Release Notes Documentation

Release 6.2

AdaCore

Dec 11, 2018

CONTENTS

Release date: June 2016

CHAPTER

ONE

GNATDOC

GPS comes with a new engine for documentation generation. This comes in the form of a command-line tool called GNATdoc.

Amongst the features of GNATdoc are:

- support of Javadoc/Doxygen style of tags in documentation comments
- support for comment placement detection
- support for separating documentation comments from code comments
- a new extensible HTML back-end
  - support for users defined image directory
  - support for package groups

Here is an example of code annotated with GNATdoc tags:

```
function Set_Alarm
  (Message : String;
   --  The text to display

   Minutes : Natural
   --  The number of minutes to wait
  ) return Boolean;
--  Display a message after the given time.
--  @exception System.Assertions.Assert_Failure raised
--     if Minutes = 0 or Minutes > 300 if Minutes = 0
--  @return True iff the alarm was successfully registered
```

GNATdoc is available from the GPS interface, and also as a command-line tool, so it can be easily automated.

The full documentation for GNATdoc can be found at http://www.adacore.com/developers/documentation/gnatdoc-users-guide/.

CHAPTER

TWO

EDITORS

We have a new preference for showing only some line numbers. c .. NF-61-MC02-016 GPS: Automatic indentation of pasted content (2013-12-17)

When you paste content in a source editor that supports automatic indentation, it will be indented automatically, provided you have switched the feature on via the preferences dialog (Editor -> Auto indent on paste)

The dialog that pops up when a file has been modified on the disk has been modified. It will now list all such files in a single dialog (as opposed to having one dialog per file when it gets the focus). There is now also an option for automatically reloading files (which can be undone).

The dialog is also smart enough not to pop-up if the file on disk has the same contents as the GPS the editor.

2.1 Multicursors

We have made a number of improvements in the handling of multicursors.

When multiple cursors are active, it is now possible to select simultaneously with all cursors as you would with one. It is also possible to cut/copy/paste with multicursors. If the copy buffer has been filled with the same cursors, then the content of each individual cursor will be pasted. If the buffer was filled with only the main cursor, or with other cursors in a precedent operation, the content of the main buffer will be replicated on every cursor's location. Cut and copy works as you would expect.

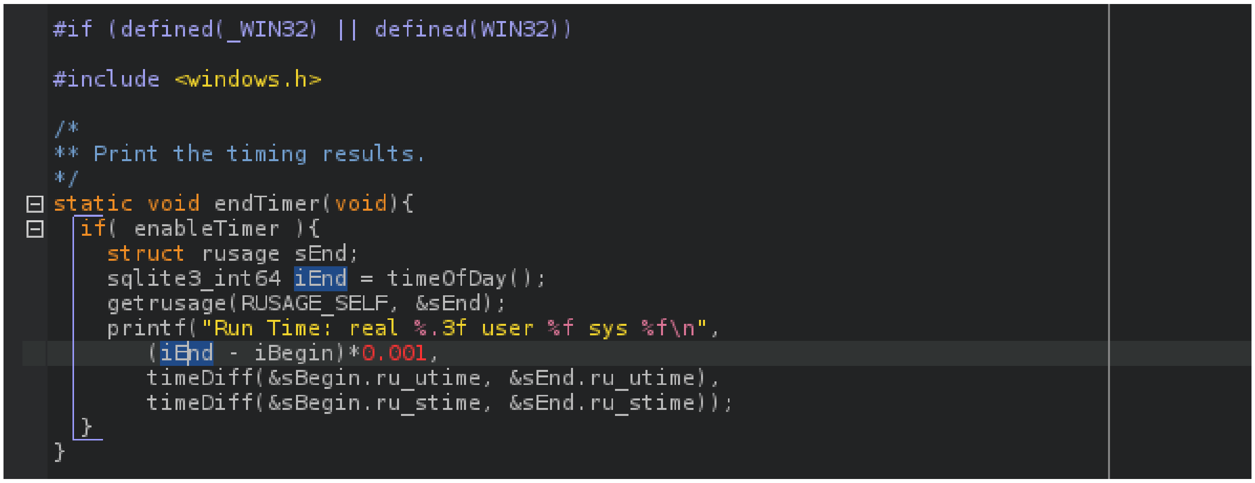
2.2 There's more

Several other improvements in the editors: the "Delete Line" actions are now atomic, there is a new plugin vim.py to emulate a few behaviors from vim. There is also a preference to change the background color of expanded code.

CHAPTER THREE

C SUPPORT

We have a new syntax highlighting engine for C, which has a number of nice capabilities: for instance it can highlight escape sequences in strings.



```
#if (defined(_WIN32) || defined(WIN32))

#include <windows.h>

/*
** Print the timing results.
*/
static void endTimer(void){
  if( enableTimer ){
    struct rusage sEnd;
    sqlite3_int64 iEnd = timeOfDay();
    getrusage(RUSAGE_SELF, &sEnd);
    printf("Run Time: real %.3f user %f sys %f\n",
       (iEnd - iBegin)*0.001,
       timeDiff(&sBegin.ru_utime, &sEnd.ru_utime),
       timeDiff(&sBegin.ru_stime, &sEnd.ru_stime));
  }
}
```

It is also entirely written in Python and is easy to extend.

CHAPTER

FOUR

ADA SUPPORT

Several editor enhancements for Ada in general:

- the alignment action preserves the selection
- improved auto-alignment for comments
- new block completions are available

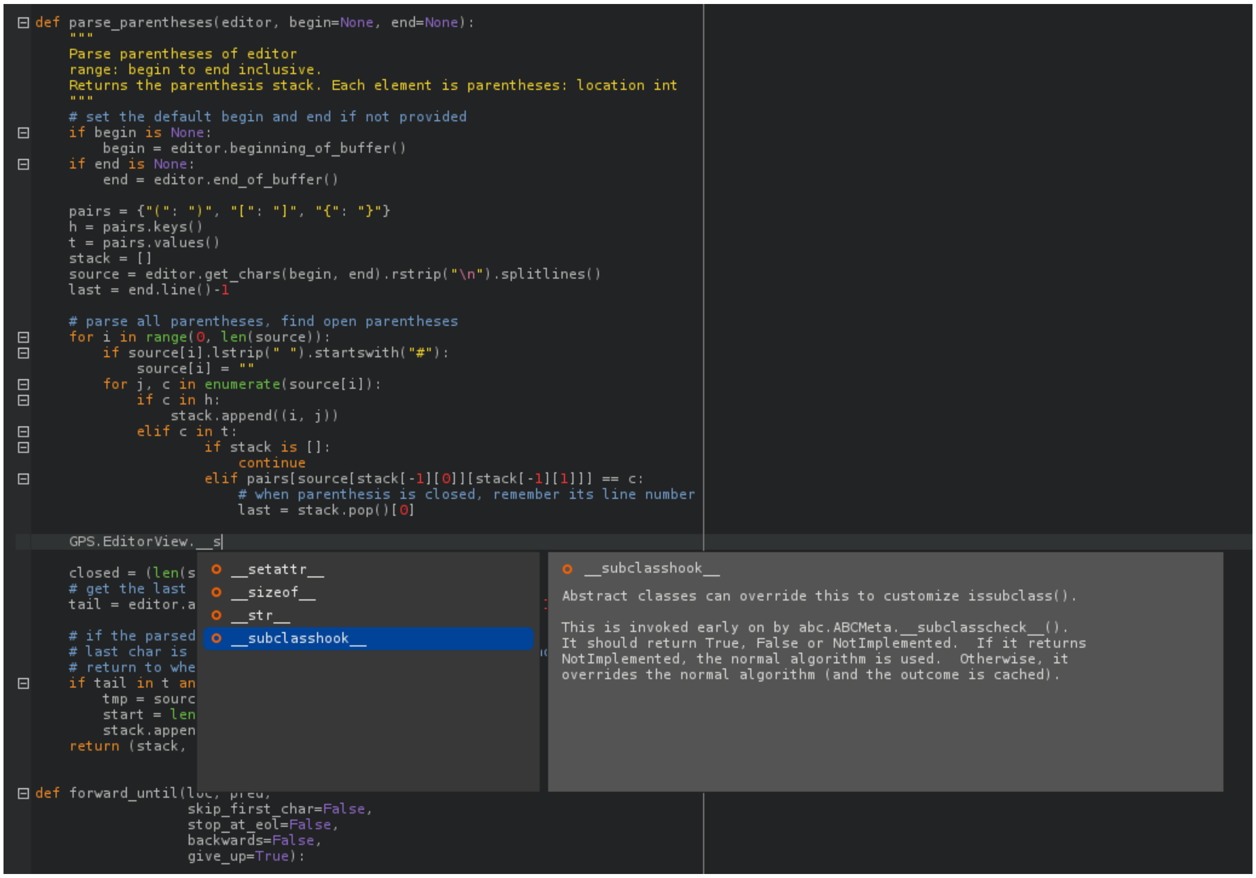
4.1 Ada 2012 and SPARK 2014

We have improved the auto indentation support for conditional expressions and subtype predicates.

FIVE

PYTHON

This release of GPS contains a lot of improvements to the support of Python.



```
def parse_parentheses(editor, begin=None, end=None):
    """
    Parse parentheses of editor
    range: begin to end inclusive.
    Returns the parenthesis stack. Each element is parentheses: location int
    """
    # set the default begin and end if not provided
    if begin is None:
        begin = editor.beginning_of_buffer()
    if end is None:
        end = editor.end_of_buffer()

    pairs = {"(": ")", "[": "]", "{": "}"}
    h = pairs.keys()
    t = pairs.values()
    stack = []
    source = editor.get_chars(begin, end).rstrip("\n").splitlines()
    last = end.line()-1

    # parse all parentheses, find open parentheses
    for i in range(0, len(source)):
        if source[i].lstrip(" ").startswith("#"):
            source[i] = ""
        for j, c in enumerate(source[i]):
            if c in h:
                stack.append((i, j))
            elif c in t:
                    if stack is []:
                        continue
                    elif pairs[source[stack[-1][0]][stack[-1][1]]] == c:
                        # when parenthesis is closed, remember its line number
                        last = stack.pop()[0]

    GPS.EditorView.__s

    closed = (len(s
    # get the last
    tail = editor.a

    # if the parsed
    # last char is
    # return to whe
    if tail in t an
        tmp = sourc
        start = len
        stack.appen
    return (stack,

def forward_until(loc, pred,
                  skip_first_char=False,
                  stop_at_eol=False,
                  backwards=False,
                  give_up=True):
```

__subclasshook__

Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

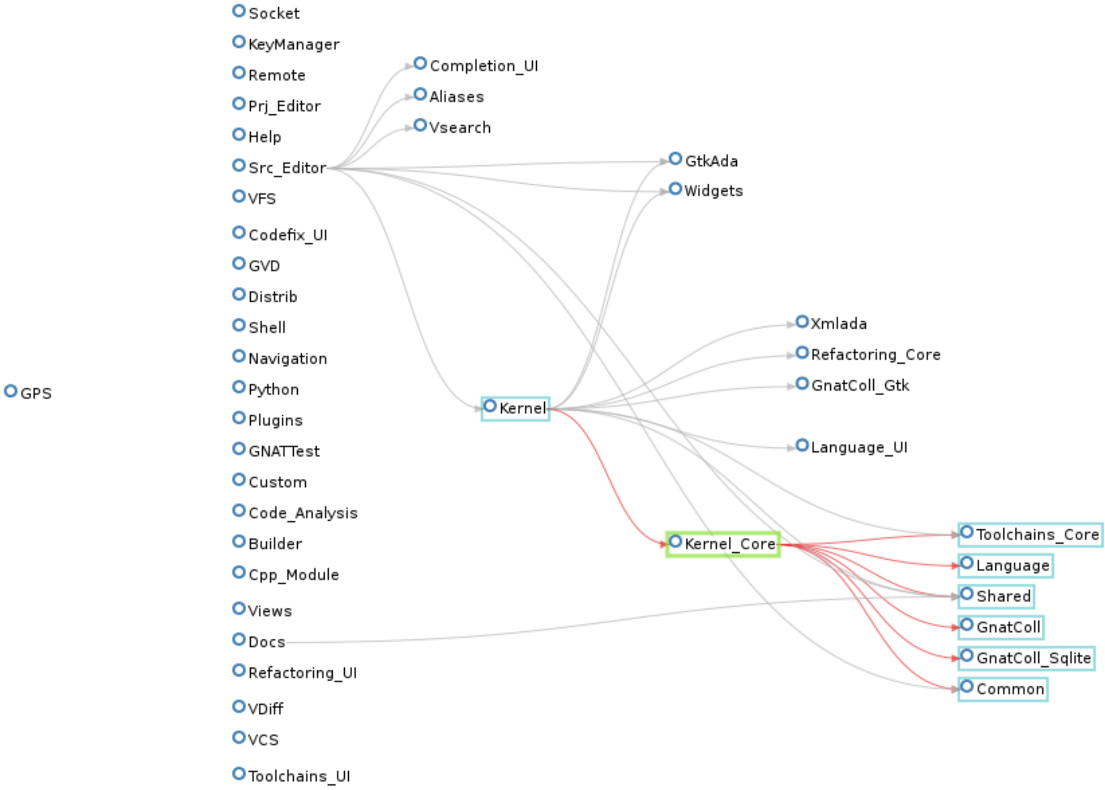
GPS now provides smart completion in Python files, providing completion for modules found in the standard Python search paths and all source directories for projects that list “Python” as their language. This is done through an integration of the Jedi library: https://github.com/davidhalter/jedi.

We also support auto-indentation in Python, and on-the-fly reporting of syntax and style errors.

CHAPTER SIX

BROWSERS

We have completely rewritten the engine for rendering browsers. This gives a brand new look and feel to the Project browser, the Entity browser and the call graph browser.

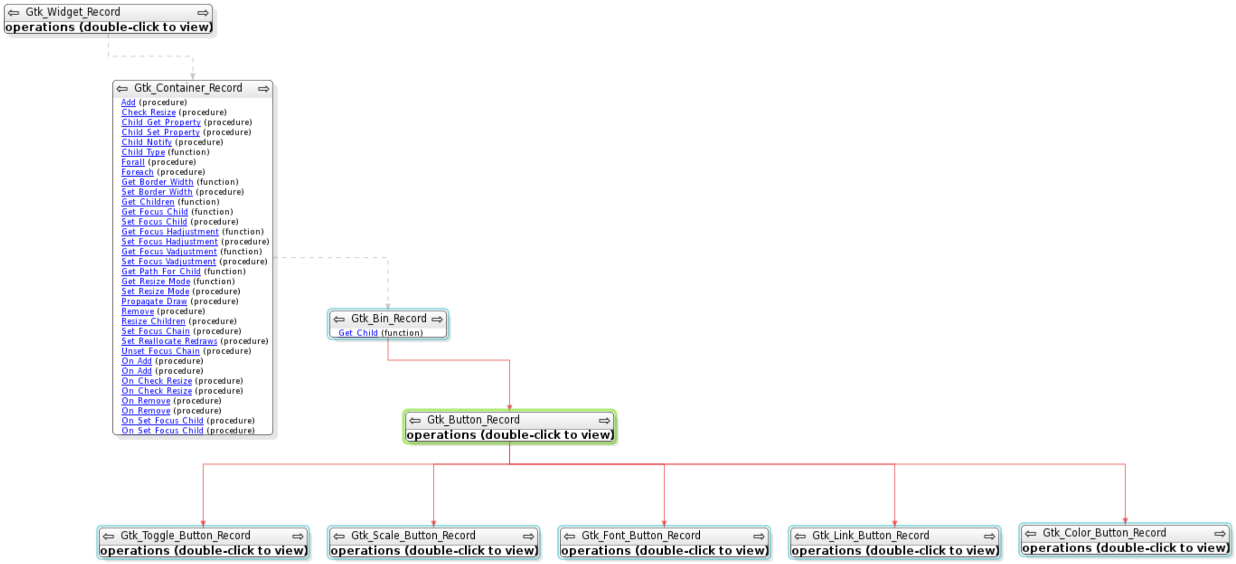


We have smarter layout algorithms, and animations to show the transitions when adding nodes or rebalancing the graph.

The Project browser now has a filter which allows searching for a given pattern directly in the browser. This allowed us to remove the corresponding scope entry in the Search dialog.

The Entity browser now has support for folding compartments.

And the contents of browsers is now saved accross GPS sessions!


⇦ Gtk_Widget_Record ⇨
operations (double-click to view)
⇦ Gtk_Container_Record ⇨
Add (procedure)
Check Resize (procedure)
Child Get Property (procedure)
Child Set Property (procedure)
Child Notify (procedure)
Child Type (function)
Forall (procedure)
Foreach (procedure)
Get Border Width (function)
Set Border Width (procedure)
Get Children (function)
Get Focus Child (function)
Set Focus Child (procedure)
Get Focus Hadjustment (function)
Set Focus Hadjustment (procedure)
Get Focus Vadjustment (function)
Set Focus Vadjustment (procedure)
Get Path For Child (function)
Get Resize Mode (function)
Set Resize Mode (procedure)
Propagate Draw (procedure)
Remove (procedure)
Resize Children (procedure)
Set Focus Chain (procedure)
Set Reallocate Redraws (procedure)
Unset Focus Chain (procedure)
On Add (procedure)
On Add (procedure)
On Check Resize (procedure)
On Check Resize (procedure)
On Remove (procedure)
On Remove (procedure)
On Set Focus Child (procedure)
On Set Focus Child (procedure)
⇦ Gtk_Bin_Record ⇨
Get Child (function)
⇦ Gtk_Button_Record ⇨
operations (double-click to view)
⇦ Gtk_Toggle_Button_Record ⇨
operations (double-click to view)
⇦ Gtk_Scale_Button_Record ⇨
operations (double-click to view)
⇦ Gtk_Font_Button_Record ⇨
operations (double-click to view)
⇦ Gtk_Link_Button_Record ⇨
operations (double-click to view)
⇦ Gtk_Color_Button_Record ⇨
operations (double-click to view)

CHAPTER SEVEN

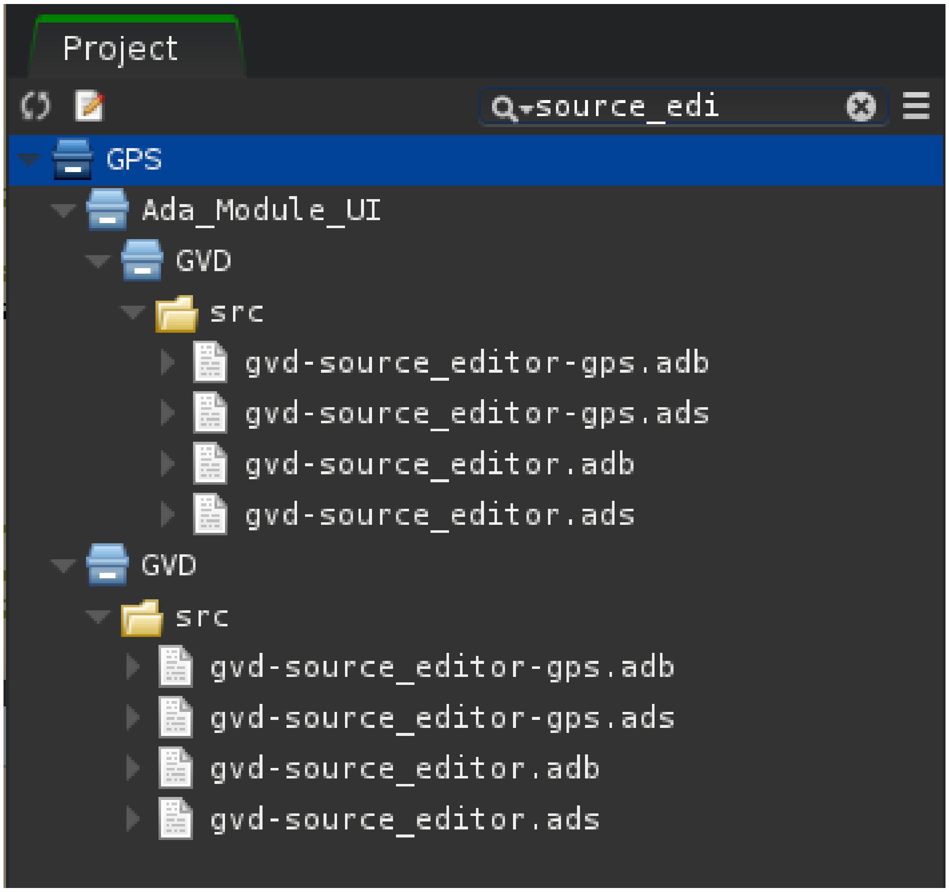
VIEWS

Several small enhancements to views:

- The Outline view for Ada can now show the “withs”.
- There are buttons to move to the previous/next entry in call trees.
- The Messages view can now highlight custom patterns.
- In the Locations view, you can now remove entries for an individual file.

We have also removed completely the GPS shell console, which has been rendered obsolete by the Python console.

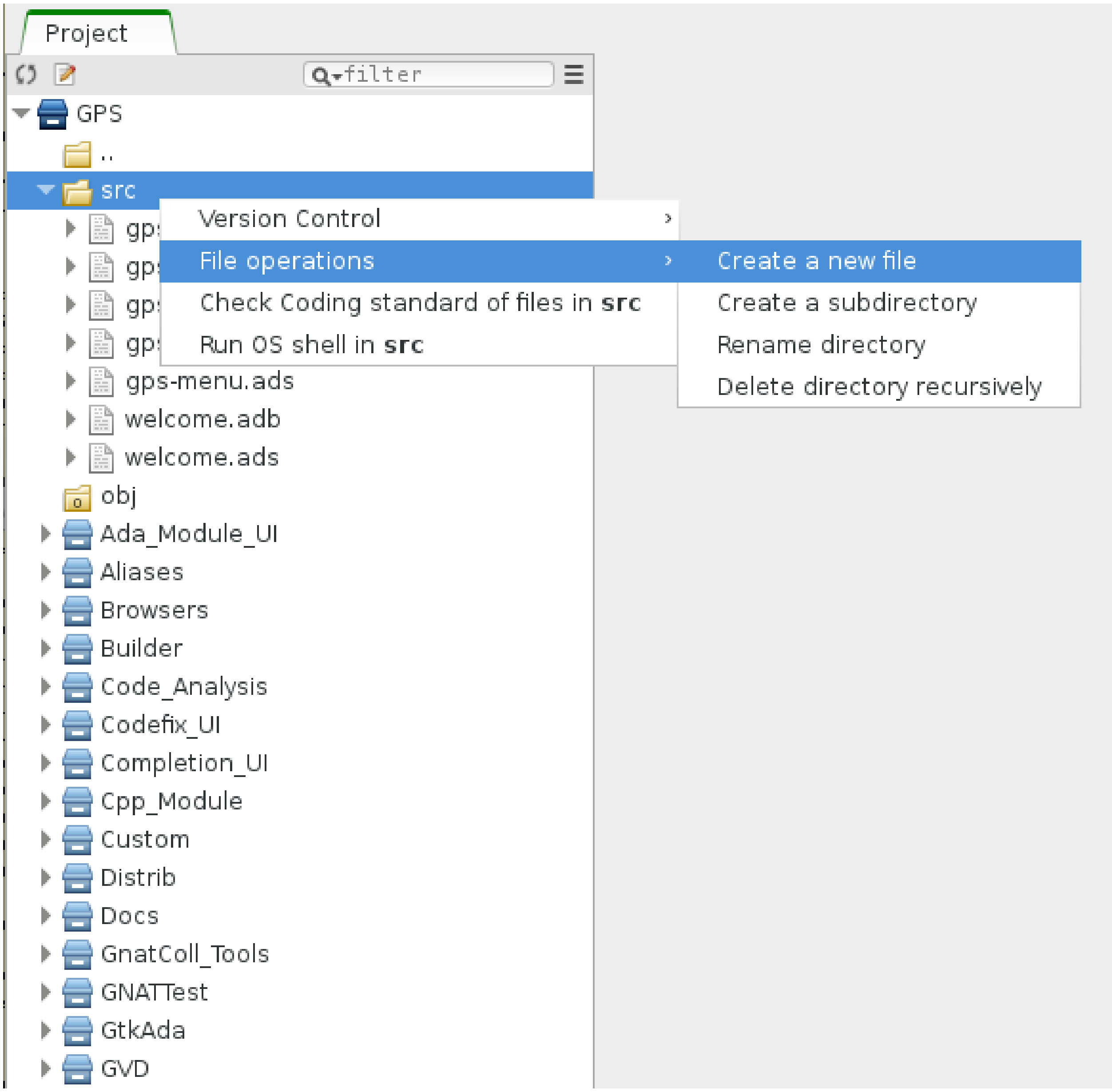
7.1 Project View



There are been a lot of enhancements to the Project view.

- There is now a filter directly in the Project view. This allowed us to remove yet another scope entry from the Search dialog.
- Several new options in the local menu:
  - you can filter out the empty directories
  - you can show files directly under the project, rather than grouped by directories
  - you can list the Ada runtime in the Project view
- Some other enhancements:
  - you can open an OS shell from a directory in the Project view
  - the icon for the root project now has its own color, which is useful to distinguish it when showing the flat view.

Due to popular demand, we have added back a menu for File operations in the project view.


Project
filter
GPS
..
src
gps
gps
gps
gps
gps-menu.ads
welcome.adb
welcome.ads
obj
Ada_Module_UI
Aliases
Browsers
Builder
Code_Analysis
Codefix_UI
Completion_UI
Cpp_Module
Custom
Distrib
Docs
GnatColl_Tools
GNATTest
GtkAda
GVD
Version Control
File operations
Check Coding standard of files in src
Run OS shell in src
Create a new file
Create a subdirectory
Rename directory
Delete directory recursively

CHAPTER
EIGHT

PROJECTS

GPS now supports aggregate projects.

The new Project attributes Target and Runtime are also supported. This is now the recommendation for handling cross projects.

The project wizard has support for gnatname, to create a project from a hierarchy of sources that have a non-GNAT naming convention.

The settings of the scenario for a given project are saved accross GPS sessions.

CHAPTER

NINE

EXTENSIBILITY / CUSTOMIZABILITY

There is support for per-project customization: when loading a project, GPS will load project-specific plugin named <project>.ide.py if it exists in the same directory as the project.

The command GPS.BuildTarget.execute now accepts an extra parameter on_exit: a function which is called when the build target terminates.

The GPS menus are now entirely described in an XML file (menus.xml) which allows you to control the layout of menus in a given GPS install.

We have added support for defining completion resolvers entirely in Python.

There is also support for defining workflows: a sequence of asynchronous actions described as a single Python coroutine - this eases the programming of complex sequences.

CHAPTER

TEN

CROSS-REFERENCES ENGINE

We have several fixes and enhancements to the engine for cross-references: a fix for growing database files, support of Ada separates, better fallbacks when the code is not in compiled state.

A new project attribute allows controlling the location of the databases, so it can be placed on local drives rather than networked drives, for instance.

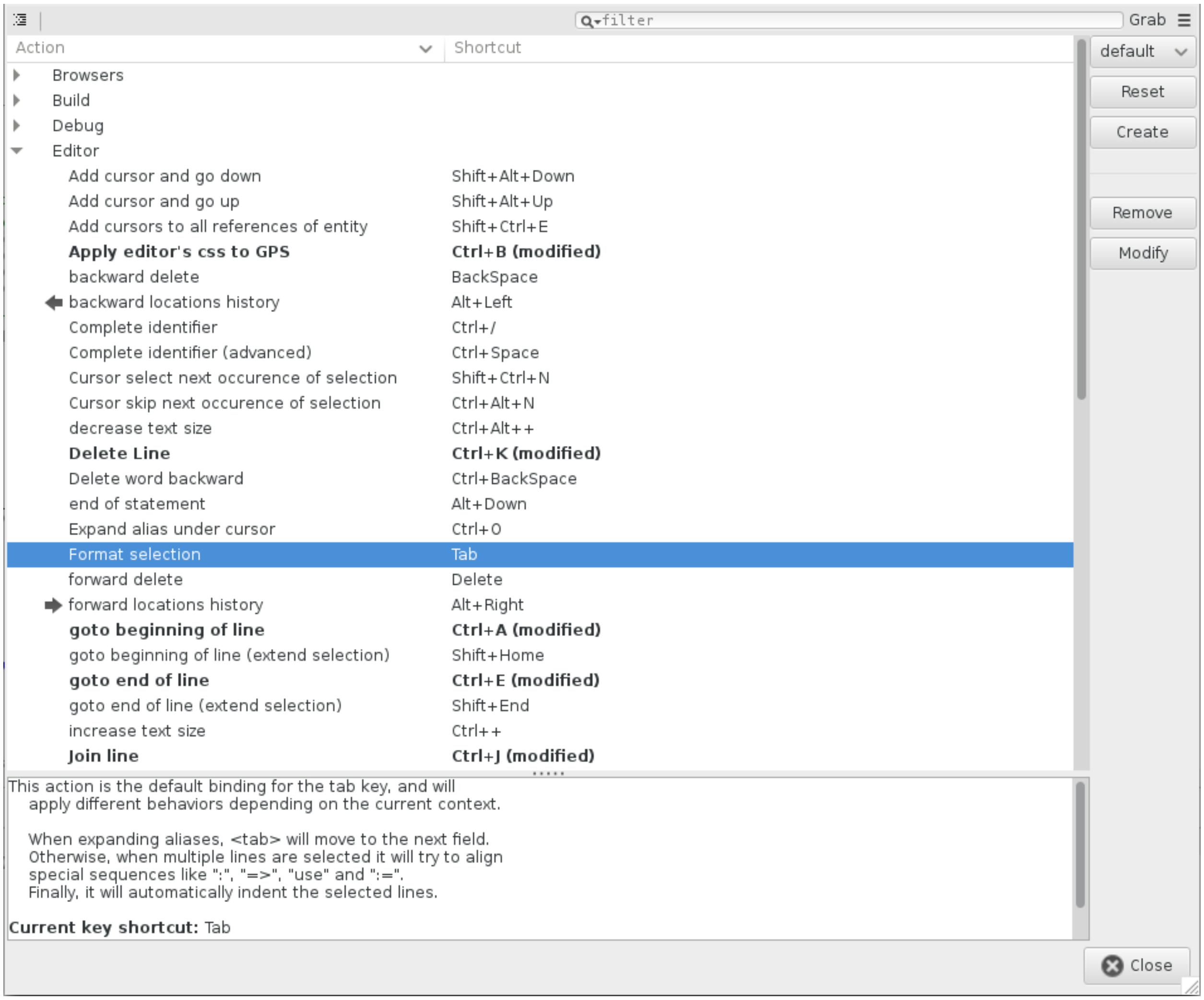
GPS also automatically cleans up the databases that it creates when creating a default project - for instance when using GPS to edit a single file.

CHAPTER

ELEVEN

KEY SHORTCUTS

The key shortcut dialog can now be embedded in the main GPS window, and kept open. Changes are automatically changed (the save button was removed). A filter has been added to make it easy to find the proper action or shortcut. User-defined shortcuts are displayed in bold. The icons associated with actions are displayed, for consistency with the rest of GPS. Menus are no longer displayed in the dialog, since it is better to associate the shortcuts with the corresponding action instead. Multiple key themes are provided and can dynamically be switched (the emacs.py plugin was removed and replaced with an Emacs key theme instead)



Action	Shortcut
Browsers	
Build	
Debug	
Editor	
Add cursor and go down	Shift+Alt+Down
Add cursor and go up	Shift+Alt+Up
Add cursors to all references of entity	Shift+Ctrl+E
Apply editor's css to GPS	**Ctrl+B (modified)**
backward delete	BackSpace
backward locations history	Alt+Left
Complete identifier	Ctrl+/
Complete identifier (advanced)	Ctrl+Space
Cursor select next occurence of selection	Shift+Ctrl+N
Cursor skip next occurence of selection	Ctrl+Alt+N
decrease text size	Ctrl+Alt++
Delete Line	**Ctrl+K (modified)**
Delete word backward	Ctrl+BackSpace
end of statement	Alt+Down
Expand alias under cursor	Ctrl+O
Format selection	Tab
forward delete	Delete
forward locations history	Alt+Right
goto beginning of line	**Ctrl+A (modified)**
goto beginning of line (extend selection)	Shift+Home
goto end of line	**Ctrl+E (modified)**
goto end of line (extend selection)	Shift+End
increase text size	Ctrl++
Join line	**Ctrl+J (modified)**

This action is the default binding for the tab key, and will apply different behaviors depending on the current context.

When expanding aliases, <tab> will move to the next field. Otherwise, when multiple lines are selected it will try to align special sequences like ":", "=>", "use" and ":=". Finally, it will automatically indent the selected lines.

Current key shortcut: Tab

GPS uses the primary modifier key instead of the control key in several places: this means that, under Mac OS, GPS uses the Command key for many actions, to better match the system settings.

The new action “Edit project source file” provides fastest way to open the source of the current project in a GPS editor.

CHAPTER

TWELVE

MISCELLANEOUS UI IMPROVEMENTS

We have removed the “busy” mouse cursor - this is replaced by the background activity indicator in the main toolbar.

The width of the omni-search field can be controlled, which can be useful in projects with very long file names for instance.

The ClearCase integration now supports “diff against working”.

There were minor enhancements to the tip-of-the-day.

You can now apply all auto-fixes to the current file only.

There are default key shortcut (alt+arrows) bound to the next/previous location.

The GNATcoverage detailed messages can be viewed in the editor.

There is an auto-refresh mode for the debugger Memory view.

CHAPTER

THIRTEEN

CODEPEER

GPS now displays the check kinds for CodePeer precondition messages - this means you can filter out messages based on these check kinds.

The CodePeer message review dialog now prevents changing the message review status when a message was reviewed in the source code with pragma Annotate.

All CodePeer messages (SCIL compilation errors, warnings and checks, race conditions) are displayed under one category in the Locations view.

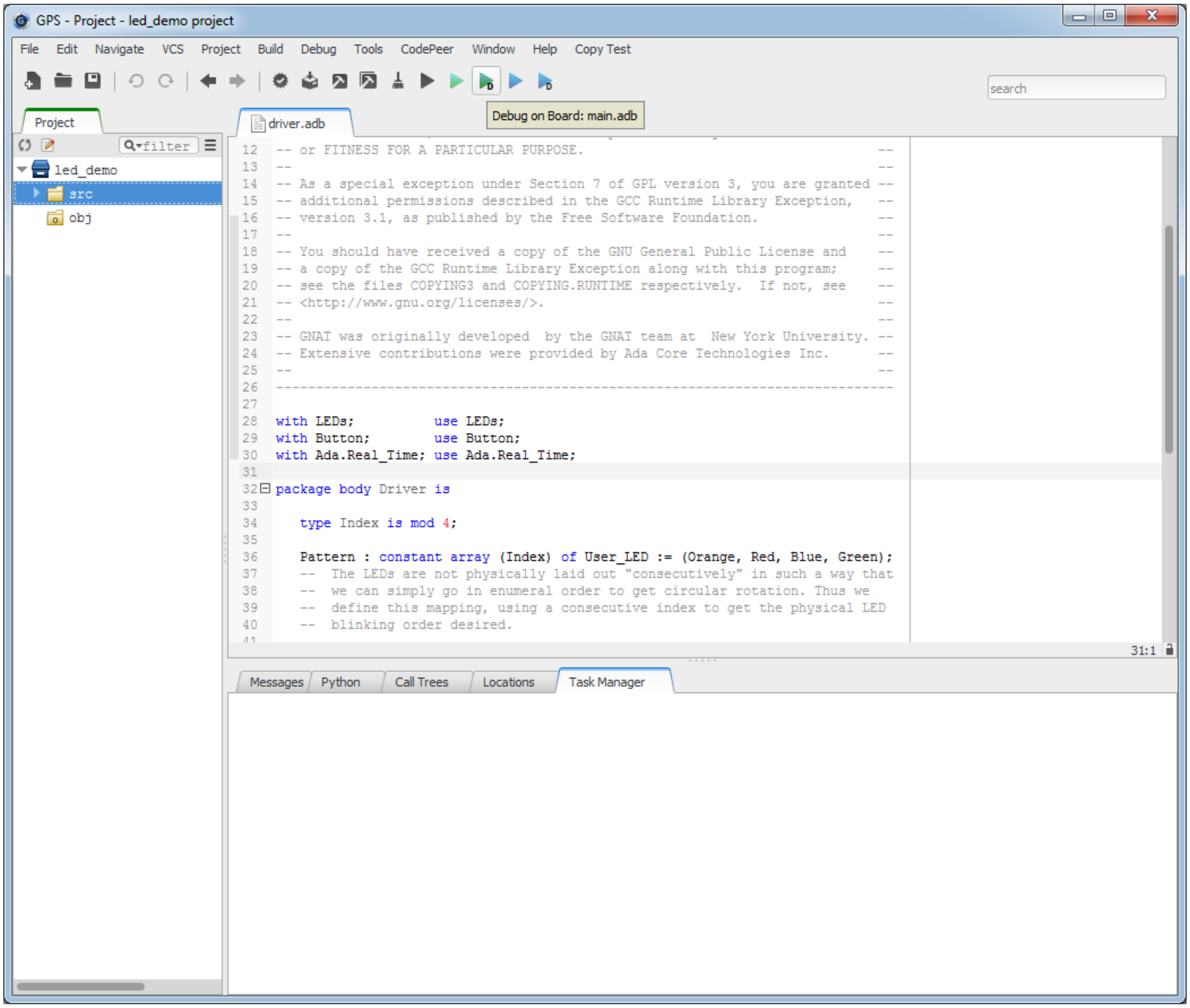
CHAPTER FOURTEEN

BAREBOARD SUPPORT

There is a new Project template to start a demo project for the STM32F4 board.

We have also added four workflows to execute complete sequences of actions at the press of a single button:

- a workflow to build the program, flash it on the board, and run it
- a workflow to build the program, flash it on the board, and launch a debugger connected to it
- a workflow to build the program and run it in the emulator
- a workflow to build the program and debug it in the emulator



```
-- or FITNESS FOR A PARTICULAR PURPOSE.                                     --
--                                                                          --
-- As a special exception under Section 7 of GPL version 3, you are granted --
-- additional permissions described in the GCC Runtime Library Exception,   --
-- version 3.1, as published by the Free Software Foundation.               --
--                                                                          --
-- You should have received a copy of the GNU General Public License and    --
-- a copy of the GCC Runtime Library Exception along with this program;     --
-- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
-- <http://www.gnu.org/licenses/>.                                          --
--                                                                          --
-- GNAT was originally developed  by the GNAT team at  New York University. --
-- Extensive contributions were provided by Ada Core Technologies Inc.      --
--                                                                          --
------------------------------------------------------------------------------

with LEDs;          use LEDs;
with Button;        use Button;
with Ada.Real_Time; use Ada.Real_Time;

package body Driver is

   type Index is mod 4;

   Pattern : constant array (Index) of User_LED := (Orange, Red, Blue, Green);
   --  The LEDs are not physically laid out "consecutively" in such a way that
   --  we can simply go in enumeral order to get circular rotation. Thus we
   --  define this mapping, using a consecutive index to get the physical LED
   --  blinking order desired.
```