GPR2 Library User Manual

Release 27.0w

AdaCore

Apr 28, 2026

CONTENTS:

Version 27.0w
Date: Apr 28, 2026

CHAPTER

ONE

INTRODUCTION

The GPR2 library provides an extensive interface to access project files (`.gpr` files).

Project files are first loaded in a project tree object.

Load functions have parameters to configure load options as it is usually done using GPRbuild on the command line with corresponding switches.

GPR project files are loaded in a project tree using `GPR2.Project.Tree.Load_Autoconf` or `GPR2.Project.Tree.Load` functions. Use the latter if you want to use a config project file (`.cgpr`)

If your application wants to support some GPR tools standard switches (such as `-P`, `-X`, `--target`, `--RTS`, etc), a `GPR2.Options` package is provided to simplify and normalize GPR standard switches handling.

Once loaded, the root project and all imported, aggregated, extended projects including the configuration and runtime projects can be accessed.

For each projects, all variables types, variables, attributes, packages can be evaluated, taking into account load configuration, default values and aliases.

Projects sources (`.ads`, `.adb` or naming conventions defined in project files) and units (package specification, body and subunits) can be analyzed as well. Units dependencies are also handled.

CHAPTER

TWO

CONCEPTS

The GPR2 library uses different types of objects to access project files data.

Types used when loading a project into a `GPR2.Project.Tree.Object`

- `GPR2.Path_Name.Object`

  Handle filesystem paths. `GPR2.Path_Name` package contains conversions from and to `GNATCOLL.VFS.Virtual_File` type.

- `GPR2.Path_Name.Set.Object`

  Handle a set of paths.

- `GPR2.Project.Tree.Object`

  Handle a project tree. Project file used to load project tree is available as `Tree.Root_Project`.

- `GPR2.Project.Tree.View_Builder.Object`

  Used to load a project as a project file path alternative. Attributes (for example `Source_Dirs`) can be defined for this object before it is used to load a project tree.

- `GPR2.Context.Object`

  Context objects are used to pass scenario variables to a tree. A context is provided when loading a tree. It can then be changed calling `GPR2.Project.Tree.Set_Context`.

- `GPR2.Environment.Object`

  Environment object provided at load time is used by GPR2 to access the environment when needed.

- `GPR2.Log.Object`

  Used to store the log messages (error/warning/information) coming from the parser.

- `GPR2.Message.Object`

  Base object to report information at the end user level.

Types used to handle loaded gpr files. A loaded gpr project is stored in a tree containing views. The tree contains root, imported, extended, aggregated, configuration and runtime views.

- `GPR2.Project.View.Object`

  Handle a gpr project file. Used to handle root project and any referenced (imported, extended, aggregated) project.

- `GPR2.Project.View.Set.Object`

  Handle a set of view objects.

- `GPR2.Project.Configuration.Object`

  Handle the configuration object for a project tree.

Types used to handle types, variables, attributes and packages.

- `GPR2.Project.Typ.Object`

  Handle a variable type definition.

- `GPR2.Project.Typ.Set.Object`

  Handle a set of variable types.

- `GPR2.Project.Variable.Object`

  Handle a variable defined in a project or in a project's package.

- `GPR2.Project.Variable.Set.Object`

  Handle a set of variables

- `GPR2.Project.Attribute.Object`

  Handle project view attributes.

- `GPR2.Project.Attribute.Set.Object`

  Handle a set of attributes.

- `GPR2.Project.Pack.Object`

  Handle a project's package which is a set of attributes and variables.

Types used to handle project sources

- `GPR2.Project.Source.Object`

  Handle a project's source file.

- `GPR2.Project.Source.Set.Object`

  Handle a set of project's source files.

Types used to handle project units

- `GPR2.Project.Unit_Info.Object`

  Handle a view's unit.

- `GPR2.Project.Unit_Info.Set.Object`

  Handle a set of view's units.

Type used to handle standard gpr tools switches.

- `GPR2.Options.Object`

  Handle loading a project using common GPR tools switches (`-aP`, `--autoconf`, `--config`, `--db`, `--db-`, `--implicit-with`, `--no-project`, `-P`, `--relocate-build-tree`, `--root-dir`, `--RTS`, `--src-subdirs`, `-subdirs`, `--target`, `--unchecked-shared-lib-imports`, `-X`).

Memory management of GPR2 objects is easy as all references are using reference counting.

API proper use is ensured using precondition aspect. These checks are activated using `debug` or `release_checks` values for `GPR2_BUILD` scenario variable. Usually object's function calls should be protected by a corresponding `Has_XXX` or `Is_XXX` check. Some object's function calls are using a `function Check_XXX (Object, out Result) return Boolean;` pattern.

```
function Has_Runtime_Project (Self : Object) return Boolean;
--  Returns True if a runtime project is loaded on this tree

function Runtime_Project (Self : Object) return View.Object
  with Pre => Self.Is_Defined and then Self.Has_Runtime_Project;
--  Returns the runtime project for the given tree

function Is_Extended (Self : Object) return Boolean
  with Pre => Self.Is_Defined;
--  Returns True if the view is extended by another project

function Extending (Self : Object) return Object
  with Pre  => Self.Is_Defined and then Self.Is_Extended,
       Post => Extending'Result.Is_Extending;
--  Return the extending view

 function Check_Source
  (Self     : Object;
   Filename : GPR2.Simple_Name;
   Result   : in out Project.Source.Object) return Boolean
  with Pre => Self.Is_Defined;
```

All sets handled by the GPR2 library can be easily iterated. They are returned as iterable objects (`GPR2.Path_Name.Set.Object`, `GPR2.Project.View.Set.Object`, and any `GPR2.<child_name>.Set.Object`) or they define a `Iterate` API.

```
function Iterate
  (Self        : GPR2.Log.Object;
   Information : Boolean := True;
   Warning     : Boolean := True;
   Error       : Boolean := True;
   Lint        : Boolean := False;
   Read        : Boolean := True;
   Unread      : Boolean := True)
   return Log_Iterator.Forward_Iterator'Class;

function Iterate
  (Self   : GPR2.Project.Tree.Object;
   Kind   : Iterator_Control := Default_Iterator;
   Filter : Filter_Control   := Default_Filter;
   Status : Status_Control   := Default_Status)
   return Project_Iterator.Forward_Iterator'Class;

function Iterate
  (Self : GPR2.Unit.List.Object)
   return Unit_Iterator.Forward_Iterator'Class;
```

2.1 GPR2.Path_Name.Object - Files and directories

Files and directories are handled in GPR2 using Object type defined in `GPR2.Path_Name` package.

Objects can be created directly using `Create_File` or `Create_Directory` API, or from an existing `Object` using Compose API.

An API is provided to easily interface these objects with `Filesystem_String` and `Virtual_File` types defined in `GNATCOLL.VFS`.

A complete API is provided to manipulate files and directories.

2.2 GPR2.Project.Configuration.Object - Configuration

Configuration files (`.cgpr` files) generated usually by `gprconfig` and provided using `--config` switch, can be loaded using `GPR2.Project.Configuration.Load` function. These objects are then used at project file tree load time.

2.3 GPR2.Project.Tree.Object - Loaded project tree

Handling a project file using GPR2 starts loading a `GPR2.Project.Tree.Object`. All dependency projects (imported, extended, aggregated), selected Ada runtime and used configuration project will be loaded as well.

`GPR2.Options` package provides a `Load_Project` function.

```
declare
   Options : GPR2.Options.Object;
   Tree    : GPR2.Project.Tree.Object;
   Loaded  : Boolean;
begin
   Options.Add_Switch (GPR2.Options.P, "test");
   Options.Finalize;
   Loaded := Options.Load_Project (Tree);
end;
```

Tree can also be loaded using `Load` and `Load_Autoconf` primitives located in `GPR2.Project.Tree package`.

```
with Ada.Text_IO;  use Ada.Text_IO;

with GPR2.Context;
with GPR2.Log;
with GPR2.Path_Name;
with GPR2.Project.Tree;
with GPR2.Project.Configuration;

procedure Test_Project is
   Project_File   : constant GPR2.Path_Name.Object :=
                       GPR2.Project.Create ("path_to_project.gpr");
   Config_Project : constant GPR2.Path_Name.Object :=
                       GPR2.Project.Create ("project.cgpr");
   Tree           : GPR2.Project.Tree.Object;
begin
   --  Load path_to_project.gpr & create project.cgpr file
```

(continues on next page)

(continued from previous page)

```
   Tree.Load_Autoconf
     (Filename          => Project_File,
      Context           => GPR2.Context.Empty,
      Config_Project    => Config_Project);

   --  Load path_to_project.gpr using a configuration file.
   Tree.Load
     (Filename         => Project_File,
      Context          => GPR2.Context.Empty,
      Config           => GPR2.Project.Configuration.Load (Config_Project),
      Build_Path       => GPR2.Project.Create ("build_path"),
      Subdirs          => "subdirs");

   --  Display object directory taking into account build tree & subdirs
   Put_Line (Tree.Root_Project.Object_Directory.Value);
exception
   when GPR2.Project_Error =>
      GPR2.Log.Output_Messages (Tree.Log_Messages.all);
end Test_Project;
```

Tree can also be loaded from `GPR2.Project.Tree.View_Builder.Object` instead of `GPR2.Path_Name.Object`. This feature is useful when you need to load a Tree but no project file is available.

```
declare
    Root     : GPR2.Project.Tree.View_Builder.Object :=
                  GPR2.Project.Tree.View_Builder.Create
                    (GPR2.Path_Name.Create_Directory ("demo"), "Custom_Project");
    Src_Dirs : GPR2.Containers.Value_List;
    Tree     : GPR2.Project.Tree.Object;

    package PRA renames GPR2.Project.Registry.Attribute;
begin
    Src_Dirs.Append ("src1");
    Src_Dirs.Append ("src2");
    Root.Set_Attribute (PRA.Source_Dirs, Src_Dirs);
    Root.Set_Attribute (PRA.Object_Dir, "obj");

    GPR2.Project.Tree.View_Builder.Load_Autoconf (Tree, Root, GPR2.Context.Empty);
end;
```

2.4 GPR2.Context.Object - Scenario variables

Scenario variables are defined using `GPR2.Context.Object`.

Key/Value are added or update using `Include` primitive.

A loaded Tree can be updated calling `Set_Context` primitive when scenario variables need to be changed.

```
Tree    : GPR2.Project.Tree.Object;
Context : GPR2.Context.Object;
```

(continues on next page)

(continued from previous page)

```
--  Change Context and update Tree

Context.Include ("KEY", "value");
Tree.Set_Context (Context);
```

2.5 GPR2.Log.Object - Messages

GPR2 is reporting project and configuration file messages through `GPR2.Log.Object`.

`GPR2.Log` package provides an configurable iterator to list selected messages. `Output_Messages` primitive is provided to print filtered messages.

A message contains the following properties.

- `Level`, can be `Information`, `Warning`, `Error` or `Lint`.
- `Status`, can be `Read` or `Unread`.
- `Message` text.
- `Sloc`, defining where Filename:Line:Column the message was issued.

2.6 GPR2.Project.View.Object - Project file

Any project file (root, imported, extended, etc…) parsed during `Load` or `Load_Autoconf` execution is reported as `GPR2.Project.View.Object`.

A View object contains attributes, types, variables, sources, units and any extra data defined in project file.

When sources files are added/deleted, `Tree.Invalidate_Sources (View);` should be used to update sources related data. Calling `Tree.Invalidate_Sources;` updates all sources for all views in the Tree.

2.7 GPR2.Project.Attribute.Object - Attributes

View's attributes can be accessed using a name, an index and a position. A name is mandatory.

Name parameter uses `GPR2.Q_Attribute_Id` type. Predefined `Q_Attribute_Id` values can be found in `GPR2.Project.Registry.Attribute` package.

`Q_Attribute_Id` value for Source_Dirs is attribute is `GPR2.Project.Registry.Attribute.Source_Dirs`.

`Q_Attribute_Id` value for Builder'Executable attribute is `GPR2.Project.Registry.Attribute.Builder.Executable`.

New `Q_Attribute_Id` values (for external tools) can be defined/registered as follows:

```
use GPR2;
Tool_Id     : constant GPR2.Package_Id := +"tool";
Attribute_A : constant GPR2.Q_Attribute_Id := (Tool_Id, +"attribute_a");

--  new packages and attributes should be registered during initialization.

GPR2.Project.Registry.Pack.Add (Tool_Id, GPR2.Project.Registry.Pack.Everywhere);
```

(continues on next page)

(continued from previous page)

```
GPR2.Project.Registry.Attribute.Add
  (Name                => Attribute_A,
   Index_Type          => GPR2.Project.Registry.Attribute.No_Index,
   Value               => GPR2.Project.Registry.Attribute.Single,
   Value_Case_Sensitive => False,
   Is_Allowed_In       => GPR2.Project.Registry.Attribute.Everywhere);
```

Registered `Q_Attribute_Id` packages/attributes can be exported/imported using JSON files through `GPR2.Project.Registry.Exchange` package API.

Index are created using `GPR2.Project.Attribute_Index.Create` primitives.

As an example to get Builder'Executable ("mains.adb" at 2) attribute use:

```
Executable : constant GPR2.Project.Attribute.Object :=
               Tree.Root_Project.Attribute
                 (Name   => GPR2.Project.Registry.Attribute.Builder.Executable,
                  Index  => GPR2.Project.Attribute_Index.Create ("mains.adb"),
                  At_Pos => 2);
```

2.8 GPR2.Project.Variable.Object - Variables

Variables defined in a gpr file can be accessed using `Variables` and `Variable` primitives of `GPR2.Project.View.Object`. `Variables` function returns the variables set in `GPR2.Project.Variable.Set.Object`. `Variable` function returns the requested variable.

As usual, requests should be protected by corresponding `Has_XXX` requests. If a variable has a type, its type can be stored in a `GPR2.Project.Typ.Object`.

```
type Build_Type is ("debug", "release", "release_checks", "gnatcov");
Build : Build_Type := external ("GPR2_BUILD", "debug");
```

The following code show how a variable and its type can be accessed.

```
declare
   Name          : constant GPR2.Name_Type := "Build";
   View          : GPR2.Project.View.Object := Tree.Root_Project;
   Variable      : GPR2.Project.Variable.Object;
   Variable_Type : GPR2.Project.Typ.Object;
begin
   if View.Has_Variables (Name) then
      Variable := View.Variable (Name);
      Ada.Text_IO.Put_Line (Variable.Value.Text);
      if Variable.Has_Type then
         Variable_Type := Variable.Typ;
         Ada.Text_IO.Put (String (Variable_Type.Name.Text) & " : ");
         for V of Variable_Type.Values loop
            Ada.Text_IO.Put (V.Text & ",");
         end loop;
         Ada.Text_IO.Put_Line ("");
      end if;
```

(continues on next page)

(continued from previous page)

```
   end if;
end;
```

2.9 GPR2.Project.Source.Object - Sources

Sources of a project file are handled by `GPR2.Project.Source.Object` type.

They can be accessed through `View.Sources` or `View.Source (Path_Name)` functions.

GPR2 parses the source file using libadalang or the corresponding ALI file generated previously by gnat to report contained unit(s) or dependencies list.

2.10 GPR2.Project.Unit_Info.Object - Units

Units of a project file are handled by `GPR2.Project.Unit_Info.Object` type.

They can be accessed through `View.Units` or `View.Unit (Unit_Name)` functions.

Note that the list of units is populated only when `Tree.Update_Sources`, `View_Has_Sources` or `View.Sources` is called.

As a performance optimization, if you don't care about units and source dependencies, don't forget when updating sources to explicitly ask for using no backends. (all backends are used as default)

```
Tree.Update_Sources (Backends => GPR2.Source_Info.No_Backends);
```

2.11 GPR2.Options.Object - GPR tools common switches support

Using this `GPR2.Options.Object`, normalize & simplify GPR tools common switches support (development & maintenance)

The following code show how this object is used.

```
declare
   Options : GPR2.Options.Object;
   Tree    : GPR2.Project.Tree.Object;
   Loaded  : Boolean;
begin
   Options.Add_Switch (GPR2.Options.AP, "added-path");
   Options.Add_Switch (GPR2.Options.P, "test");
   Options.Add_Switch (GPR2.Options.Autoconf, "autoconf.cgpr");
   Options.Add_Switch (GPR2.Options.X, "BUILD=Debug");
   Options.Finalize;
   Loaded := Options.Load_Project (Tree);
end;
```

GPR tools common supported switches are:

- `-aP<dir>` (`GPR2.Options.AP`)

  -aP<dir> or -aP <dir> Add directory dir to project search path

- `--autoconf=<file.cgpr>` (`GPR2.Options.Autoconf`)

  Specify/create the main config project file name

- `--config=<file.cgpr>` (`GPR2.Options.Config`)

  Specify the configuration project file name

- `--db <dir>` (`GPR2.Options.Db`)

  Parse dir as an additional knowledge base

- `--db-` (`GPR2.Options.Db_Minus`)

  Do not load the standard knowledge base

- `--implicit-with` (`GPR2.Options.Implicit_With`)

  Add the given projects as a dependency on all loaded projects

- `--no-project` (`GPR2.Options.No_Project`)

  Do not use project file

- `-P<proj[.gpr]>` (`GPR2.Options.P`)

  Use Project File <proj>

- `--relocate-build-tree[=dir]` (`GPR2.Options.Relocate_Build_Tree`)

  Root obj/lib/exec dirs are current-directory or dir

- `--root-dir` (`GPR2.Options.Root_Dir`)

  Root directory of obj/lib/exec to relocate

- `--RTS[:lang]=<runtime>` (`GPR2.Options.RTS`)

  Use runtime <runtime> for language Ada or for language <lang>

- `--src-subdirs=<dir>` (`GPR2.Options.Src_Subdirs`)

  Prepend <obj>/dir to the list of source dirs for each project

- `-subdirs=<dir>` (`GPR2.Options.Subdirs`)

  Use dir as suffix to obj/lib/exec directories

- `--target=<targetname>` (`GPR2.Options.Target`)

  Specify a target for cross platforms

- `--unchecked-shared-lib-imports` (`GPR2.Options.Unchecked_Shared_Lib_Imports`)

  Shared lib projects may import any project

- `-X<nm>=<val>` (`GPR2.Options.X`)

  Specify a value for an external reference for project files

CHAPTER

THREE

ADA API TUTORIAL

3.1 Preliminary setup

In order to compile/run GPR2 tutorial examples you need to have GNAT compiler and libgpr2 library installed. Preliminary setup can be checked running

```
# Checking the GPR2 library installation
$ gprls -P gpr2
```

The first thing to do in order to use gpr2 is to load a project file:

```
with Ada.Text_IO; use Ada.Text_IO;

with GPR2.Containers;
with GPR2.Context;
with GPR2.Path_Name;
with GPR2.Project.Attribute;
with GPR2.Project.Attribute_Index;
with GPR2.Project.Registry.Attribute;
with GPR2.Project.Registry.Pack;
with GPR2.Project.Source;
with GPR2.Project.Tree;
with GPR2.Project.Typ;
with GPR2.Project.Unit_Info;
with GPR2.Project.Variable;
with GPR2.Project.View;
with GPR2.Source_Info;
with GPR2.Unit;

procedure Main is
   Context : GPR2.Context.Object;
   --  Context used to set scenarios

   Tree    : GPR2.Project.Tree.Object;
   --  "gpr2_tutorial.gpr" tree.

--  Insert tutorials here.

begin
   Tree.Load_Autoconf
     (Filename => GPR2.Path_Name.Create_File ("gpr2_test.gpr"),
```

(continues on next page)

(continued from previous page)

```
      Context  => Context);
   Tree.Log_Messages.all.Output_Messages;

--  Uncomment tutorial call
--   Views;
--   Scenarios;
--   Packages;
--   Attributes;
--   Types;
--   Variables;
--   Sources;
--   Units;
--   Registry;
end Main;
```

This very simple program will allow us to make sure the build environment is properly set to use gpr2. Save the above in a `main.adb` source file, and then write the following project file to `gpr2_test.gpr`:

```
with "gpr2";

project GPR2_Test is
   for Main use ("main.adb");
   for Object_Dir use "obj";
end GPR2_Test;
```

Now you can run `gprbuild` to compile the test program:

```
$ gprbuild -Pgpr2_test
```

This command should return without error and create an executable in `obj/main` or `obj\main.exe` depending on your platform. The last step is to check if the program works properly: it should do nothing, so no errors expected!

```
# Empty program output
$ obj/main
$
```

3.2 Views

When loading a gpr file in a Tree object a lot of gpr & cgpr files are loaded. In autoconf mode a configuration project is automatically generated. A configuration, a runtime project and all imported, extended or aggregated projects are recursively loaded.

All of these projects are available through `GPR2.Project.View.Object` objects. After a successful load, `Tree.Root_Project` contains the project passed when loading the tree.

```
procedure Views is
begin
   --  How to get views of a loaded tree.

   New_Line;
   Put_Line ("GPR2 project views");
```

(continues on next page)

(continued from previous page)

```
   --  root, configuration & runtime projects

   Put_Line ("gpr2_test.gpr at " & Tree.Root_Project.Path_Name.Value);
   Put_Line ("generated configuration project at " &
               Tree.Configuration.Corresponding_View.Path_Name.Value);
   Put_Line ("used runtime project at " &
               Tree.Runtime_Project.Path_Name.Value);

   --  View can be retrieved by name

   Put_Line ("gpr2.gpr at " &
               Tree.Root_Project.View_For ("gpr2").Path_Name.Value);

   --  Using a configurable iterator

   New_Line;
   for C in Tree.Iterate loop
      Put_Line ("view: " &
                  GPR2.Project.Tree.Element (C).Path_Name.Value);
   end loop;

   --  Using view's accessor functions

   New_Line;
   for V of Tree.Root_Project.Imports loop
      Put_Line ("imported: " & V.Path_Name.Value);
   end loop;

end Views;
```

As for all tutorial examples don't forget to add procedure call in `Main` procedure.

```
begin
   Tree.Load_Autoconf
     (Filename => GPR2.Path_Name.Create_File ("gpr2_test.gpr"),
      Context  => Context);
   Tree.Log_Messages.all.Output_Messages;

--  Uncomment tutorial call
   Views;
--   Scenarios;
--   Packages;
--   Attributes;
--   Types;
--   Variables;
--   Sources;
--   Units;
--   Registry;
end Main;
```

3.3 Scenarios

Various project properties can be modified based on scenarios. The above code shows how scenarios are handled by the GPR2 library.

```
procedure Scenarios is
   GPR2_View : constant GPR2.Project.View.Object :=
                  Tree.Root_Project.View_For ("gpr2");
begin
   --  Print current GPR2 Library_Kind using default scenarios values.

   New_Line;
   Put_Line ("Library_Kind (default): " &
               String (GPR2_View.Library_Kind));

   --  Change LIBRARY_TYPE to relocatable & print modified Library_Kind.

   Context.Insert ("LIBRARY_TYPE", "relocatable");
   Tree.Set_Context (Context);
   Put_Line ("Library_Kind (relocatable): " &
               String (GPR2_View.Library_Kind));
end Scenarios;
```

3.4 Packages

This tutorial shows how view's packages are listed. Attributes, variables and types parts will explain how packages are handled when accessing package content.

```
procedure Packages is
   GPR2_View : constant GPR2.Project.View.Object :=
                  Tree.Root_Project.View_For ("gpr2");

    procedure Print (With_Defaults : Boolean; With_Config : Boolean) is
      Print_Comma : Boolean := False;
   begin
      if With_Defaults then
         Put ("including packages defined by default values: ");
      end if;
      if With_Config then
         Put ("including packages defined by configuration file: ");
      end if;
      for Id of GPR2_View.Packages (With_Defaults, With_Config) loop
         if Print_Comma then
            Put (", ");
         end if;
         Print_Comma := True;
         Put (GPR2.Image (Id));
      end loop;
      New_Line;
   end Print;
begin
```

(continues on next page)

(continued from previous page)

```
   New_Line;
   Put_Line ("GPR2 project packages");
   Print (False, False);
   Print (True, False);
   Print (False, True);
end Packages;
```

3.5 Attributes

This tutorial shows how attributes can be accessed or listed.

```
procedure Attributes is
   GPR2_View : constant GPR2.Project.View.Object :=
                  Tree.Root_Project.View_For ("gpr2");

   procedure Print (Attribute : GPR2.Project.Attribute.Object);
   procedure Print (Attribute : GPR2.Project.Attribute.Object) is
      use GPR2;
      use GPR2.Project.Registry.Attribute;
   begin
      Put ("for " & Image (Attribute.Name.Id.Attr));
      if Attribute.Has_Index then
         Put (" (""" & Attribute.Index.Text & """");
         if Attribute.Index.Has_At_Pos then
            Put (" at " & Attribute.Index.At_Pos'Image);
         end if;
         Put (")");
      end if;
      if Attribute.Kind = GPR2.Project.Registry.Attribute.Single then
         Put (" use """ & Attribute.Value.Text & """");
         if Attribute.Value.Has_At_Pos then
            Put (" at " & Attribute.Value.At_Pos'Image);
         end if;
      else
         Put (" use " & GPR2.Containers.Image (Attribute.Values));
      end if;
      Put (";");
      if Attribute.Is_Default then
         Put (" -- default value");
      end if;
      if Attribute.Is_From_Config then
         Put (" -- from configuration project");
      end if;
      New_Line;
   end Print;

begin
   New_Line;
   Put_Line ("Compiler.Default_Switches (""Ada"") attribute");
   Print
```

(continues on next page)

(continued from previous page)

```
     (GPR2_View.Attribute
        (Name  => GPR2.Project.Registry.Attribute.Compiler.Default_Switches,
         Index => GPR2.Project.Attribute_Index.Create (GPR2.Ada_Language)));

   New_Line;
   Put_Line ("GPR2 project attributes");
   for Attribute of GPR2_View.Attributes loop
      Print (Attribute);
   end loop;
end Attributes;
```

3.6 Types

This tutorial shows how variables types can be accessed/listed.

```
procedure Types is
   GPR2_View : constant GPR2.Project.View.Object :=
                  Tree.Root_Project.View_For ("gpr2");

   procedure Print (Typ : GPR2.Project.Typ.Object) is
      Print_Comma : Boolean := False;
   begin
      Put (String (Typ.Name.Text) & ", values: ");
      for Value of Typ.Values loop
         if Print_Comma then
            Put (", ");
         else
            Print_Comma := True;
         end if;
         Put (Value.Text);
      end loop;
      New_Line;
   end Print;
begin
   New_Line;
   Put_Line ("GPR2 project types");
   Print (GPR2_View.Typ ("bool"));

   New_Line;
   Put_Line ("GPR2 project types");
   if GPR2_View.Has_Types then
      for Typ of GPR2_View.Types loop
         Print (Typ);
      end loop;
   end if;
end Types;
```

3.7 Variables

This tutorial shows how project-level and package-level variables can be listed and accessed.

```
procedure Variables is
   GPR2_View : constant GPR2.Project.View.Object :=
                  Tree.Root_Project.View_For ("gpr2");

   procedure Print (Variable : GPR2.Project.Variable.Object);
   procedure Print (Variable : GPR2.Project.Variable.Object) is
      use GPR2.Project.Registry.Attribute; --  "=" function visibility
   begin
      Put (String (Variable.Name.Text));
      if Variable.Has_Type then
         Put (", type: " & String (Variable.Typ.Name.Text));
      else
         Put (" with no type");
      end if;
      if Variable.Kind = GPR2.Project.Registry.Attribute.Single then
         Put (", value: " & String (Variable.Value.Text));
      else
         Put (", values: " & GPR2.Containers.Image (Variable.Values));
      end if;
      New_Line;
   end Print;
begin
   New_Line;
   Put_Line
     ("GPR2 project build & compiler.langkit_parser_options variable");
   Print (GPR2_View.Variable ("build"));
   if GPR2_View.Has_Variables
     (GPR2.Project.Registry.Pack.Compiler, "langkit_parser_options")
   then
      Print (GPR2_View.Variable
             (GPR2.Project.Registry.Pack.Compiler,
                "langkit_parser_options"));
   end if;

   New_Line;
   Put_Line ("GPR2 project variables");
   if GPR2_View.Has_Variables then
      for Variable of GPR2_View.Variables loop
         Print (Variable);
      end loop;
   end if;

   New_Line;
   Put_Line ("GPR2 project Compiler package variables");
   if GPR2_View.Has_Variables (GPR2.Project.Registry.Pack.Compiler) then
      for Variable of
        GPR2_View.Variables (GPR2.Project.Registry.Pack.Compiler) loop
         Print (Variable);
      end loop;
```

(continues on next page)

(continued from previous page)

```
   end if;
end Variables;
```

3.8 Sources

This tutorial shows how projects sources are parsed, listed, and accessed.

```
procedure Sources is
   GPR2_View : constant GPR2.Project.View.Object :=
                 Tree.Root_Project.View_For ("gpr2");
   Source    : GPR2.Project.Source.Object;
   Part      : GPR2.Project.Source.Source_Part;
   use GPR2.Unit;
begin

   --  Update_sources required after load to to get a source unless
   --  Tree.For_Each_Source or View.Sources was called.

   Source := GPR2_View.Source ("gpr2-project-tree.ads");
   if not Source.Is_Defined then
      Tree.Update_Sources
        (Stop_On_Error => True,
         With_Runtime  => False,
         Backends      => GPR2.Source_Info.All_Backends);

      Source := GPR2_View.Source ("gpr2-project-tree.ads");
   end if;

   New_Line;
   Put_Line ("gpr2-project-tree.ads source");
   Source := GPR2_View.Source ("gpr2-project-tree.ads");
   Put_Line (" kind is " & Source.Kind'Image);
   Put_Line (" unit name is " & String (Source.Unit_Name));

   if Source.Has_Other_Part then
      Put_Line
        (" other part is"
         & String (Source.Other_Part.Source.Path_Name.Simple_Name));
   end if;

   --  Separate file.

   Source := GPR2_View.Source ("gpr2-project-tree-load_autoconf.adb");
   if Source.Is_Defined and then Source.Kind = GPR2.Unit.S_Separate then
      Put_Line ("gpr2-project-tree-load_autoconf.adb unit name is "
                & String (Source.Unit_Name));
      Part := Source.Separate_From (GPR2.No_Index);
      Put_Line ("gpr2-project-tree-load_autoconf.adb separate from "
                & String (Part.Source.Path_Name.Simple_Name));
   end if;
```

(continues on next page)

(continued from previous page)

```
end Sources;
```

3.9 Units

This tutorial shows how units are listed and accessed.

```
procedure Units is
   GPR2_View : constant GPR2.Project.View.Object :=
                  Tree.Root_Project.View_For ("gpr2");
   Unit : GPR2.Project.Unit_Info.Object :=
            GPR2_View.Unit ("gpr2.project.tree");

   procedure Print
     (Prefix : String; SUI : GPR2.Unit.Source_Unit_Identifier) is
      use GPR2;
   begin
      Put (Prefix & String (SUI.Source.Simple_Name));
      if SUI.Index /= No_Index then
         Put (" at " & SUI.Index'Image);
      end if;
      New_Line;
   end Print;
begin
   if GPR2_View.Units.Is_Empty then

      --  Update_sources required after load to get a source or unit
      --  unless Tree.For_Each_Source or View.Sources was called.
      Put_Line ("Tree.Update_Sources");

      Tree.Update_Sources
        (Stop_On_Error => True,
         With_Runtime  => False,
         Backends      => GPR2.Source_Info.All_Backends);

   end if;

   Unit := GPR2_View.Unit ("gpr2.project.tree");

   New_Line;
   Put_Line ("Unit:" & String (Unit.Name));
   if Unit.Has_Spec then
      Print (" Specification in ", Unit.Spec);
   end if;
   if Unit.Has_Body then
      Print (" Body in ", Unit.Main_Body);
   end if;
   declare
      Separates : constant GPR2.Unit.Source_Unit_Vectors.Vector :=
                    Unit.Separates;
   begin
```

(continues on next page)

(continued from previous page)

```
      if not Separates.Is_Empty then
         for S of Separates loop
            Print (" Separate in ", S);
         end loop;
      end if;
   end;
 end Units;
```

3.10 Custom packages and attributes

This tutorial shows how custom packages and attributes can be added to gpr2 package/attribute registry. This should be done before loading projects.

```
procedure Registry is
   use GPR2;

   Custom_Package_Id   : constant Package_Id := +"custom";
   Custom_Attribute_Id : constant Q_Attribute_Id :=
                           (Custom_Package_Id, +"new_attribute");
begin
   --  Add package

   Project.Registry.Pack.Add
     (Name     => Custom_Package_Id,
      Projects => GPR2.Project.Registry.Pack.Everywhere);

   --  Add new attribute
   GPR2.Project.Registry.Attribute.Add
     (Name                 => Custom_Attribute_Id,
      Index_Type           =>
        Project.Registry.Attribute.FileGlob_Or_Language_Index,
      Index_Optional       => True,
      Value                => Project.Registry.Attribute.List,
      Value_Case_Sensitive => True,
      Is_Allowed_In        => Project.Registry.Attribute.Everywhere,
      Config_Concatenable  => True);
end Registry;
```