describes changes and new features in Pascal Browser version 3, compared to version 2.x.

- support for Delphi 12 Athens (all compiler targets) has been added, released in November 2023

- support for Delphi 11 Alexandria (all compiler targets) has been added, including up to 11.3 released in February 2023

- support for Delphi 10.4 Sydney (all compiler targets) has been added

- support for Delphi 10.3 Rio (all compiler targets) has been added

- support for Delphi 10.2 Tokyo (all compiler targets) has been added

- Pascal Browser is now available also in a 64-bits version, but 32-bits version is still included

- optimizations and speed-ups: Pascal Browser is now about twice as fast as the 2.x version

- new licensing scheme, subscription model (see the Orders page at our web site for details)

- relative paths for folders can be specified. This applies to the main file folder, output folder, searched folders, excluded report folders and excluded search folders. The relative path should be relative to the folder where the project file (PBR-file) is located.

See also:

Introduction
What's new in PAB 2.x? (February 2014, updated May 2016)

3 What's new in PAB 2.x (February 2014, updated May 2016)?

This text describes changes and new features in Pascal Browser version 2, compared to version 1.x.

- standalone CHM files can now be created from output HTML files. The CHM file format is standardized, and provides full-text search. It also compresses its content, and so needs much less disk space compared to the original HTML files.

- support for Delphi versions up to and including Delphi 10.1 Berlin is available

See also:

Introduction
What's new in PAB 3.x

4 How to use PAB.EXE and PAB32.EXE

Activation

When starting PAB for the first time, your license must be activated through the Internet, unless you activated it during the installation. If you are running PAB on a computer that has not got access to the Internet, you can create an activation XML file and send to us. You will then receive a response XML file that you use to manually activate.

For the activation, use the registration key that was sent to you by mail when buying the product license. This key is good for a small number of activations. Contact us if you run out of activations, for example when reinstalling on a new computer. You are entitled to install Pascal Browser on up to four computers, as long as you are the Pascal Browser user on those computers. If more than one developer needs Pascal Browser, additional licenses must be bought.

If you need to move the installation to another computer, you can deactivate the license on the current computer. Then you will be able to activate the license on the new computer. Use the menu command "Deactivate License" in the About-menu for this.

Summary

Pascal Browser is an easy-to-use standalone Windows program. Just create a new project and select a source file to generate documentation for. Either select a complete Delphi project (DPR-file), Delphi package (DPK-file) or a single source file (PAS-file), set a few options and start the process.

Unlike some other tools, you do not have to alter your code in order to process it with Pascal Browser. It does not either change or affect your code in any way.

It is very easy to use and to get started with. Follow this simple procedure to create documentation with Pascal Browser:

1. Create a new Pascal Browser project, by selection File|New from the main menu. Then select a main file to analyze, either a complete Delphi project or a single source file. You can also open an existing Pascal Browser project.

2. Make sure that the selected compiler target is suitable for the source code. Enter other options that are required, like which main file to analyze.

3. Press the Run button and wait from a few seconds up to several minutes, depending on the size of the code. Pascal Browser will parse the code, and apply templates in XSL-format to the gathered information. The output is created as a collection of files, in the selected output folder. Files in the template folder (and optionally sub folders) are copied to the output location (except *.XSL files).

4. Examine the documentation with an external program (normally a browser). By default, Pascal Browser will automatically open the main document after the process is finished.

Command-line parameters

In this version, there is one type of parameter you can use on the command-line:

You can use a path setting on the command-line, for the project you want Pascal Browser to open at start-up. Normally the last used project is loaded, so this is a way to override that behaviour.

Example:

PAB.EXE c:\project\MyProj.pbr

Pascal Browser will open MyProj.pbr at start-up.

Pascal Browser is not a Delphi IDE plug-in (expert or wizard). It is a standalone program. You can however install the application in the Delphi IDE, allowing easy access from within the Delphi development environment.

To install Pascal Browser in the Delphi IDE, follow these simple steps (those are written for Delphi 7, but should apply even to other versions):

1. Start Delphi and select Tools|Configure Tools
2. In the dialog box, press the Add button
3. Fill in the fields, for instance with these values:

Title:	Pascal Browser
Program:	C:\Program Files\Peganza\Pascal Browser\PAB.exe
Working dir:	C:\Program Files\Peganza\Pascal Browser

See also:

Introduction
Main window
What's new in PAB 3.x (January 2018, updated December 2018)
What's new in PAB 2.x? (February 2014, updated May 2016)

5 How to use PABCMD.EXE and PABCMD32.EXE

The standalone command-line version PABCMD.EXE (and 32-bits PABCMD32.EXE) is useful when you want to automate the process of creating documentation. For example, you may want to integrate Pascal Browser into your build process.

PABCMD.EXE uses exactly the same engine as the GUI version PAB.EXE and produces the same output.
You run PABCMD.EXE from the command prompt using the following syntax:

PABCMD projectpath|sourcepath [options]

Option	Explanation
/A+	Parse both source/form files
/A-	Parse source files only
/FA	Parse all files
/FR	Parse main file and directly used files
/FM	Parse main file only
/Q	Quiet mode
/CBP	Borland Pascal 7 (or earlier)
/CD1	Delphi 1
/CD2	Delphi 2
/CD3	Delphi 3
/CD4	Delphi 4
/CD5	Delphi 5
/CD6	Delphi 6
/CD7	Delphi 7
/CD8	Delphi 8 for .NET
/CD9W	Delphi 2005 for Win32
/CD9N	Delphi 2005 for .NET
/CD10W	Delphi 2006 for Win32 (also Turbo Delphi for Win32)
/CD10N	Delphi 2006 for .NET (also Turbo Delphi for .NET)
/CD11W	Delphi 2007 for Win32
/CD11N	Delphi 2007 for .NET
/CD12W	Delphi 2009 for Win32
/CD14W	Delphi 2010 for Win32
/CDXEW	Delphi XE for Win32
/CDXE2W32	Delphi XE2 for Win32
/CDXE2W64	Delphi XE2 for Win64
/CDXE2OSX	Delphi XE2 for OSX
/CDXE3W32	Delphi XE3 for Win32
/CDXE3W64	Delphi XE3 for Win64
/CDXE3OSX	Delphi XE3 for OSX
/CDXE4W32	Delphi XE4 for Win32
/CDXE4W64	Delphi XE4 for Win64
/CDXE4OSX	Delphi XE4 for OSX
/CDXE4IOSDEV	Delphi XE4 for iOS Device

/CDXE4IOSSIM	Delphi XE4 for iOS Simulator
/CDXE5W32	Delphi XE5 for Win32
/CDXE5W64	Delphi XE5 for Win64
/CDXE5OSX	Delphi XE5 for OSX
/CDXE5IOSDEV	Delphi XE5 for iOS Device
/CDXE5IOSSIM	Delphi XE5 for iOS Simulator
/CDXE5ANDROID	Delphi XE5 for Android
/CDXE6W32	Delphi XE6 for Win32
/CDXE6W64	Delphi XE6 for Win64
/CDXE6OSX	Delphi XE6 for OSX
/CDXE6IOSDEV	Delphi XE6 for iOS Device
/CDXE6IOSSIM	Delphi XE6 for iOS Simulator
/CDXE6ANDROID	Delphi XE6 for Android
/CDXE7W32	Delphi XE7 for Win32
/CDXE7W64	Delphi XE7 for Win64
/CDXE7OSX	Delphi XE7 for OSX
/CDXE7IOSDEV	Delphi XE7 for iOS Device
/CDXE7IOSSIM	Delphi XE7 for iOS Simulator
/CDXE7ANDROID	Delphi XE7 for Android
/CDXE8W32	Delphi XE8 for Win32
/CDXE8W64	Delphi XE8 for Win64
/CDXE8OSX	Delphi XE8 for OSX
/CDXE8IOSDEV	Delphi XE8 for iOS Device 32-bits
/CDXE8IOSDEV64	Delphi XE8 for iOS Device 64-bits
/CDXE8IOSSIM	Delphi XE8 for iOS Simulator
/CDXE8ANDROID	Delphi XE8 for Android
/CD10W32	Delphi 10 for Win32
/CD10W64	Delphi 10 for Win64
/CD10OSX	Delphi 10 for OSX
/CD10IOSDEV	Delphi 10 for iOS Device 32-bits
/CD10IOSDEV64	Delphi 10 for iOS Device 64-bits
/CD10IOSSIM	Delphi 10 for iOS Simulator
/CD10ANDROID	Delphi 10 for Android
/CD101W32	Delphi 10.1 for Win32
/CD101W64	Delphi 10.1 for Win64
/CD101OSX	Delphi 10.1 for OSX
/CD101IOSDEV	Delphi 10.1 for iOS Device 32-bits
/CD101IOSDEV64	Delphi 10.1 for iOS Device 64-bits
/CD101IOSSIM	Delphi 10.1 for iOS Simulator
/CD101ANDROID	Delphi 10.1 for Android
/CD102W32	Delphi 10.2 for Win32
/CD102W64	Delphi 10.2 for Win64
/CD102OSX	Delphi 10.2 for OSX
/CD102IOSDEV	Delphi 10.2 for iOS Device 32-bits
/CD102IOSDEV64	Delphi 10.2 for iOS Device 64-bits
/CD102IOSSIM	Delphi 10.2 for iOS Simulator

/CD102ANDROID	Delphi 10.2 for Android
/CD102LINUX64	Delphi 10.2 for Linux 64-bits
/CD103W32	Delphi 10.3 for Win32
/CD103W64	Delphi 10.3 for Win64
/CD103OSX	Delphi 10.3 for OSX 32-bits
/CD103OSX64	Delphi 10.3 for OSX 64-bits
/CD103IOSDEV	Delphi 10.3 for iOS Device 32-bits
/CD103IOSDEV64	Delphi 10.3 for iOS Device 64-bits
/CD103IOSSIM	Delphi 10.3 for iOS Simulator
/CD103ANDROID	Delphi 10.3 for Android 32-bits
/CD103ANDROID64	Delphi 10.3 for Android 64-bits
/CD103LINUX64	Delphi 10.3 for Linux 64-bits
/CD104W32	Delphi 10.4 for Win32
/CD104W64	Delphi 10.4 for Win64
/CD104OSX	Delphi 10.4 for OSX 32-bits
/CD104OSX64	Delphi 10.4 for OSX 64-bits
/CD104IOSDEV	Delphi 10.4 for iOS Device 32-bits
/CD104IOSDEV64	Delphi 10.4 for iOS Device 64-bits
/CD104IOSSIM	Delphi 10.4 for iOS Simulator
/CD104ANDROID	Delphi 10.4 for Android 32-bits
/CD104ANDROID64	Delphi 10.4 for Android 64-bits
/CD104LINUX64	Delphi 10.4 for Linux 64-bits
/CD11W32	Delphi 11 for Win32
/CD11W64	Delphi 11 for Win64
/CD11OSX	Delphi 11 for OSX 32-bits
/CD11OSX64	Delphi 11 for OSX 64-bits
/CD11IOSDEV	Delphi 11 for iOS Device 32-bits
/CD11IOSDEV64	Delphi 11 for iOS Device 64-bits
/CD11IOSSIM	Delphi 11 for iOS Simulator
/CD11ANDROID	Delphi 11 for Android 32-bits
/CD11ANDROID64	Delphi 11 for Android 64-bits
/CD11LINUX64	Delphi 11 for Linux 64-bits
/CD11OSXARM64	Delphi 11 for OSX ARM 64-bits
/CD12W32	Delphi 12 for Win32
/CD12W64	Delphi 12 for Win64
/CD12OSX	Delphi 12 for OSX 32-bits
/CD12OSX64	Delphi 12 for OSX 64-bits
/CD12IOSDEV	Delphi 12 for iOS Device 32-bits
/CD12IOSDEV64	Delphi 12 for iOS Device 64-bits
/CD12IOSSIM	Delphi 12 for iOS Simulator
/CD12ANDROID	Delphi 12 for Android 32-bits
/CD12ANDROID64	Delphi 12 for Android 64-bits
/CD12LINUX64	Delphi 12 for Linux 64-bits
/CD12OSXARM64	Delphi 12 for OSX ARM 64-bits

Options are read from PAB.INI. Some of the settings may be overridden by options on the command-line (see above).

The command-line version can, in contrast to the GUI version, also analyze source code without first creating a project. Just supply a source code path on the command-line instead of a project path. PABCMD will then use the settings according to the template which is used for new projects.

If an error occurs when PABCMD is run, the application terminates with exit code 99.

Example:

PABCMD c:\projects\MyProj.pbr

Runs PABCMD and analyses the project c:\projects\MyProj.pbr

PABCMD c:\Units\MyUnits.pas /FM- /CBP

Runs PABCMD and analyses c:\projects\MyProj.dpr with defaults as set in PAB.INI, but specifies that only the main file should be parsed, and that the compiler target is Borland Pascal 7.

See also:

How to use PAB.EXE and PAB32.EXE
Introduction
Main window
What's new in PAB 3.x (January 2018, updated December 2018)
What's new in PAB 2.x? (February 2014, updated May 2016)

6 Installation folders

Pascal Browser is typically installed with this folder structure:

C:\Program Files\Peganza\Pascal Browser 3

This folder contains executable files, help system etc.

C:\Documents and Settings\<acc>\Application Data\Peganza\Pascal Browser

This is where the PAB.INI file is stored. The INI file contains common settings for the environment.
When running under Windows Vista, this folder is instead C:\Users\<acc>\AppData\Roaming\Peganza\Pascal Browser.

C:\Documents and Settings\<acc>\My Documents\Pascal Browser\Templates\Default

This folder contains the default template files (*.xsl) and other files for the web site. These extra files can optionally be copied to the output folder when creating documentation. You must have write access to this folder, because it is used for storing temporary files.

C:\Documents and Settings\<acc>\My Documents\Pascal Browser\Projects

This folder contains the project files (*.pbr). You can store your projects in any folder, this is just the default folder.

C:\Documents and Settings\<acc>\My Documents\Pascal Browser\Projects\Output

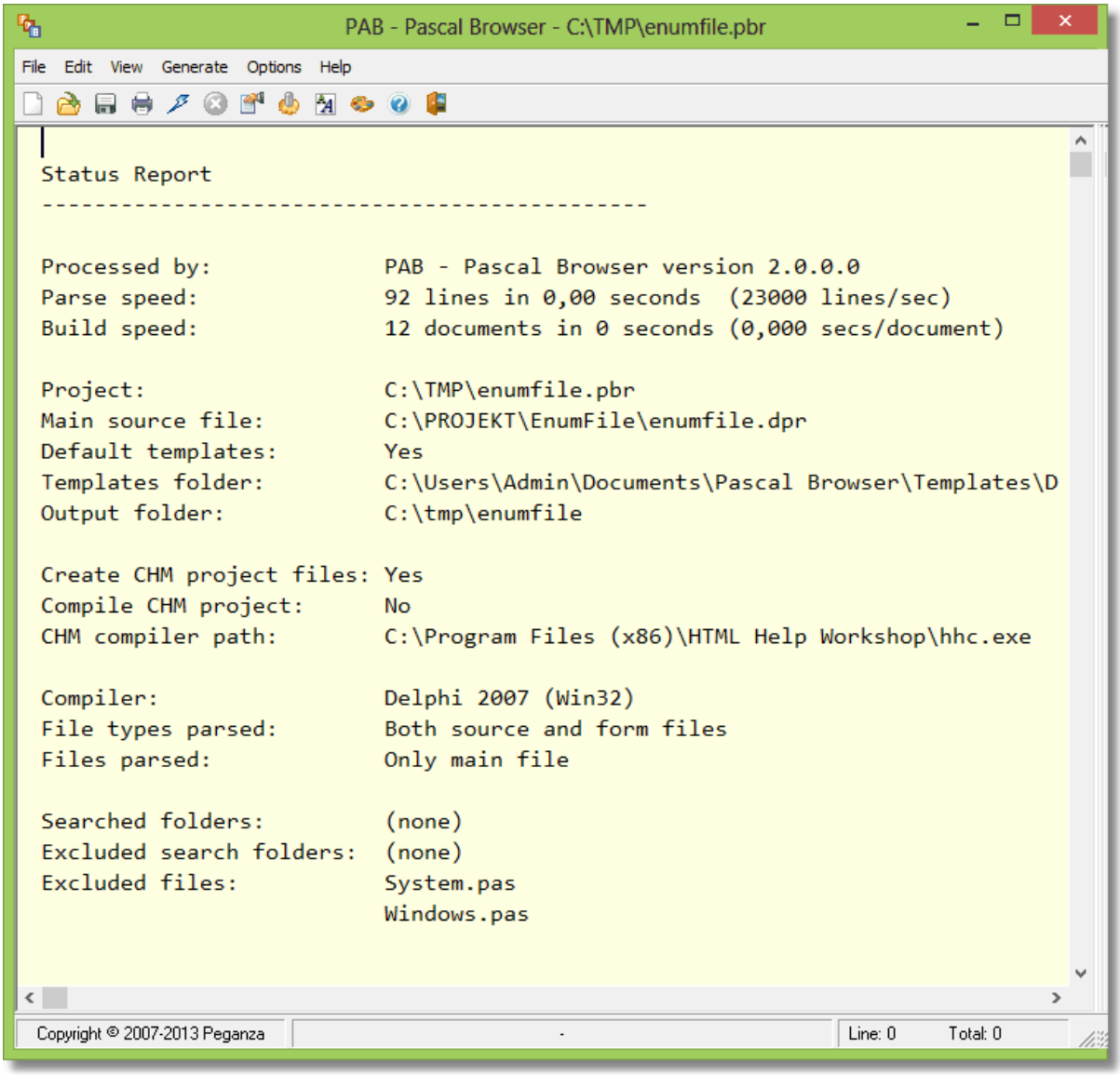
This folder is the default root folder for your documentation files. If you for example create documentation for the project MyProj, the output files will be found in the folder C:\Documents and Settings\<acc>\My Documents\Pascal Browser\Projects\Output\MyProj. Of course you are free to select any folder for the output.

See also:

How to use PAB.EXE
Introduction
Main window

7 Main window

The main interface of Pascal Browser consists of a menu, a toolbar, a report list window used to display the status report, and a status bar. The toolbar provides speed buttons for common menu items.



The main window in Pascal Browser

Toolbar

The toolbar contains speed buttons for some of the menu selections.

Status bar

The status bar presents different information, such as the estimated remaining time needed to create the documentation.

See also:

How to use PAB.EXE
Introduction

8 Templates

XML file

To create documentation, Pascal Browser first gathers information about the source code and outputs this to an XML file. You can select to keep this XML file, if you check the appropriate option on the Properties - General tab page. The name of the file is <projectname>.xml.

This XML file keeps data about each identifier that is reported in the project. Here follows a short sample for how a module is represented in XML code:

```
<Item kind="Module" id="106EDC0" modid="106EDC0" link="1">
  <Name>MyModule</Name>
  <Path>C:\PROJEKT\MyProj\MyModule.pas</Path>
  <Modified>2007-02-24 15:15:10</Modified>
  <Size>78</Size>
  <MainFile>1</MainFile>
  <HasMain>1</HasMain>
  <IsForm>0</IsForm>
  <HasIni>0</HasIni>
  <HasFin>0</HasFin>
  <Used>
    <Item id="1527DC0" interface="1"/>
    <Item id="1B2D4B0" interface="1"/>
    <Item id="2171630" interface="1"/>
    <Item id="1E4EC40" interface="1"/>
    <Item id="207AD80" interface="1"/>
  </Used>
</Item>
```

The XML code generated for each identifier depends on the kind of identifier that is written. This is a sample for a procedure:

```
<Item kind="Procedure" id="1072660" modid="1E4EC40" link="1">
  <Name>AddDataModule</Name>
  <ClaScope>Private</ClaScope>
  <OwnerTypeID>1597A60</OwnerTypeID>
  <Calls>
    <Item id="CBA120"/>
  </Calls>
  <CalledBy>
    <Item id="CD0A40"/>
  </CalledBy>
</Item>
```

The XML file also has a general section, describing the project, and a section for the todo items that are found in the project.

XSLT templates

Pascal Browser uses XSLT (XML Stylesheets) template files to create the documentation.

XSLT is an XML-based language designed to transform one XML document into another. When Pascal Browser generates documentation, these templates are transformed with the XML information for the source code, producing documents. The documentation that is produced by the default templates has HTML format.

The default templates consist of nine primary templates:

- Home.xsl
- Contents.xsl
- Index.xsl
- Overview.xsl
- ClassIndex.xsl
- IdentifierIndex.xsl
- ModuleIndex.xsl
- SubprogIndex.xsl
- Settings.xsl

The primary templates are processed first of all, and form the starting point for the rest of the documentation. Each of these templates produces a corresponding output file. This is done by transforming the XSLT stylesheet with the XML file. For example, **SubprogIndex.xsl** produces a **SubprogIndex.htm** file. The file extension is a setting that you can select in the Properties|Templates dialog box.

The names and numbers of these files are not fixed. You can create your own set of templates and use as many primary templates as you like. You can also edit the contents of these templates (but you should work on copies in another folder so the original templates are not damaged).

For each major source code identifier included in a primary template, a new page is created, based on one of the following seven identifier templates:

- Class.xsl
- Constant.xsl
- Interface.xsl
- Module.xsl
- Subprog.xsl
- Type.xsl
- Variable.xsl

If, for example, a page is produced for a function or a procedure, the identifier template **Subprog.xsl** will be applied.

The created document is given a file name on the format <ID>.htm, like 21FE672.htm where the number is a unique hexadecimal number for the identifier that is described in the document. In this document there may be links (URLs) to documents for other identifiers.

The names of the identifier templates are fixed, unlike the primary templates. But you are free to alter the contents of these identifier templates. As mentioned earlier, you should also in this case work on copies in another folder.

There are also some XSLT include files that are used both by the primary and identifier templates. You can also customize these files.

- FooterInc.xsl
- HeaderInc.xsl
- ImageInc.xsl
- InfoInc.xsl
- TodoInc.xsl
- ToolsInc.xsl

Images

Supplied with Pascal Browser is a collection of custom-tailored images (*.gif) files that you can use in your own documentation projects. Each image is available in three sizes, 16, 24 and 32 pixels. They are used by the default templates, and can be found in the C:\Documents and Settings\<acc>\My Documents\Pascal Browser\Templates\Default\Images folder.

Image	File name
	Class_XX.gif
	Codeblock_XX.gif
	Constant_XX.gif
	Constructor_XX.gif
	Destructor_XX.gif
	Field_XX.gif
	Function_XX.gif
	Interface_XX.gif
	Module_XX.gif
	Procedure_XX.gif
	Property_XX.gif
	Type_XX.gif
	Typed_Constant_XX.gif
	Variable_XX.gif

"XX" in the file name should be substituted by "16", "24" or "32", depending on the size of the image.

This is about all you will ever want to know, if you just use Pascal Browser just as it is. But you can create your own set of templates, and produce totally different forms of documentation, like text files, RTF files, or even XML files.

See also:

Customization
Examples
How to use PAB.EXE
Introduction
Main window

9 Customization

You can customize how the resulting documentation is generated. There are different levels of customization possible, ranging from very easy to advanced. It helps if you have some knowledge in these areas:

- CSS (cascading style sheets)
- HTML
- XML
- XSLT (XML stylesheets)

The customization levels are:

Level 1
Change the CSS style sheet that is used for the default HTML documentation. This file is in the templates folder. By editing this file you can change fonts, background colors and similar items. This is the easiest form of customization.

Level 2
Edit the XSLT parimary and identifier template files (medium difficult)

Level 3
Create your own set of XSLT files as primary templates (most difficult)

Pascal Browser creates the documentation files, normally a set of HTML files, by following this process:

1. The source code is parsed, interpreted, and information is collected about all identifiers.

2. An XML structure is created from this information.

3. For every primary template, Pascal Browser produces a corresponding file. For example, Index.xsl leads to the new file Index.htm. The new file is created by XSLT transformation, where the XSLT template is ”merged” with the XML data producing some output.

4. The resulting output file may contain links to other HTML pages. For example if the Start.htm files contain links to the units MyUnitA and MyUnitB, two new pages are created for these units. The identifier template Module.xsl is applied to create these pages, which will get names like 1EF25.htm and 33EA28.htm. This numeric name is used to ensure that two pages get unique names. These pages can themselves contain references to other identifiers, and so, pages are created for these identifiers as well. This process continues until all pages have been created.

Remember, if you plan to create your own templates, a recommended practice is to first make a copy of the default templates. Place these templates in your own template folder and start editing and/or create your own files.

In case you should damage the contents of the default templates, you will have to rerun the installation program to restore them.

Pascal Browser also applies some special processing before and after the XSLT transformation. Some special magic words are substituted with actual values. Most of the contents for the documentation is loaded directly from the XML structure, but some is generated “on-the-fly”. The reason is that it is not practical to store everything in the XML structure, like source-code listings.

PAB achieves this with so-called “magic words”. Those magic words are written in the XSL files on the format

{#MagicWord}

For example, the magic word {#AllModCallsHier} will be substituted with a module call hierarchy list.

You can use these magic words in your own templates. One category of these magic words start with the string “self”, like

{#selfID}

These magic words all refer to the active object that the template is currently applied to. At runtime these magic words will be substituted with values for the active object.

This is a list of all valid magic words (non-case-sensitive) in alphabetical order, and a description of their actions.

{#AllModCallsHier}
This word is substituted for a complete module call hierarchy, showing all modules. This word can be used in any template page.

Sample:

```
pab
 ├BGuard
 │  └BGlobals
 ├NGlobals
 └DMainFrm
    ├PAnaProg
    │  ├GAsmCode
    │  ├BDebug
...
```

{#CallsHier}
This word is substituted for a call hierarchy for the current subprogram.
The word can only be used in Subprog.xsl.

Sample:

```
TDsAppHandler.BuildProject
 ├TDsProjectHandler.PreAnalysis
 │  └TCodeHandler.Create
 │     ├TCodeHandler.DoUpdateCaption
 │     ├TCodeHandler.DoUpdateCounter
 │     └TCodeHandler.Initialize
…
```

{#CalledByHier}
This word is substituted for a called-by hierarchy for the current subprogram.
The word can only be used in Subprog.xsl.

Sample:

```
TTodoAppHandler.DoOnGeneralTimer
        TTodoAppHandler.Create┘
    TTodoMainForm.FormCreate┘
```

{#DfmHier}
This word is substituted for a hierarchical listing of the DFM objects for the current form class. The word can only be used in Class.xsl.

Sample:

```
FrPreviewDialog : TFrPreviewDialog
 ├pnlToolBar : TPanel
 │  ├btnZoomToFit : TfrSpeedButton
 │  ├btnZoomTo100 : TfrSpeedButton
 │  ├btnZoomToWidth : TfrSpeedButton
 │  ├btnFirst : TfrSpeedButton
 │  ├btnPrev : TfrSpeedButton
 │  ├btnNext : TfrSpeedButton
 │  ├btnLast : TfrSpeedButton
 │  ├btnPrint : TButton
 │  └btnClose : TButton
 └frPreview : TfrPreview
```

{#DocName}
This word is substituted for the current document file name, e.g. “342877.htm”

{#DocPath}
This word is substituted for the complete path to the document file, e.g. “C:\Output\MyProj\342877.htm”

{#DownHier}
This word is substituted for a downward class hierarchy for the current class. The word can only be used in Class.xsl.

{#Fin}
This word is substituted for the source code for the finalization section of the unit, and can only be used in Module.xsl

{#FullUpHier}
This word is substituted for a full upward class hierarchy for the current class. The word can only be used in Class.xsl.

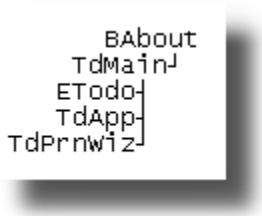
{#IdName}
This word is substituted for the name of the current identifier.

{#Init}
This word is substituted for the source code for the initialization section of the unit, and can only be used n Module.xsl.

{#MainCode}
This word is substituted for the source code for the main code of the unit, and can only be used in Module.xsl

{#ModCalledByHier[x]}
where x is a number 0-9.
This word is substituted for a module call hierarchy for the current module, which shows the modules that call the current module.
The keyword can only be used in Module.xsl. If x=0, then the entire complete hierarchy is shown, otherwise only the number of levels given by x.

Sample:



{#ModCallsHier[x]}
where x is a number 0-9.
This word is substituted for a module call hierarchy for the current module, which shows the modules called by the current module.

The keyword can only be used in Module.xsl. If x=0, then the entire complete hierarchy is shown, otherwise only the number of levels given by x.

Sample:

```
BAbout
 ├BGuard
 │  └BGlobals
 ├BGlobals
 └BUtils
    └BGlobals
```

{#selfHasSource}
This returns “1” or “0” depending on if the current identifier has any source code or not.

{#selfID}
This word is substituted with the ID for the current object.

{#Template}
This word is substituted for the name of the template file that is used for the document file, e.g. "Module.xsl".

{#TotalHier}
This word is substituted for a class hierarchy for all classes. The word can be used in any page.

Sample:

```
TObject
 ├TAbsElement
 │  ├TAbsBaseElement
 │  │  ├TAbsSubProg
…
```

{#UpHier}
This word is substituted for an upward class hierarchy for the current class. The word can only be used in **Class.xsl**.

{#UpDownHier}
This word is substituted for an upward and downward class hierarchy for the current class. The word can only be used in **Class.xsl**.

Sample:

```
TObject
 └TAbsElement
   └TAbsBaseElement
     └TAbsTypedElement
       └TAbsPascalVariable
         └TPascalRecordField
```

Special tags for comments

Comments in source code can also be used with special tags:

<#IMG> Insert an image

Example:

```
// <#IMG>"Images/PAB_16.gif" ALT="bla" TITLE="MyTitle"</#IMG>
```

Result:

The image will be inserted in the resulting HTML code

<#CODE> Format a code block

Example:

```
// <#CODE>
// procedure TheProc;
// var
//   I : integer;
// begin
//   MyProc(I);
// end;
// </#CODE>
```

Result:

The code block will be syntax formatted in the HTML page

<#URLA>/<#URLB> Insert a hyperlink (URL)

Example:

```
// <#URLA>"http://www.peganza.com"
TARGET="_blank"</#URLA><#URLB>Peganza</#URLB>
```

Result:

The URL will be inserted as a hyperlink in the HTML page

See also:

Templates
Examples
How to use PAB.EXE
Introduction
Main window

10 Examples

Here are some examples describing how you can change the output from Pascal Browser.

Change the default templates so that a different background color is used for the resulting HTML pages

Solution:

1. Open the DefStyle.css file in the default templates folder. The full path to this file is C:\Documents and Settings\<acc>\My Documents\Pascal Browser\Templates\Default\DefStyle.css.

2. Edit this section. customizing it the way you like:

```
BODY
{
   BACKGROUND: white;
   FONT-FAMILY: Verdana,Arial,Helvetica,sans-serif;
   FONT-SIZE: x-small;
}
```

3. Save DefStyle.css

Create a template for classes that produces separate lists for procedures and functions in the class

Solution:

1. First create a new folder for the new set of templates, for example: C:\Documents and Settings\<acc>\My Documents\Pascal Browser\Templates\MyTemplates

2. Copy all files from the default template folder to the new folder.

3. Open Class.xsl (in the new folder), in a text editor, or an editor suitable for XSLT files.

4. Change the section:

```
      <xsl:call-template name="table">
        <xsl:with-param name="filter" select=
"$ItemsInMe[(@kind='Procedure') or (@kind='Function')]"/>
        <xsl:with-param name="caption" select="'Methods'"/>
        <xsl:with-param name="bNameColumn" select="1"/>
        <xsl:with-param name="bScopeColumn" select="1"/>
        <xsl:with-param name="bCommentsColumn" select="1"/>
        <xsl:with-param name="bDeclarationColumn" select="1"/>
```

```
      </xsl:call-template>
```

to:

```
      <xsl:call-template name="table">
        <xsl:with-param name="filter" select
="$ItemsInMe[@kind='Procedure']"/>
        <xsl:with-param name="caption" select="'Procedures'"/>
        <xsl:with-param name="bNameColumn" select="1"/>
        <xsl:with-param name="bScopeColumn" select="1"/>
        <xsl:with-param name="bCommentsColumn" select="1"/>
        <xsl:with-param name="bDeclarationColumn" select="1"/>
      </xsl:call-template>

      <xsl:call-template name="table">
        <xsl:with-param name="filter" select="$ItemsInMe[@kind='Function']"
/>
        <xsl:with-param name="caption" select="'Functions'"/>
        <xsl:with-param name="bNameColumn" select="1"/>
        <xsl:with-param name="bScopeColumn" select="1"/>
        <xsl:with-param name="bCommentsColumn" select="1"/>
        <xsl:with-param name="bDeclarationColumn" select="1"/>
      </xsl:call-template>
```

11 Main menu

The main menu consists of these child menus:

11.1 File menu

File|New Project (Ctrl+N)

This command creates a new blank project. Settings from the template for projects will be used to initialize the new project.

File|Open (Ctrl+O)

Open an existing Pascal Browser project (*.pbr files). The currently opened project is active. There can only be one active project at any single time.

File|Reopen

Under this menu item you can reopen one of the latest opened projects.

File|Save (Ctrl+S)

This command saves the currently active project. The default folder for saving your projects is below "My Documents", in "C:\Document and Settings\<acc>\My Documents\Pascal Browser Projects".

File|Save As...

This command lets you save the currently active project with a new name.

File|Close

This command closes the currently active project.

File|Print (Ctrl+P)

This command prints the current status text that is shown in the main window after documentation has been generated.

File|Printer Setup

Show Windows standard printer setup dialog box.

File|Exit

Quit the application. Your settings are saved to the file PAB.INI in your settings folder.

See also:

Edit menu
View menu
Generate menu
Options menu
Help menu

11.2 Edit menu

Edit|Copy (Ctrl+C)

Copy the selected text to the clipboard.

Edit|Select all (Ctrl+A)

Select all text.

Search|Find (Ctrl+F)

Search for a string in the text.

Search|Search again (F3)

Repeat the last search.

See also:

File menu
View menu
Generate menu
Options menu

Help menu

11.3 View menu

View|Show Toolbar

Mark this menu item if you want the toolbar to appear.

Default = Yes

See also:

File menu
Edit menu
Generate menu
Options menu
Help menu

11.4 Generate menu

Generate|Run (Ctrl+R)

Select this command to let Pascal Browser generate the documentation for the currently active project. Be prepared that the process of generating all documentation can take some time.

For example, if one thousand HTML files will be produced, and each file takes a second to create, you will have to wait roughly 15 minutes till everything is done.

Also note that the number of files generated by Pascal Browser can be rather large, in thousands or tens of thousands. Remember, for each major identifier, Pascal Browser generates a document file. The number of files depends on the amount of source code that you make available for Pascal Browser, and which parts of the source code that you will allow it to produce documentation for.

After the run is completed, Pascal Browser will show a status text report in the main window. This report is also stored as a file _Status.txt in the output folder. The status report shows information about the generated documentation.

If an error occurs, there will be some additional files created in the output folder. The file names start with the underscore ("_") letter, so they will be easy to recognize. You may find additional clues to why the error occurred, by examining these files.

Generate|Stop

This command stops (cancels) the current documentation run.

Opening main document

After a completed run, this menu command will open the main document in the application that is associated with the file type.

See also:

File menu
Edit menu
View menu
Options menu
Help menu

11.5 Options menu

Options|Properties

Select properties for the currently active project. The properties are divided over a number of tab pages in the dialog window.

General Tab
Templates Tab
Source Tab
Parser Tab
Switches Tab

Click **OK** to confirm your selections and return to the main window.

When selecting paths, conditional defines and other parameters, a special selection dialog is shown:
In this dialog box, select items and add them to a list. For instance, to add a search folder, enter the path in the input field below the list box. Then press the Add-button to add the folder to the list. Click on the ellipsis button to select a folder by browsing. When selecting excluded folders, it is possible to mark a checkbox, meaning that also subfolders are excluded. Those folders will show up in the main dialog box with a suffix "<+>" attached.

Options|Set as default for new projects

Select this command to save the currently active project as a template. The template will be used to initialize new projects. The settings for the templates are stored in the PAB.INI file.

Options|Report Viewer font

Select a non-proportional font for the report viewer window. Default = Courier New 10

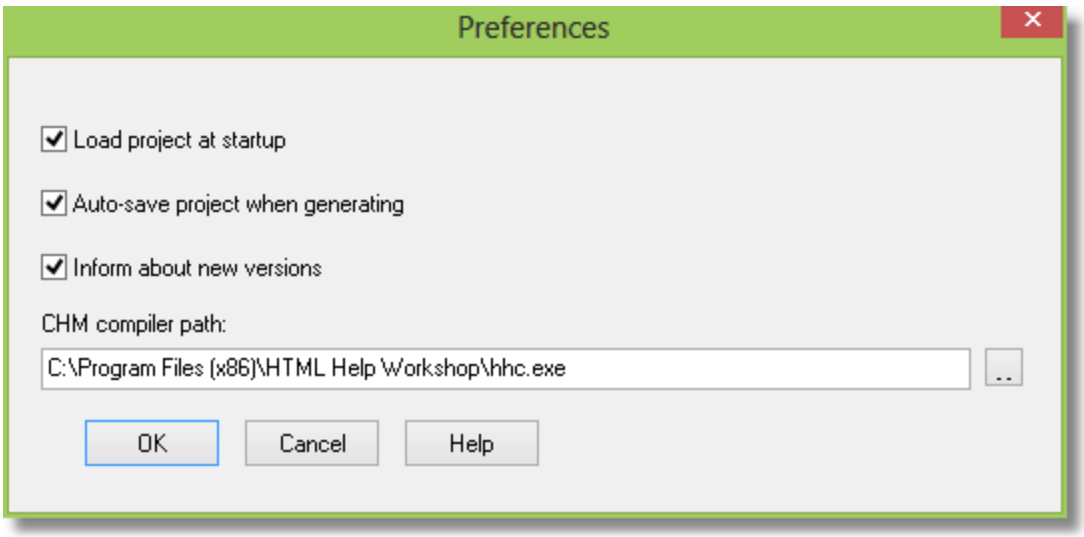
pt, black color

Options|Report Viewer color

Select a background color for the report viewer window. Default = White

Options|Preferences... (F12)

Select this command to set options that are common for all projects:



Load project at startup
Mark this checkbox if you want Pascal Browser to load the last used project at startup.

Default = Yes

Auto-save project when generating
Mark this checkbox if you want to automatically save the project file when the generation is started.

Default = Yes

Inform about new versions
Default = Yes

Mark this checkbox if you want to be notified when new versions of PAB are available. To protect your privacy, no identifying information is transferred from your computer when PAB checks for new versions. It is just a simple HTTP request to the Peganza web site to retrieve the latest product version number. If you do not want to check automatically by activating this option, there is a menu command under the Help menu that lets you check manually.

CHM compiler path

Edit the path to the hhc.exe help compiler if needed.

Default = <PROGRAMFILES>\HTML Help Workshop\hhc.exe

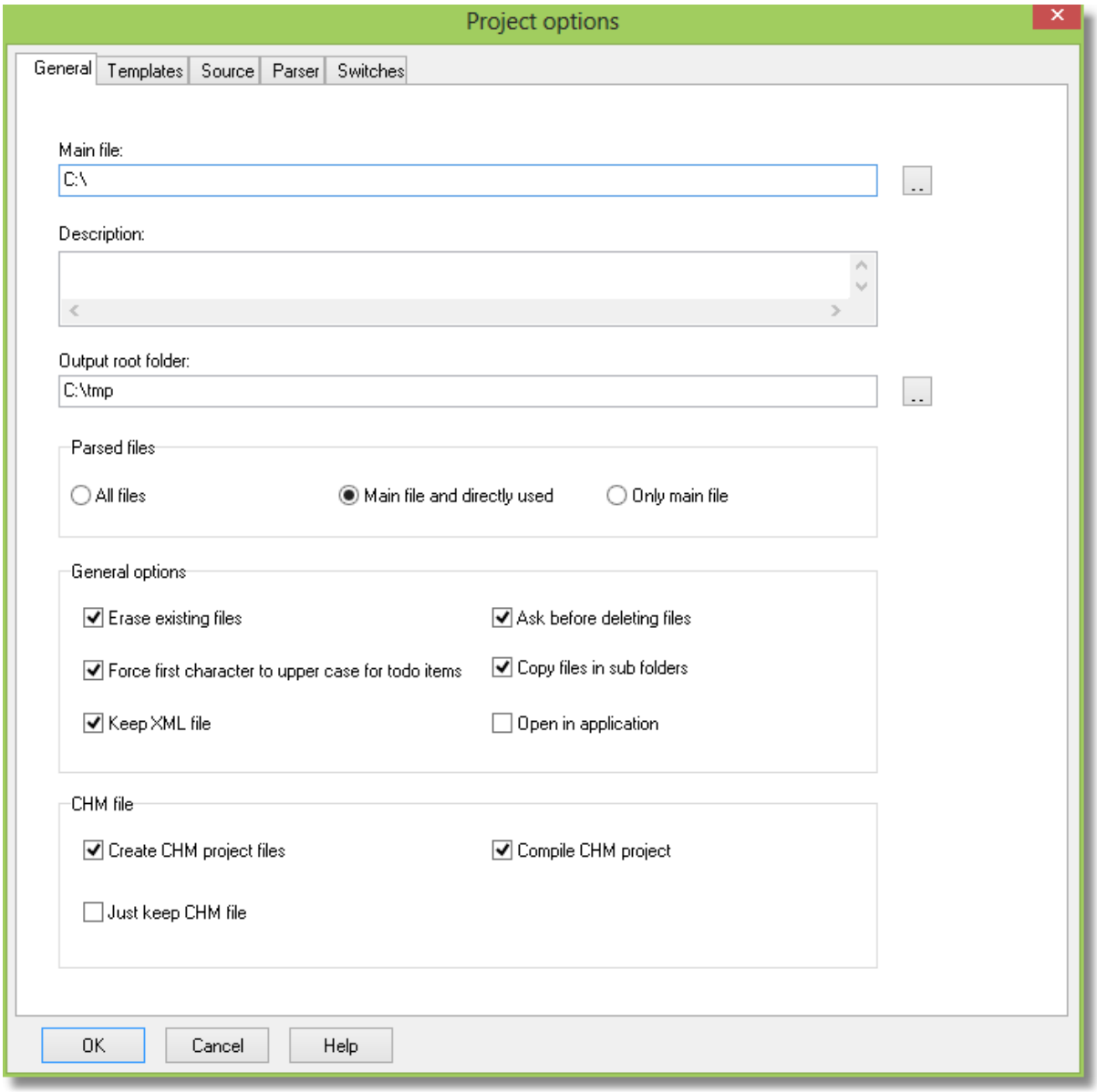
See also:

Properties - General
Properties - Templates
Properties - Source
Properties - Parser
Properties - Switches

File menu
Edit menu
View menu
Generate menu
Help menu

11.5.1 Properties - General

The General tab page lets you among other things, select which main file to generate documentation for.



The General tab page

Main file

Select either a single source file (*.PAS), or an entire Delphi project (DPR-file), or an entire Borland Pascal project (PAS-file). You can also select a Delphi package (DPK file).

You can also use a relative path. The path is then relative to the folder where the project file (PBR-file) is located.

Example:

The project file is saved at C:\Projects\MyProj.pbr. The main file is C:\Projects\Mine\MyProj.dpr.

You could use the relative path "..\Mine\MyProj.dpr" for the main file.

Description

Enter a descriptive string for this project. The string can be referenced by templates when documentation is created. It is included in the XML file that is generated.

Output root folder

Specify the root folder where Pascal Browser writes the documentation files. For example, if a project called “MyProject” is processed, the documentation files will be written to a folder “\MyProject” just below the output root folder. The folder will be created if it does not exist.

The default output root folder is in a folder below "My Documents", like "C:\Documents and Settings\<account>\My Documents\Pascal Browser\Projects\Output".

You can also use a relative path. The path is then relative to the folder where the project file (PBR-file) is located.

Example:

The project file is saved at C:\Projects\MyProj.pbr. If you want the output root folder in the same folder, just enter "." as the output root folder.

Parsed files

There are three possible settings:

All files
Pascal Browser will parse all found files if this option is selected.

NB. Selecting this option may force PAB to load and store a huge amount of identifiers and meta-data. This will make the generation process much slower. A good idea is to start with one of the other options, to see if they are sufficient to use.

Main files and directly used (default)
The main files and those files (units) that are listed in the uses lists of the main file, will be parsed.

Main file
Only the selected main file will be parsed.

Erase existing files

Mark this checkbox if you want Pascal Browser to first erase existing files in the output folder, before creating the new files. Only files matching the file extension selected, will be erased. For example, if "htm" is selected as the file extension, only files with this extension are affected. Any sub folders and files in those will also be deleted.

Because Pascal Browser does not necessarily create files with the same names each time you generate documents, it is advisable to turn on this option, so you do not get loose document files hanging around and wasting disk space.

Default = Yes

Ask before deleting files

Mark this checkbox if you want Pascal Browser to ask before deleting files recursively in the output folder. If you run PABCMD.EXE, the command-line version, there will be no question. Always check carefully that you have set your output folder correctly, so you do not inadvertently delete any files.

Default = Yes

Force first character to upper case for todo items

Mark this checkbox if you want the first letter of each todo item to be in upper case.

Default = Yes

Copy files in sub folders

When documentation is generated, files are copied from the template folder. Check this option to also copy files in sub folders.
Note that this could be a dangerous action, if you select the wrong template folder. If for example, "C:" is selected as the root folder, all files on your disk would be copied. So use this option with care.

Default = Yes

Keep XML file

Mark this checkbox if you want Pascal Browser to keep the generated XML file in the output folder. If you plan to customize or write your own XSLT templates, this file will be essential to you, so you would want to keep it. Otherwise Pascal Browser deletes it after the building process, because it is not longer needed.

Default = No

Open in application

Mark this checkbox if you want Pascal Browser to open the main document after completion of the generation. For example, if the main document is Start.htm, this page

will load the application you have associated with HTML-files, normally a web browser like MS Internet Explorer.
You can always access the main document with the menu selection "Open main document" in the Generate menu, regardless of this setting.

Default = Yes

Create CHM project files

Mark this checkbox if you want Pascal Browser to create CHM project files, in addition to the web site. This is especially handy if you have generated HTML files and want full-text search.

The CHM project files including the resulting CHM file will be written to the output directory. You can either work with the project files in HTML Help Workshop, or compile them directly from within Pascal Browser.

Default = No

Compile CHM project

Mark this checkbox if you want Pascal Browser to compile the CHM project, and produce a compressed CHM file. You must first download and install "HTML Help Workshop version 1.3" from Microsoft.

When compiling, a log file will be created in the output directory. It will have the name format **<ProjectName>.log**. Check this file if errors occur.

Default = No

Just keep CHM file

Mark this checkbox if you just want to keep the standalone CHM file, after compiling the CHM project. If Yes, all files in the output folder and all subfolders will be deleted, except for the CHM file.

The CHM file is produced by HTML files, so it is not necessary to keep those, if you just want to use the CHM file.

Default = No

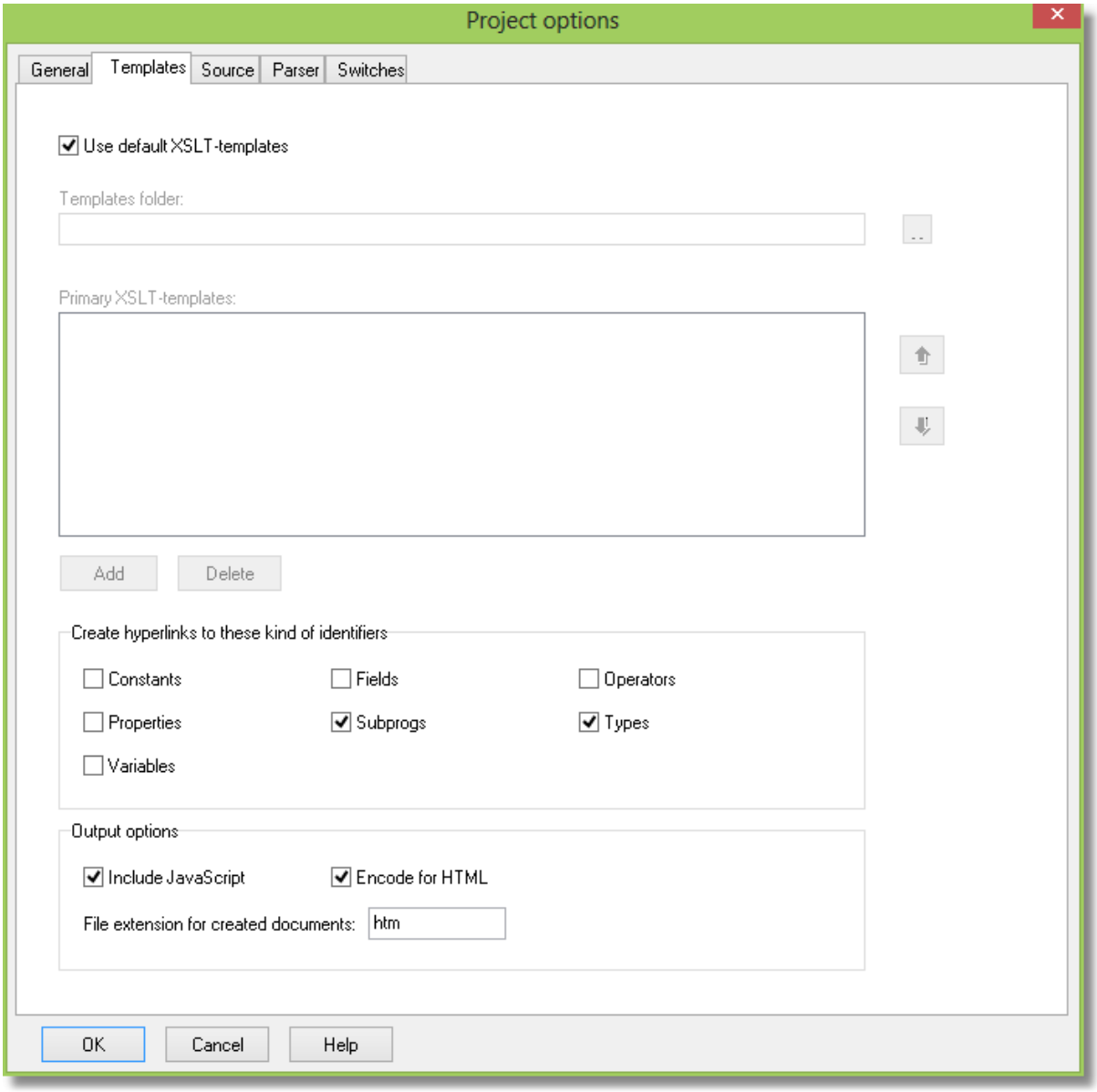
See also:

Options menu

Properties - Templates
Properties - Source
Properties - Parser
Properties - Switches

11.5.2 Properties - Templates

The second tab page is the Templates page:



The Templates tab page

Use default XSLT-templates

Mark this checkbox if you want to use the default templates in the "C:\Documents and Settings\<acc>\My Documents\Pascal Browser\Templates\Default" folder.

Default = Yes

Templates folder

This option is only available when not using the default templates.

Select the folder where the templates for the project are stored. The default templates are stored in a folder below "My Documents", in "C:\Documents and Settings\<account>\My Documents\Pascal Browser\Templates\Default".

When using your customized own templates, these should be located elsewhere than in the folder for default templates, maybe in a folder next to the "Default" template folder.

The folder you use for templates can also contain other files that are used by the produced documents, like GIF image files, CSS style sheets and JavaScript files. These files, if they are located in the templates folder, are automatically copied to the output folder when generating documentation. Files with the extension *.xsl are not copied however, because these are the template files, that are only used when producing other files.

Primary XSLT-templates

This option is only available when *not* using the default templates.

This list contains all primary templates. You can add and delete files by pressing the Add and Delete buttons. Primary templates are like the other templates located in the templates folder. Primary templates are XSLT files that will be processed when documentation is generated. Depending on their content, these templates will in turn invoke identifier templates.

It is possible to change the order for the templates. This affects the order in which they are processed. Also, the template that is at the top of the list is considered the main document. For example, if the top template is "Start.htm", this is the page that will be opened in your web browser when processing is complete (if you have checked the option "Open in application" on the Properties|General tab page.

Create hyperlinks to these kind of identifiers

Mark the checkbox for each kind of identifier that you want hyperlinked from source code listings. If for example, “Constants” is checked, links will be created to constants and the template **Constant.xsl** will be used to produce the documentation page. If “Constants” is not checked, these pages would not be created.

By *not* marking selected kinds of identifiers, you can reduce the number of created pages. It probably makes little sense however to deselect the main identifier types like subprograms and types, even if it is possible.

Note that local subprograms *are* hyperlinked for subprograms. Other kinds of local identifiers are not hyperlinked.

Include JS

Mark this checkbox if you want JavaScript code to be linked in on the generated HTML pages. JavaScript code will give special effects, like highlighting identifiers when the

mouse cursor is moved over the identifier.

Default = Yes

File extension for created documents

Enter the file extension that you want for the documentation files that are created. If you use the default templates, this setting is "htm", as in HTML files. If you produce other kinds of documents, you may need to change this setting.

Default = htm

Encode for HTML

Mark this checkbox if you want Pascal Browser to encode the output for HTML. This is needed when the documentation files are in HTML format.

Default = Yes

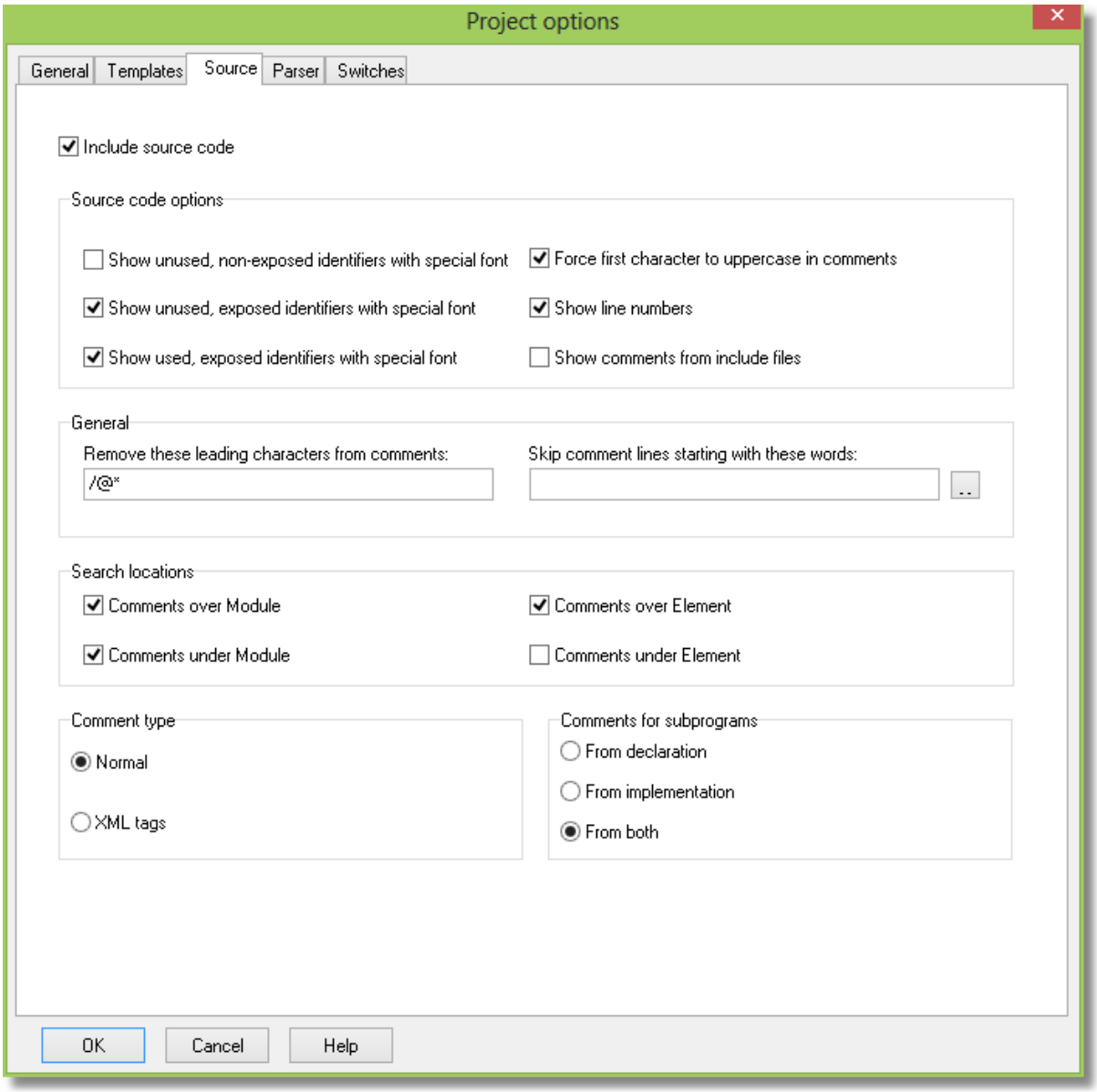
See also:

Options menu

Properties - General
Properties - Source
Properties - Parser
Properties - Switches

11.5.3 Properties - Source

The Source tab page lets you decide how source code is treated, and how comments are handled.



The Source tab page

Include source code

Mark this checkbox if you want Pascal Browser to include source code in the documentation. This happens if the necessary commands are present in the template files. If they are and this checkbox is not marked, no source code will appear in the documentation. This feature can be handy if you want to hide the source code, but still use your source code enabled templates.

Default = Yes

Show unused, non-exposed identifiers with special font

Mark this checkbox if you want Pascal Browser to use a special font for identifiers that are not used and not exposed.

Default = No

Show unused, exposed identifiers with special font

Mark this checkbox if you want Pascal Browser to use a special font for identifiers that are unused and exposed (mentioned in the interface section but only used in the implementation section).

If you are generating for a library project, you may want to turn off this option. Otherwise your exposed library routines will be shown as unused.

Default = Yes

Show used, exposed identifiers with special font

Mark this checkbox if you want Pascal Browser to use a special font for identifiers that are used but exposed (mentioned in the interface section but only used in the implementation section).

Default = Yes

Show line numbers

Mark this checkbox if you want line numbers in the left margin when source code is included in the documentation.

Default = Yes

Force first character to uppercase in comments

Mark this checkbox if you want the first character in each comment automatically converted to upper case.

Default = Yes

Example

```
...
type
  TMyClass = class // this is my class
  end;
...
```

Pascal Browser will present the comment as “This is my class”, if the checkbox is marked.

Show comments from include files

Select this option if you want Pascal Browser to also look for comments in include files, if the include directive is in a place where Pascal Browser looks for comments.

Default = Yes

Remove these leading characters from comments

Specify leading characters that should be removed from the comments.

Default = /@* (the characters “/”, “@” and “*” should be filtered out)

Example:

Consider this code:

```
...
type
// *** database record
  TMyRec = record
    Key : integer;
    Value : longint;
  end;
...
```

Pascal Browser will extract the text “database record” if “*” is included in the filter string.

Skip comment lines starting with these words

Specify words that cause a comment line to be skipped. The line will be skipped if the comment line starts with any of the specified words. For example if "ENDIF" is such a word all comment lines "// ENDIF" are skipped.

Default = (none)

Comments over Module

Mark this checkbox if you want Pascal Browser to include comments found above the “unit” keyword in the source code. Blank lines are allowed between the declaration and the comment.

Default = Yes

Example:

Consider this code:

```
// general routines
unit General;
// created for internal use
...
```

Pascal Browser will extract the text “general routines”

Comments under Module

Mark this checkbox if you want Pascal Browser to include comments found under the “unit” keyword in the source code. Blank lines are allowed between the declaration and the comment.

Default = Yes

Example:

Consider this code:

```
// general routines
unit General;
// created for internal use
...
```

Pascal Browser will extract the text “created for internal use”

Comments over Element

Mark this checkbox if you want Pascal Browser to include comments found above the declaration of the identifier in the source code. Blank lines are allowed between the declaration and the comment.

Default = Yes

Example:

Consider this code:

```
…
type
  // forward declaration
  TMyClass = class; // my class
  // needed
...
```

Pascal Browser will extract the text “forward declaration” if you mark this checkbox. The comment to the right of the declaration (“my class”) will always be extracted.

Comments under Element

Mark this checkbox if you want Pascal Browser to include comments found under the declaration of the identifier in the source code. Blank lines are allowed between the declaration and the comment.

For subprograms (functions and procedures) when comments are collected from the implementation code, the option is always turned off, regardless of this setting.

```
procedure MyProc;
begin
...
end;
// comments are not collected from this line for MyProc
```

Default = No

Example:

Consider this code:

```
...
type
  // forward declaration
  TMyClass = class; // my class
  // needed
...
```

Pascal Browser will extract the text “needed” if you mark this checkbox. The comment to the right of the declaration (“my class”) will always be extracted.

Comment type

Select if you want to collect comments either as normal, or as XML comments.

XML comments are mostly used in .NET code, and normally looks like:

```
/// <summary>
/// This is a summary comment
/// </summary>
```

Unlike in .NET C# and Delphi code, the XML comments do not have to start on lines with three consecutive "/".
Note that you can select XML comment type even if your code is not written for .NET.

Default = Normal

Comments for subprograms

Select if you want to collect comments for subprograms, from interface or implementation sections, or both.

Default = Both

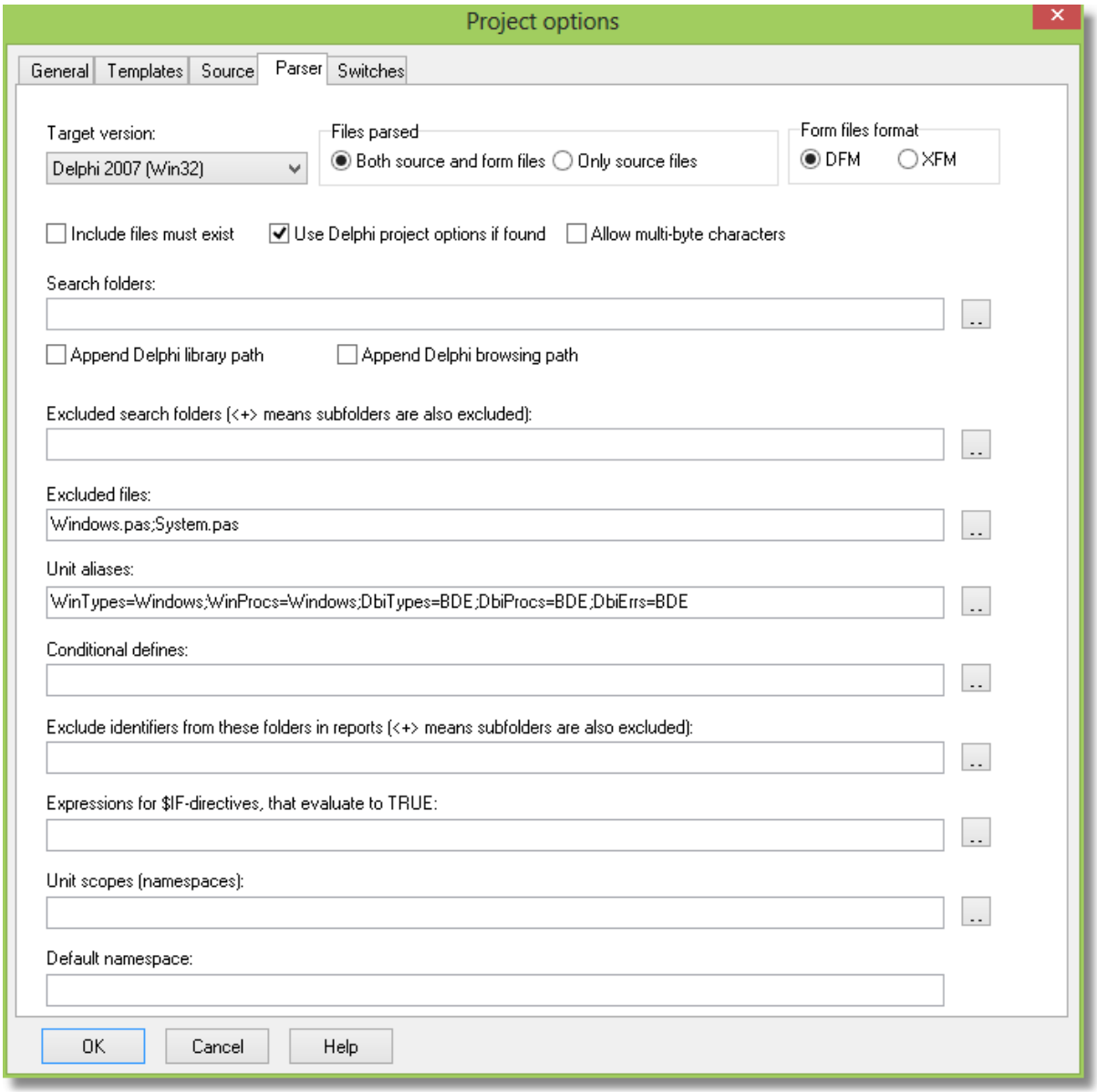
See also:

Options menu

Properties - General
Properties - Templates
Properties - Parser
Properties - Switches

11.5.4 Properties - Parser

The Parser tab page lets you decide which parts of the source that should be found and documented.



The Parser tab page

Target version

Select the compiler version targeted for the current documentation run. Note that, although Pascal Browser scans the source files in the same way as the compiler, it does not detect every syntax error. Make sure that the source code compiles correctly for the specified target, otherwise the results output by Pascal Browser may be incorrect.

Pascal Browser supports code written for these compilers/versions:

- Borland Pascal 7 (or earlier)
- Delphi 1
- Delphi 2
- Delphi 3
- Delphi 4
- Delphi 5
- Delphi 6
- Delphi 7
- Delphi 8 for .NET

- Delphi 2005 for Win32
- Delphi 2005 for .NET
- Delphi 2006 for Win32 (also Turbo Delphi for Win32)
- Delphi 2006 for .NET (also Turbo Delphi for .NET)
- Delphi 2007 for Win32
- Delphi 2007 for .NET
- Delphi 2009 for Win32
- Delphi 2010 for Win32

- Delphi XE for Win32

- Delphi XE2 for Win32
- Delphi XE2 for Win64
- Delphi XE2 for OSX

- Delphi XE3 for Win32
- Delphi XE3 for Win64
- Delphi XE3 for OSX

- Delphi XE4 for Win32
- Delphi XE4 for Win64
- Delphi XE4 for OSX
- Delphi XE4 for iOS Device
- Delphi XE4 for iOS Simulator

- Delphi XE5 for Win32
- Delphi XE5 for Win64
- Delphi XE5 for OSX
- Delphi XE5 for iOS Device
- Delphi XE5 for iOS Simulator
- Delphi XE5 for Android

- Delphi XE6 for Win32
- Delphi XE6 for Win64
- Delphi XE6 for OSX
- Delphi XE6 for iOS Device
- Delphi XE6 for iOS Simulator
- Delphi XE6 for Android

- Delphi XE7 for Win32
- Delphi XE7 for Win64
- Delphi XE7 for OSX

- Delphi XE7 for iOS Device
- Delphi XE7 for iOS Simulator
- Delphi XE7 for Android

- Delphi XE8 for Win32
- Delphi XE8 for Win64
- Delphi XE8 for OSX
- Delphi XE8 for iOS Device
- Delphi XE8 for iOS Device 64-bits
- Delphi XE8 for iOS Simulator
- Delphi XE8 for Android

- Delphi 10 for Win32
- Delphi 10 for Win64
- Delphi 10 for OSX
- Delphi 10 for iOS Device
- Delphi 10 for iOS Device 64-bits
- Delphi 10 for iOS Simulator
- Delphi 10 for Android

- Delphi 10.1 for Win32
- Delphi 10.1 for Win64
- Delphi 10.1 for OSX
- Delphi 10.1 for iOS Device
- Delphi 10.1 for iOS Device 64-bits
- Delphi 10.1 for iOS Simulator
- Delphi 10.1 for Android

- Delphi 10.2 for Win32
- Delphi 10.2 for Win64
- Delphi 10.2 for OSX
- Delphi 10.2 for iOS Device
- Delphi 10.2 for iOS Device 64-bits
- Delphi 10.2 for iOS Simulator
- Delphi 10.2 for Android
- Delphi 10.2 for Linux 64-bits

- Delphi 10.3 for Win32
- Delphi 10.3 for Win64
- Delphi 10.3 for OSX 32-bits
- Delphi 10.3 for OSX 64-bits
- Delphi 10.3 for iOS Device 32-bits
- Delphi 10.3 for iOS Device 64-bits
- Delphi 10.3 for iOS Simulator
- Delphi 10.3 for Android 32-bits
- Delphi 10.3 for Android 64-bits
- Delphi 10.3 for Linux 64-bits

- Delphi 10.4 for Win32
- Delphi 10.4 for Win64
- Delphi 10.4 for OSX 32-bits
- Delphi 10.4 for OSX 64-bits
- Delphi 10.4 for iOS Device 32-bits

- Delphi 10.4 for iOS Device 64-bits
- Delphi 10.4 for iOS Simulator
- Delphi 10.4 for Android 32-bits
- Delphi 10.4 for Android 64-bits
- Delphi 10.4 for Linux 64-bits

- Delphi 11 for Win32
- Delphi 11 for Win64
- Delphi 11 for OSX 32-bits
- Delphi 11 for OSX 64-bits
- Delphi 11 for iOS Device 32-bits
- Delphi 11 for iOS Device 64-bits
- Delphi 11 for iOS Simulator
- Delphi 11 for Android 32-bits
- Delphi 11 for Android 64-bits
- Delphi 11 for Linux 64-bits
- Delphi 11 for OSX ARM 64-bits

- Delphi 12 for Win32
- Delphi 12 for Win64
- Delphi 12 for OSX 32-bits
- Delphi 12 for OSX 64-bits
- Delphi 12 for iOS Device 32-bits
- Delphi 12 for iOS Device 64-bits
- Delphi 12 for iOS Simulator
- Delphi 12 for Android 32-bits
- Delphi 12 for Android 64-bits
- Delphi 12 for Linux 64-bits
- Delphi 12 for OSX ARM 64-bits

Default = Delphi 12 for Win64

Pascal Browser may also work with earlier versions of Turbo Pascal for DOS and Windows (prior to Borland Pascal 7), but this has not been validated and consequently is not guaranteed. In this case, select *Borland Pascal 7* for best results.

Files parsed

Select an option:

Both source and form files (DFM/FMX/XFM/NFM-files)
This is the default option. If Pascal Browser finds a form file, it will be examined together with the corresponding PAS-file.

Only source files
No form files will be examined.

Include files must exist

Mark this checkbox if a missing include file should trigger an error and stop the analysis. Keep this option selected if possible, since a vital missing include file could generate

incorrect results.

Default = No

Use Delphi project options if found

When a Delphi compiler and a DPR file is selected, Pascal Browser tries to load the corresponding project options file (CFG file in Delphi 4 or higher, DOF file in Delphi 2 and 3, and OPT file in Delphi 1). If successful, these options are used. For search paths, unit aliases and defines, the options are merged with the options you select. This makes it possible to instance, in Pascal Browser to provide the path to the VCL source files.

If a Delphi project file (DPR file) is parsed, search paths following the *in* keyword are automatically followed.

Allow multi-byte characters

Mark this checkbox if multi-byte characters (such as Shift-JS or Japanese) exist in source code comments.
Default = No

Search folders

Select the drives and folders where Pascal Browser will search for source files. The folder containing the primary source file is automatically searched, and there is no need to include this folder.

Unlike the compiler, Pascal Browser does not require that all source code is available. However, it is often best to make as much source code visible to Pascal Browser as possible.

Enter search folders, in priority order, separated with a semicolon e. g.:

c:\source\myunits;c:\source\generic;c:\source\proj1

Alternatively, press the ellipsis button to select the folders in a selection dialog box.

You may also enter relative paths, like "..\..\generic".

If a Delphi project or package file (DPR, or DPK file) is parsed, search paths following the *in* keyword are automatically followed.

Because the parser looks in the directories according to the order specified, it is wise to put more frequently used directories first in the list.

User-defined environmental variables set in the Delphi IDE, may also be used.

You can also use a relative path. The path is then relative to the folder where the project file (PBR-file) is located.

Append Delphi library path

Mark this checkbox if you want Pascal Browser to look for modules also in these folders. The Delphi library path is set in the Delphi IDE under Tools|Environment Options and the Library tab page.

Default = No

Append Delphi browsing path

Mark this checkbox if you want Pascal Browser to look for modules also in these folders. The Delphi browsing path is set in the Delphi IDE under Tools|Environment Options and the Library tab page.

Default = No

Exclude identifiers from these folders in documentation (but always report main file):

Identifiers declared in source code from these folders will NOT appear in the generated documentation.

One exception however is identifiers for types. For example when displaying types for identifiers, like "integer" or "TStringList", these are written out even if their folders are excluded.

Enter excluded folders separated with a semicolon e. g.:

c:\source\myunits;c:\source\generic

Alternatively, press the ellipsis button to select the folders in a dialog box.

It is possible to select that an exclude folder should also apply to its subfolders. In this way it is possible to exclude “C:\Program Files\Borland\Delphi7\Source” and all the subfolders. When subfolders are excluded, the folder name is suffixed with “<+>”.

User-defined environmental variables, like “$(UTILS)”, set in the Delphi IDE, may also be used.

You can also use a relative path. The path is then relative to the folder where the project file (PBR-file) is located.

Unit aliases

Select unit aliases (Delphi 2 and upwards).

Default for Win32 versions:
WinTypes=Windows;WinProcs=Windows;DbiTypes=BDE;DbiProcs=BDE;DbiErrs=BDE

Default for .NET versions:
WinTypes=Borland.Vcl.Windows;WinProcs=Borland.Vcl.Windows;DbiTypes=BDE;DbiProcs=BDE;DbiErrs=BDE

See the Delphi documentation for an explanation of unit aliases.

Conditional defines

Specify conditional compilation directives that are valid for the current analysis. Pascal Browser parses source code just like the compiler, and must treat conditional directives, meaning that it will ignore sections of not activated code. You may also include defines within the source code.

Enter conditional defines, separated with a semicolon e. g.:

Final;Special

Alternatively, press the ellipsis button to enter the defines in a dialog box.

Pascal Browser even initializes predefined defines just as the compiler does (e g 'WIN32'). It also keeps track of the state of all compiler directives so that directives like {$IFOPT} will function correctly.

The conditional compilation directive _PEGANZA_ is always defined.

Expressions for $IF-directives, that evaluate to TRUE

In Delphi 6, the new $IF-directive was introduced. The $IF-directive is followed by an expression, that evaluates to TRUE or FALSE. If you use $IF-directives, you must supply all expressions that evaluate to TRUE, because Pascal Browser cannot always determine the value of an expression. Enter the expressions separated with semicolons, like:

RTLVersion > 14;Declared(Windows)

Please observe that you do not need to include Defined-directives like "Defined(MSWINDOWS)", because PAB manages to evaluate those directives.

When Pascal Browser parser finds a $IF-directive in code, it will try to evaluate it. If it is an expression that you have supplied, it will be evaluated to TRUE, otherwise it will be evaluated as FALSE. Directives that are evaluated as FALSE, imply that the corresponding code is not activated.

Namespaces

This field is only enabled (and relevant) if the compiler is set to Delphi 8 or higher compilers for .NET. Press the ellipsis button to select the namespaces.

Default namespace

This field is only enabled (and relevant) if the compiler is set to Delphi 8 or higher compilers for .NET.

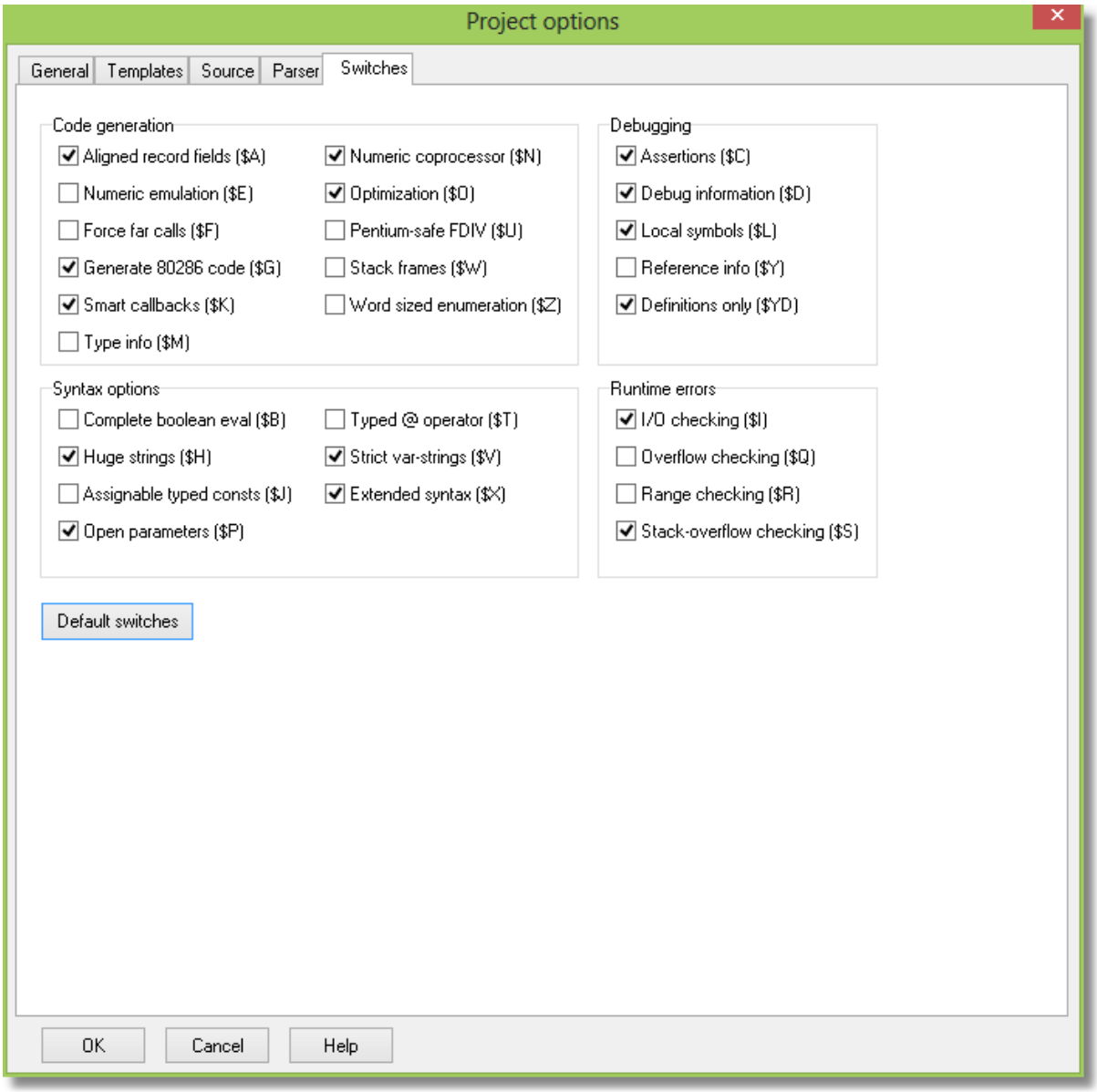
See also:

Options menu

Properties - General
Properties - Templates
Properties - Source
Properties - Switches

11.5.5 Properties - Switches

The last tab page is the Switches page:



The Switches tab page

On this tab, you can select which compiler switches that are active for the current analysis. This is important for how Pascal Browser evaluates $IFOPT directives. For instance, if $R+ (range check) is activated, a directive {$IFOPT R+} will evaluate to

TRUE, and the following code is activated.

A better method than setting these switches is to maintain a common include file used by all your modules. The purpose of the file is to set compiler directives and conditional defines that govern the compilation (and parsing by Pascal Browser). The include file directive $I is used to embed the contents of the include file into the source file.

See also:

Options menu

Properties - General
Properties - Templates
Properties - Source
Properties - Parser

11.6 Help menu

Help|Contents

This command displays the help system.

Help|Visit Peganza on the Internet

This command opens Peganza's home page in your web browser. Read the latest information about Pascal Browser and our other products.

Check for newer version

This command checks if there is a newer version of PAB available. This is the same kind of check that can be done automatically when starting the program, see Options menu.

Help|Deactivate License

If you need to move the installation to another computer, you can deactivate the license on the current computer. Then you will be able to activate the license on the new computer. The installation on the old computer will not longer run.

See the license.txt file in the program directory for more information.

Help|About

Displays the application’s About dialog box. You can here find, among other information, the version number.

Click "Refresh" to update the information about your support plan, and when it expires.

See also:

File menu
Edit menu
View menu
Generate menu
Options menu

Index

- $ -

- * -

- < -

- A -

- B -

- C -

- D -

- E -

- F -

- G -

- H -

- I -