GNAT Programming Studio Tutorial

Version 4.3.0
Document revision level $Revision: 119731 $
Date: $Date: 2007-11-28 15:22:16 +0100 (Wed, 28 Nov 2007) $

AdaCore

Table of Contents

1 Introduction

This document provides a guide through the major capabilities of the GNAT Programming Studio by working on a code example: sdc, a simple desktop calculator.

It is important to realize that the features that you are about to experiment with are available on multiple platforms, using the same user interface and capabilities, providing a user-friendly environment with a tight integration between the tools.

Start GPS in the directory containing the tutorial files, or if the directory is read-only, copy the ‘`tutorial`’ directory and its subdirectories in a local (writable) area, and start GPS from the ‘`tutorial`’ directory, so that GPS will load the right context.

By default, the tutorial sources can be found under `<prefix>/share/examples/gps/tutorial`, where `<prefix>` is the prefix directory of the GPS installation.

Alternatively, if you have already started GPS in another directory, you can load the project `sdc.gpr` by using the menu `Project->Open...`

2 Quick overview of the GPS areas

Having launched GPS, you should now have access to a main window composed of several areas:

- a menu bar at the top
- a tool bar under the menu bar
- a scenario view under the tool bar, on the left side
- a project view under the scenario view, on the left side
- a working area on the right of the project view
- a messages window under the working area

3 Editing sources

In the project view, open the `common` directory by clicking on the `[+]` sign (a triangle under unix systems) on the left of `common`. This will open the directory and display a list of source files located in this directory.

Now, double click on '`sdc.adb`': this will open a source editor on this file. The source code is syntax-highlighted: keywords, comments, strings and characters have different colors.

As with many other properties, colors are configurable in GPS:

Select the menu `Edit->Preferences`. This will open a preferences dialog window.

Select the `Editor->Fonts & Colors` page by clicking on the cross next to the item `Editor` and then selecting the `Fonts & Colors` item.

As you go over the various lines and labels, you will notice that by holding the mouse over a label, a tool tip pops up displaying on-line help about the selected item.

Change the background color of the `Keywords` by clicking on the last down arrow, at the right of the `Keywords` line.

Choose a color, e.g a light green. When you're done with the color selection, simply click again on the arrow.

Click on the `Apply` button and look at the effects in the source editor. If you like the new display, click on `OK` to confirm the changes, otherwise clicking on `Cancel` will revert to the previous color.

4 Building applications

Select the menu `Build->Make->sdc.adb`: this will launch a complete build of the *sdc* application. Note also that a key binding is associated with this menu item (⟨F4⟩).

The build has generated a number of errors in a new window: the *Locations* tree, displayed in the bottom area. The errors are also highlighted in the corresponding source editor.

GPS has automatically jumped to the first error message (*sdc.adb, 28:6 : (style) bad indentation*), at the line (28) and column (6) of the error.

Fix the error by hand by inserting a space.

Now you can fix the next error by moving the cursor to the line 30 (press the ⟨down⟩ arrow twice), and by using ⟨Ctrl-Tab⟩ (press first the ⟨Control⟩ key, and then the ⟨Tab⟩ key on your keyboard): this key combination asks the source editor to automatically re-indent the current line.

Note that on some window managers or desktop environments, ⟨Ctrl-Tab⟩ is already defined. If this is the case, you can change this shortcut from the key shortcuts dialog (menu `Edit->Key shortcuts`, `Source editor` section, `Format selection` item).

You can then fix all the remaining errors by selecting the whole block (from line 28 to line 40) and pressing ⟨Ctrl-Tab⟩. To select a block, you can either click on the left mouse button and select the area while holding the button, or using the keyboard by pressing the ⟨Shift⟩ key and moving the cursor using the ⟨Up⟩ or ⟨Down⟩ keys.

Press the ⟨F4⟩ key to build again. GPS will automatically save the modified files, and start a build. This behavior (automatic saving of files before building) can be configured in the preferences dialog.

If you look at the bottom right of the GPS window, you will notice that a progress bar has appeared, displaying the current number of files compiled, and the number of remaining files. This progress bar disappears when the build is finished.

This should now report a successful build.

5 Source Navigation

Now let's try to understand a little bit about how the program is working by looking at the '`sdc.adb`' editor: there's a loop, the main processing is done by the functions Process and Next (at line 30).

Click around line 30, move the mouse over `Process` and let a tool tip appear (`Tokens.Process global procedure declared at tokens.ads:19`): this gives information about the kind of entity and the location (file and line) of the declaration of this procedure, the profile of the parameters, and documentation for this function, as extracted from the comments surrounding the procedure declaration.

Do the same for `Next` (`Tokens.Next global function declared at tokens.ads:15`).

Keeping the mouse over `Next`, display the contextual menu by clicking on the right mouse button, then click on `Goto declaration of Next`: we're now in the package `Tokens`, in file '`tokens.ads`'; but where is this file in the project?

6 Search Dialog

Select the menu `Navigate->Find or Replace...`: this will open a search dialog. In the `Search for:` text entry type '`tokens.ads`'. Then select `Project view` in the `Look in:` area. The search area provides an easy way to search for text or regular expressions in several contexts including the current file, the project view, ...

Now click on `Find`. The file '`tokens.ads`', in directory `struct` is highlighted.

Close the search dialog by clicking on the `Close` button.

Note that in this specific case, a simpler way to locate a file in the project view is to use the contextual menu from the source editor: `Locate in Project View: tokens.ads`.

7 Project View (entities)

Click on the `[+]` sign (or triangle) to open ‘`tokens.ads`’ entities. When you click on a file in the project view, you get language sensitive information about the file, such as `packages`, `subprograms`, `tasks`, ... for `Ada`.

Open the `subprogram` category, click on `Process`: this will open ‘`tokens.ads`’ and move the cursor on the first line corresponding to the procedure `Process`.

Similarly, click on `Next`, and move your mouse on `Next` in the source editor.

8 Back to Source Navigation

Using the contextual menu, select `Goto body of Next`; scroll through the procedure `Next`, move the mouse on `Instructions.Read` at line 46 and from the contextual menu, select `Goto body of Read`.

We've now navigated quite a bit through the application source code, which you can verify by clicking on the left arrow in the tool bar, to go back to the previous locations visited.

Repeat the operation until you're back in '`sdc.adb`'. As with the undo/redo capability in the source editor, the `goto previous/next location` is infinite.

9 Code Completion

Go on the line 38 of sdc.adb. You can see that there is a null instruction for the case of Stack.Overflow. We are going to add some code here, using the code assist capabilities.

Type ⟨enter⟩ to create a new line, and then `Scr`, and hit ⟨Ctrl+Space⟩. If you've never used auto-completion before, GPS will pop a window asking you if you want to enable the auto-completion engine. Answer `"yes"`. You will see at the bottom right of GPS a progress bar, showing the status of the completion database load. Note that you can still work during this operation. The auto completion mechanism can be disabled from the preferences if needed.

Wait until the progress bar disappears, and then hit ⟨Ctrl+Space⟩ again. A completion popup will be displayed, showing all the entities of the project begining with `Scr`. Select `Screen_Output`. The code will be automatically completed in the editor. Then add a dot in your code. The completion popup will be automatically triggered, and will offer you to complete your code with the entities contained in the `Screen_Output` package. Select `Msg`, add a space, and then an open parenthesis. Once again, the completion windows will pop up, and show you the possible parameters for msg. If you choose the first entry of the completion list (`"`params of Msg`"`), the call will be automatically completed by a list of named parameters. Complete the list by giving e.g. `"The stack is full."` for `S1`, `""` for `S2`, and `True` for `End_Line`.

Don't forget to add a semicolon at the end of the instruction. Then hit ⟨F4⟩ in order to rebuild the application.

10 Run

It is now time to run the application: select the menu `Build->Run->sdc`, which will open a dialog window. Type `input.txt` in the text entry: this is the name of a text file that will be passed as argument to the *sdc* program.

Now click on `OK`: a new window titled `Run: sdc input.txt` is created at the bottom of the main window where the sdc application runs and displays an unexpected internal error: this is a good opportunity to use the integrated debugger.

Close the execution window by clicking on the x icon on the top right corner of this window.

11 Debug

Open the preferences dialog (menu `Edit->Preferences`) and click on the `Debugger` item on the left; set the button `Break on exceptions` to *Enabled*: this will enable by default a special breakpoint every time an exception is raised. Click on `OK` to confirm your change.

Now select the menu `Debug->Initialize->sdc`: a new window is created: this is the debugger console. You can also look at the various debug menu item and tool bar buttons which are now activated.

Use the menu `Debug->Data->Call Stack`: this opens a new window on the right of the source editors. If you select the contextual menu in the call stack, various pieces of information can be displayed or removed in the call stack. From this contextual menu, add the `Frame Number` info by clicking on it.

Now select the menu `Debug->Run...`. Notice that `input.txt` has been filled automatically for you since the two menus `Build->Run...` and `Debug->Run...` are synchronized. Click on `OK`: the debugger should stop on an exception (`Constraint_Error` in the file '`stack.adb`', at line 49).

Go up in the call stack by clicking on the `tokens.process` line (frame number 6 or 7, depending on your GNAT version).

If you move the mouse over the parameter `T` at line 64, a tool tip is displayed showing the value of `T`. You have probably noticed that tool tips, like menus, are contextual: depending on the current session and on the entity selected, different information is displayed.

Select the contextual menu `Debug->Display T`: this will open a new window: the data window, with a box displaying graphically the contents of the different fields of `T`, each clearly separated.

On `T` data display, select the contextual menu `Display->Show Value + Type`: this displays for all fields both their type and value.

Special colors are used in the data display: blue for pointers that can be dereferenced by a double-click (double click on `T.val`); red for fields that have been modified since last step.

In the contextual menu that pops up when you right-click on `T`, select `Debug->View memory at address of T`: a memory view is opened. Use the `up` and `down` arrows on the right to visit memory.

Click in the memory dump, and modify it by typing numbers. Notice the red color for modified values; click on `Undo Changes` to cancel the modifications; click on `Close` to close the memory window.

In the call stack, go back to `stack.push` frame (num 4 or 5). Move the mouse over `Last` and let the debugger display its value: 0. From the contextual menu, select `Goto declaration of Last`: this will jump to the line 16 of '`stack.adb`', where you can see that `Last` is a `Natural`. Now click on the `Goto Previous`

`Location` button in the tool bar: we're now back at line 49 where we can see that for a `Push` procedure, `Last` should be incremented, and not decremented.

Fix the line to `Last := Last + 1;`

Save the file (⟨Ctrl-S⟩); End the debug session: menu `Debug->Terminate`; Rebuild (press ⟨F4⟩ key); Rerun (menu `Build->Run->sdc`): the program now completes as expected. Close the execution window.

12 Call Graph

Now go back to the file '`sdc.adb`', move the mouse over the procedure *sdc* at line 8, select the contextual menu `Browsers->Sdc calls`: this will open a new window titled *Call graph browser*.

Note that there is also a top level contextual menu (`Sdc calls`) which provides a tree view of the callers/callees.

In the call graph, click on the right arrow of `Process` (one of the first items on the top). Also click on the right arrow of `error_msg`.

Select `Orthogonal links` in the contextual menu of the graph to change the way links are displayed in the graph. You may then play with the zoom (⟨=⟩ and ⟨-⟩ keys).

If you select `Hide links` from `error_msg` contextual menu, this will hide all the links that are related to this item: the link between the callers and callees of `error_msg` are no longer displayed. This can be useful when the graph becomes complex, to hide some parts. If you go back to the contextual menu, you can now select `Show links` to show the links again.

Click on right arrow of `process` (`(Decl) instructions.ads:12`).

The items can also be moved: move e.g `msg` item around.

You can also recompute the layout of all the current items by using the contextual menu `Refresh layout`.

Click on left arrow of `msg` to display who is calling `msg`. Notice that `view` calls `msg`.

Click on left arrow of `view`: the arrow disappears, and no new items are created, which means that `view` isn't called by anyone, so we're now going to remove this procedure.

13 Locations Tree

From *view*, click on the blue link: `stack.ads:32`, this will open the file '`stack.ads`' at line 32. Then from the source editor (file '`stack.ads`'), select the contextual menu `References->Find all references to View`: this highlights the `Locations` tree which now contains all the references for `view`, grouped by files ('`stack.ads`' and '`stack.adb`').

The first location is highlighted automatically: this is the spec of the procedure `View`. Now click in the tree on the `+` sign (or triangle) at the left of '`stack.adb`': two locations are listed, at line 90 and 97. Click on each of these locations: they correspond to the procedure body.

The `Find all references` capability is another way to list all the uses of an entity, and it confirms that `View` isn't called in our project.

Remove *View* body by e.g selecting it, and pressing the ⟨Delete⟩ key, then save the file (⟨Ctrl-S⟩).

Do the same for the spec, save the file.

Close the '`stack.ads`' and '`stack.adb`' files (menu File->Close, or using the shortcut ⟨Ctrl-W⟩), as well as the call graph window. Rebuild by pressing the ⟨F4⟩ key.

Let's now see how to create a project corresponding to the *sdc* project we've used in this tutorial.

14 Projects

14.1 Project Wizard

Go to the menu `Project->New...`: this is a standard wizard, with various steps listed on the left area of the window.

The first page of the wizard allows you to select what kind of project you want to build, depending on the information you have. Select the default choice `Single Project`, and press `Forward`.

Type *sdc2* in the project name field.

Click on `Forward`: we are now on the language selection page. It is possible to create a multi-language project by e.g. selecting the C or C++ check box.

Click on `Forward`: we are now on the `VCS page`. *VCS* stands for *Version Control System*. GPS provides a generic framework for *VCS* which allows it to support new systems easily. Systems supported by default are CVS, ClearCase and Subversion. Select `CVS`.

Click on `Forward`: this is the source directories selection, used to specify the project's sources. Click on the `Add` button, and select the `struct` directory, then click on `OK` to validate.

Click on `Forward`: this is the `Build` and `Exec` directory selection, used to store object, ali files, ...

Click on the first `Browse` button, then click on `obj`, and finally click on `OK`.

Click on `Forward`: this is the main units selection, used mainly for building executables and debugging.

Click on `Add`, open the `common` directory and select `sdc.adb`.

Click on `Forward`: this is the naming scheme editor. GNAT is very flexible and can use any kind of naming scheme for Ada files. In particular, you can easily set the default file extensions (e.g by using one of the predefined schemes) and you can also specify exceptions that use non standard file names.

Click on `Forward`: we're now in the switch selector. Select `Recompile if switches changed`.

Click on `Ada` page.

Select `Full errors` and `Overflow checking`. The boxes and the command line (the text entry at the bottom of the page) are fully synchronized, e.g if you click on the command line, and change `-gnatf` to `-gnat`, the `Full errors` check box is unselected; now type `a` to get `-gnata`, and notice that `Enable assertions` is now selected.

We've now created a project similar to the one used in this tutorial.

Click on `Cancel` to close the wizard.

Clicking on `Apply` instead would have created the project file and loaded it in GPS.

14.2 Project properties

In the project view, on the project *sdc*, use the contextual menu `Project->Properties`. All the properties set in the project wizard can be found here as well. You can switch between pages by clicking on the tabs located along the left side of the window.

Once you're done exploring the property pages, click on the `Cancel` button to close the properties window.

14.3 Variable editor

Select the window titled "Scenario View". If not available, you can open it using the menu `Tools->Views->Scenario`. This window contains a `Build` label.

This is a configuration variable. With GPS and the GNAT project facility, you can define as many configuration variables as you want, and modify any project settings (e.g. switches, sources, ...) based on the values of configuration variables. These variables can also take any number of different values.

The `Build` variable demonstrates a typical `Debug/Production` configuration where we've set different switches for the two modes.

Click on the button at the left (`Edit variable properties`): this is the variable editor, where values can be added or renamed. Close the variable editor by clicking on the `Cancel` button.

Now, let's take a look at the switches set in the project.

14.4 Switch editor

Select the menu item `Project->Edit File Switches`: a global switch editor is displayed in the working area, showing the switches associated with each file in the `sdc` project.

The editor lists the switches associated with each file in the project. Gray entries indicate default (global) switches. Notice that '`screen_output.adb`' has specific switches, which are highlighted using a different font.

Switch between `Debug` and `Production` mode in the `Build` combo box: the switches are updated automatically.

Back to our project, let's now examine the dependencies between sources.

14.5 Source dependencies

Select '`sdc.adb`' in the `Project View` and then the contextual menu item `Show dependencies for sdc.adb`: this will open a new graph showing the dependencies between sources of the project.

Click on the right arrow of '`tokens.ads`' to display the files that '`tokens.ads`' depends on. Similarly, click on the right arrow of '`stack.ads`'.

14.6 Project dependencies

Back in the project view, on the *Sdc* project, select the contextual menu `Project->Dependencies`, then on the `Add From File`, then open the *tutorial* directory and click on the `projects` subdirectory. Select the file `prj1.gpr`. Click on `Apply` to validate the change.

You can see the new dependency added in the project view, as a tree of projects. In particular, project dependencies are duplicated: if you open the `prj1` icon by clicking on the `[+]` sign (or triangle), and then similarly open the `prj2` icon, you will notice that the project `prj4` is displayed twice: once as a dependency of `prj2`, and once as a dependency of `prj1`.

GPS can also display the graph of dependencies between projects: on *Sdc* project, use the contextual menu `Show projects imported by Sdc`: this will open a project hierarchy browser.

On the `Sdc.gpr` project, select the contextual menu `Show projects imported by Sdc recursively`.

In the browser, you can move the project items, and select them to highlight the dependencies.

Close the project browser by clicking on the `[x]` sign at the top right area of the window, or by typing ⟨Ctrl-W⟩, or by using the menu `File->Close`.

15 Epilogue

This terminates our tour of GPS, the GNAT Programming Studio. We hope this tutorial gave you a good overview of the general capabilities available with GPS. A non exhaustive list of the features not mentioned in this document includes:

- Documentation generation
- Automatic generation of body files
- Pretty printing
- Visual comparison of files
- Version control
- Flexible multiple document interface
- Code coverage

For more information, please look at the *User's Guide* (`gps.html`), and also look at the `Tools` menu which gives access to most of these capabilities.