

[sewiki.iai.uni-bonn.de](http://sewiki.iai.uni-bonn.de)

---

# Working with PDT (all-in-one version) [SE-Wiki]

10-13 minutes

---

## Syntax Highlighting


Depending on their properties, predicates, quoted atoms and comments are highlighted with a specific colours. These colours can be modified in the [preferences](#).


## Name Highlighting

Occurrences of atom and variable names are highlighted in the [current clause](#). The highlighting is updated instantly, upon typing.

## Warnings and Errors

All SWI-Prolog errors and warnings that occur while consulting a file are displayed in the Prolog Editor:

- **Warnings** are highlighted by a yellow underline and  markers in the sidebars of the editor.

- **Syntax errors** are highlighted by a red underline and  markers in the sidebars of the editor.

These errors and warnings are also displayed in the Eclipse Package Explorer and Eclipse Problems View:

## Saving and consulting files

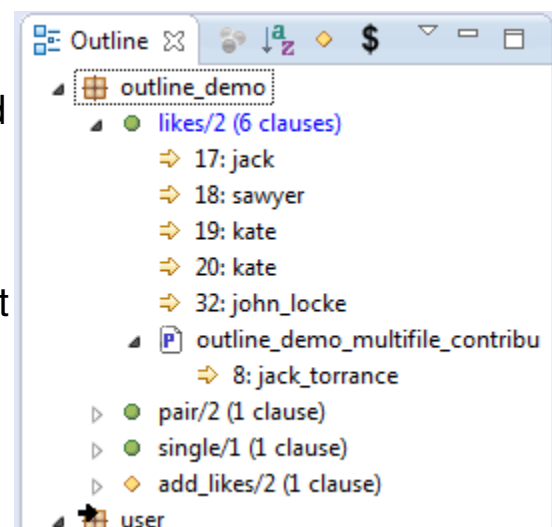
The file which is opened in the editor can be consulted to the current process by pressing **F9**.


Whenever you save a file in the editor it is automatically (re)consulted. If you don't want a file to be consulted after saving you can save the file by pressing **Ctrl+Alt+S** or by selecting the menu item "Save without consult". [1\)](#)

Furthermore you have the possibility to deactivate the reconsult on save in the [Preferences](#). This means, that files will only be consulted manually.

## Outline

The Outline view displays an outline of all predicates defined in the Prolog file opened in the editor. The Outline supports **multi-file awareness**, that is, it shows *all* the clauses of a predicate, also for multifile predicates:



- **First level:** Modules.  [message\\_hook/3 \(15 clauses\)](#)
- **Second level:** Predicate indicators of predicates defined in this file and total number of clauses of each predicate (in this and in other files). Dynamic predicates are emphasized with blue text colour. The icon to the left of the predicate indicator represents the visibility of the predicate: A green icon indicates exported predicates and predicates in non-module files. A yellow icon indicates non-exported predicates of a module.
- **Third level:**
  - Top: Clauses of the predicate defined in the local file (each clause with its first line number and its first argument. If the argument is a function term it is presented as Functor/Arity.
  - Rest: Other files that contain multifile contributions to the predicate.
- **Fourth level:** Clauses in each file listed at the third level (= multifile contributions from other files). Each clause is displayed with its first line number and its first argument. If the argument is a function term it is presented as Functor/Arity.

In the outline, multifile predicates are shown either as “for” or “from” contributions:
- **For:** If the [local file](#) contains predicates for [remote modules](#) the remote modules are displayed in the outline with a black arrow overlay on the module icon. Below each remote module icon, its [local predicates](#) are displayed with all their clauses: The [local clauses](#) at the top, followed by the other files that

contribute clauses (at the third level) and the contributed contributed clauses (at the fourth level).

- **From:** If local predicates from other files are listed below a file icon. The local clauses are displayed immediately below the predicate icon, no matter whether they are for the local module or another module.

A single click on the icon of a **local** predicate or clause selects the first line of that predicate / clause in the associated editor.

A single click on the icon of [remote clause](#) has no effect.

Double clicking a remote clause shows its first line **in another editor**. Single click is equivalent to navigating via the arrow keys, double click is equivalent to hitting <Return>.

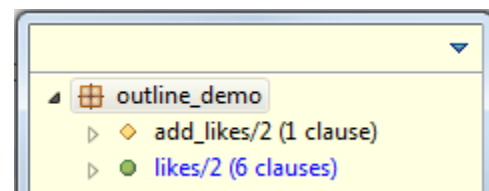
The order of modules in the outline is

1. local module first
2. then [remote modules](#) in alphabetic order

Within each module's section, predicates are displayed by default according to the textual order in the local file. However, the predicates can also be sorted alphabetically using the corresponding icon in the outline's tool bar. The order of modules cannot be changed.

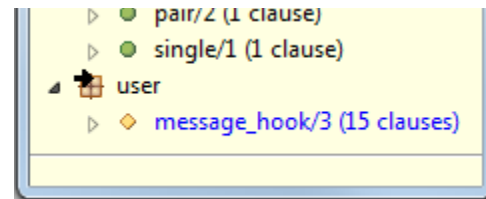
## Quick Outline

In the Prolog Editor a quick outline is available via **Ctrl+O** or via the



context menu.

When nothing is typed into the input field, the quick outline shows






the same content as the outline view. Typing retains only the predicates whose name contains the typed string. Hitting **Return** when there is just one choice left jumps to that predicate.

## Code Completion

The Prolog Editor supports code completion dependent on the current file and the loaded code in the currently selected Prolog process. Code completion in the Prolog editor uses the default Eclipse key binding (**<Ctrl><Space>**). The following elements are shown in the completion:

- **Predicates** (●, ◆, ■, ⊙): All matching predicates that are visible in the module of the current file.
- If the prefix is qualified with a module, all matching predicates visible in the qualifying module are listed.
- If a documentation of a predicate is available, it is shown in the right window.
- The icon shows the visibility of the predicate.
- The text inserted as completion depends on the key mask when accepting a proposal:
- **No key mask:** A predicate head is inserted. It is possible to

navigate through the arguments with **<TAB>**. If a documentation is available, the argument names defined in the documentation will be used as variables for the inserted head.

- **CTRL**: A predicate indicator (*Functor / Arity*) is inserted.
- **Modules** (): all matching modules.
- **atoms** (): all matching atoms currently known in the prolog system.
- **logtalk templates** (): all matching logtalk templates for entities, directives, etc. only available in logtalk files.

the following table shows some examples for the completion (click on the images to enlarge).

## Open Primary Declaration

**Open Primary Definition or Declaration** offers the possibility to easily jump to the first clause of a selected predicate, similar to the “Open Declaration” command in the Java editor. In the example above the declaration is contained in the same file and therefore the cursor will jump to line 17. If the declaration of the predicate is contained in another file, the other file will be opened in new editor.

- In case of multifile predicates the first clause which is found will be opened in the editor. Additionally the [find definitions and declarations](#) search will be started to display all clauses.

- If the selected predicate exists but there is no source for it, a message will be displayed.
- If the selected predicate does not exist or is not visible locally, the editor will offer a list of similar predicates or show a warning if there are no similar predicates.

## Find Definitions and Declarations

[Search](#) for all *declarations and definitions* of the currently selected *predicate* in the editor. This search is done using the currently selected prolog process in the [Prolog Console](#). The resulting modules get a visibility depending on the context of the selected goal. Possible values for visibility are *local*, *super*, *sub* and *invisible*.

## Find References

[Search](#) for all *references* to the currently selected *predicate* in the editor. This search is done using the currently selected prolog process in the [Prolog Console](#). Each result represents one clause that contains a call to the selected predicate.

## Breakpoints

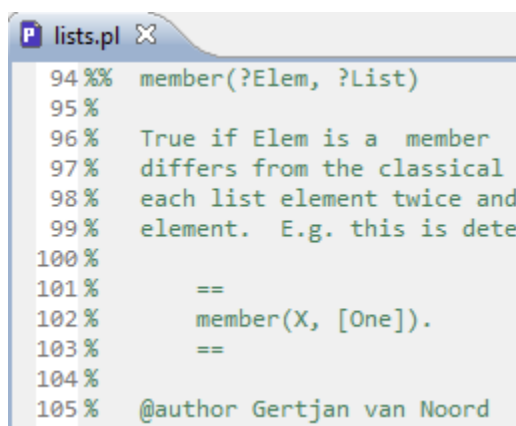
SWI Prolog offers the possibility of setting breakpoints in the prolog code. These breakpoints are visible in the [SWI GUI Tracer](#) and they can be set and removed within the [GUI Tracer](#). The PDT Prolog Editor also displays these breakpoints. These

breakpoints are those from the currently selected prolog process in the [Prolog Console](#). In the PDT Prolog Editor, one line in the editor can only have one breakpoint. It is possible to set and remove breakpoints directly from the editor. This can be done by double-clicking at the required location on the bar at the left edge of the editor or by using the context menu at the same location (see the screenshots below).

Setting a breakpoint in the editor means setting the breakpoint in the currently selected prolog process in the [Prolog Console](#). Therefore the file in the editor must be consulted in this prolog process, otherwise setting of breakpoints is not possible.

## External files

Files which are not contained in any project in the workspace are called **external files**. For example the prolog code of SWI Prolog is usually not contained in any project in the workspace. To emphasize that a file in the Prolog Editor is an external file the editor uses a different icon in the editor title bar and the background colour of the editor is different (light gray).

A screenshot of the PDT Prolog Editor window. The title bar shows a purple icon with a 'P' and the filename 'lists.pl' followed by a close button. The editor area has a light gray background and contains Prolog code with line numbers 94 through 105. The code defines a 'member' predicate and includes a comment about its difference from classical Prolog, as well as an author tag '@author Gertjan van Noord'.

```
94 %% member(?Elem, ?List)
95 %
96 % True if Elem is a member
97 % differs from the classical
98 % each list element twice and
99 % element. E.g. this is dete
100 %
101 % ==
102 % member(X, [One]).
103 % ==
104 %
105 % @author Gertjan van Noord
```

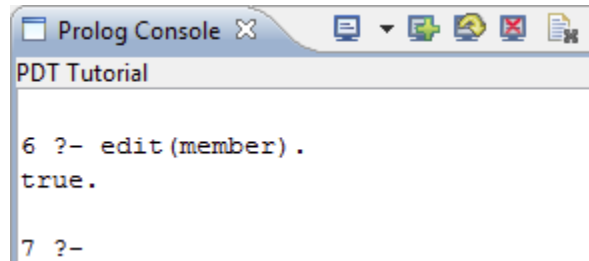


```
106  
107 member(E1, [H|T]) :-  
108     member_(T, E1, H).
```

## Open a file with edit/1

The SWI-Prolog predicate [edit/1](#) can be used in the [Prolog Console](#) to open a [Prolog editor](#) for any currently loaded predicate

that has attached source code. This applies also to predicates residing in [external files](#).



```
Prolog Console X  
PDT Tutorial  
6 ?- edit(member).  
true.  
7 ?-
```

E. g. the query “?- edit(member).” will open the library file “lists.pl” and position the cursor in the first clause of member/2 (see screenshots).

Even system files, that is, libraries that are part of the PDT or your Prolog system can be edited this way. However, this is something for experts. You should be very sure you know what you are doing.

Even then, please be aware that the default behaviour of saving an edited file includes reconsulting the file. So it is very easy to break a running system if you edit a system file and save it before editing another file that would need a consistent change.

If you want to perform a set of related changes in different system files use <Ctrl> <Alt> S for saving, disabling the auto-consult for that particular action. You can also change the

default behaviour of <Ctrl> S in the PDT preferences.