



GPR Project Files

Vincent Celier

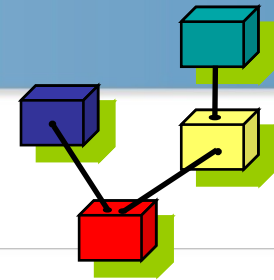
Senior software engineer, AdaCore

FOSDEM

8-9 February 2009

The GPR Project Facility

- **Overview**
- **GPR Project Files**
- **Property Values**
- **Setting Project Properties**
- **External and Conditional References**
- **Importing Projects**
- **Extending (“Modifying”) Projects**
- **Source File Naming Schemes**



The GPR Project Facility

- **Provides configurable properties for source files**
 - Represented by “projects”
- **Supports incremental, modular project definition**
 - Projects can **import** other projects containing needed files
 - *Child* projects can **extend** *parent* projects, inheriting source files and optionally overriding any of them
- **Facilitates the structuring of large development efforts into hierarchical subsystems**
 - With build decisions deferred to the subsystem level

Configurable Properties

- **Source directories and specific files' names**
- **Output directory for object modules and .ali files**
- **Target directory for executable programs**
- **Switch settings for project-enabled tools**
- **Source files for main subprogram(s) to be built**
- **Source programming languages**
 - Ada / C / C++ / Fortran / ...
- **Source file naming conventions**

Sample Uses

- **Common set of sources generating object files in different directories, via different switches**
 - “Debug” version
 - “Release” version
- **Multiple versions of the body for a package spec**
- **Note this is not a configuration management facility**
 - “Projects” should be under CM too!



GPR Project Files

- **Text files with Ada-like syntax**
- **Integrated into command-line tools**
 - Specified via the `-Pproject-file-name` switch
- **Integrated into GPS**
 - A fundamental part of the IDE
 - Automatically generated
 - There is even a Project Wizard



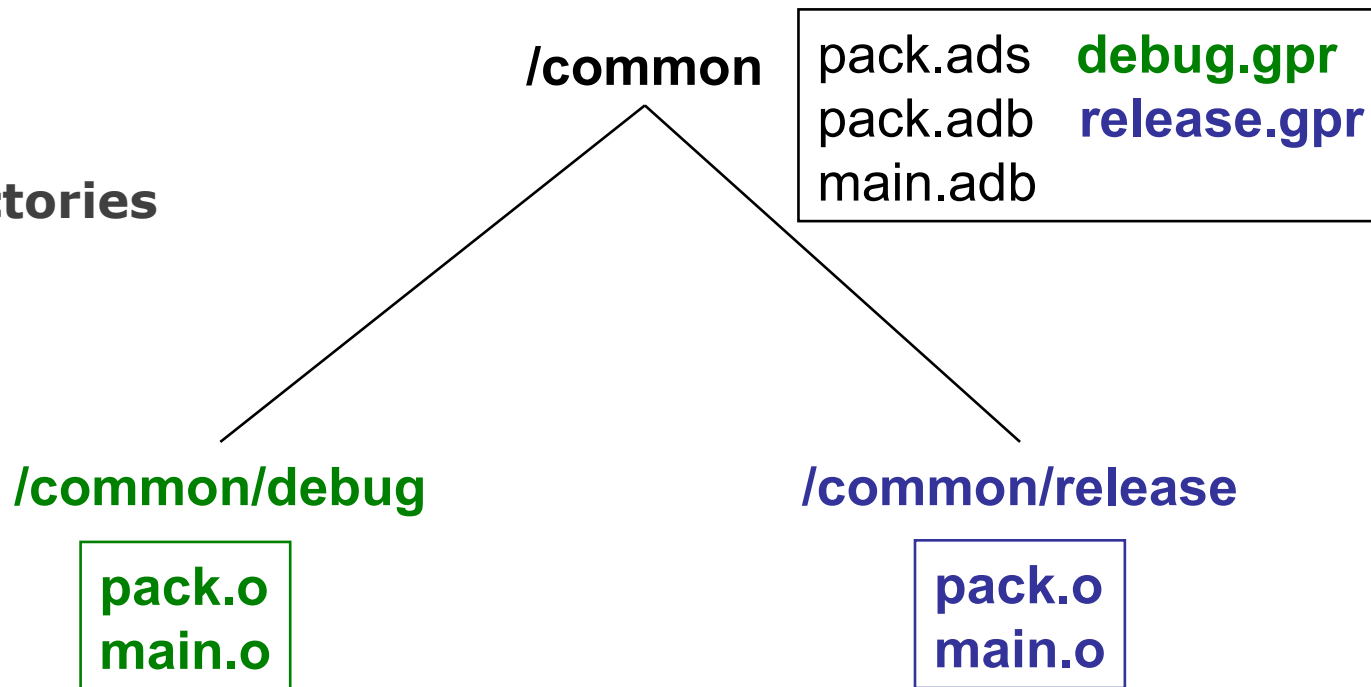
Property Values Introduction

- **Strings**
- **Lists of strings** `("-v", "-gnatv")`
- **Associative arrays**
 - Like functions that map an input string to either a single string or a list of strings

```
for Switches ("Ada") use ("-v", "-gnatv");
```

Common Files with Different Switches Example

- **Directories**



- **Example usage for debug case**

- `gnatmake -P/common/debug.gpr /common/main.adb`

Common Files with Different Switches Projects

```
project Debug is  
  for Object_Dir use "debug";  
  package Builder is  
    for Default_Switches ("Ada") use ("-g");  
  end Builder;  
  package Compiler is  
    for Default_Switches ("Ada")  
      use ("-fstack-check", "-gnata", "-gnato");  
    end Compiler;  
end Debug;
```

```
project Release is  
  for Object_Dir use "release";  
  package Compiler is  
    for Default_Switches ("Ada") use ("-O2");  
  end Compiler;  
end Release;
```

Conventions for Naming GPR Project Files

- **File name should match project name**
 - Not case sensitive
- **Extension should be “.gpr”**
- **Warnings issued if not followed**



Setting Project Properties

- **Packages**
- **Variables**
- **Types**
- **Switches**
- **Source Directories**
- **Source Files**
- **Object Directory**
- **Executable Directory**

Packages In GPR Project Files

- **Package names correspond to tools**
 - Builder
 - Compiler
 - Linker
 - Binder
 - etc.
- **Allowable names and content defined by facility**
 - Not by users

Typed Versus Untyped Variables

- **Typed variables have only listed values possible**
 - Case sensitive, unlike Ada
- **Typed variables are declared *once* per scope**
 - Once at project level
 - Once within any package
 - Essentially read-only constants
 - Especially nice for external inputs
- **Untyped variables may be “declared” many times**
 - No previous declaration required

Switches In GPR Project Files

- **May be specified to apply by default**

```
package Compiler is  
  for Default_Switches ("Ada") use ("-gnaty", "-v");  
end Compiler;
```

- **May be specified on a *per-unit* basis**

- Associative array "Switches" indexed by unit name

```
package Builder is  
  for Switches ("main1.adb") use ("-O2");  
  for Switches ("main2.adb") use ("-g");  
end Builder;
```

Source Directories In GPR Project Files

- One or more in any project file
- Default is same directory as project file
- Can specify additional individual directories

```
for Source_Dirs use ("mains", "drivers");
```

- Can specify additional directory trees

```
for Source_Dirs use ("foo/**", "./**");
```

- Can specify that none are present

```
for Source_Dirs use ();
```

Source Files In GPR Project Files

- **Must contain at least one “immediate” source file**
 - In one of the source directories of the project file
 - Unless explicitly specifies none present

```
for Source_Files use ();
```

- **Can specify source files by name**

```
for Source_Files use ("main.adb","pack1.ads","pack2.adb");
```

- **Can specify an external *file* containing source names**

```
for Source_List_File use "files.list";
```

Object Directory In GPR Project Files

- **Specifies the location for compiler's output**
 - Such as "ali" files and object files
 - For the project's *immediate* sources

```
project Release is
  for Object_Dir use "release";
  ...
end Release;
```

- **Only one per project**
 - When extending a parent project the child's object directory is used for any inherited sources not already compiled in the parent

Executable Directory In GPR Project Files

- **Specifies the location for executable image**

```
project Release is
  for Exec_Dir use "executables";
  ...
end Release;
```

- **Default is same directory as object files**
- **Only one per project**

External and Conditional References

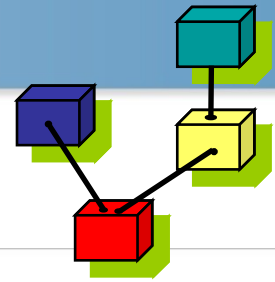
- **Allow project file content to depend on value of environment variables & command-line arguments**
- **Reference to external values is by function**
 - `external(name [, default])` returns value of *name* as supplied on the command line or as environment variable
 - If *name* is undefined, return *default* (if supplied) or ""
- **Set via command line switch (for example)**
 - `gnatmake -P... -Xname=value ...`

```
gnatmake -P/common/build.gpr -Xtarget=test /common/main.adb
```

External/Conditional Reference Example

```
project Build is
  type Targets is ("release", "test");
  Target : Targets := external("target", "test");
  case Target is -- project attributes
    when "release" =>
      for Object_Dir use "release";
      for Exec_Dir use ".";
    when "test" =>
      for Object_Dir use "debug";
  end case;

  package Compiler is
    case Target is
      when "release" =>
        for Default_Switches ("Ada") use ("-O2");
      when "test" =>
        for Default_Switches ("Ada") use
          ("-g", "-s", "-gnata", "-gnato", "-gnatE");
    end case;
  end Compiler;
end Build;
```



Importing Projects

- **Compilation units in source files of one project may depend on compilation units in source files of others**
 - “Depend” in the Ada sense (contains with-clauses)
- **We want to localize properties of other projects**
 - Switches etc.
 - Defined in one place and not repeated elsewhere
- **Thus dependent projects “import” other projects to place other projects’ source files on search path**

Project Import Notation

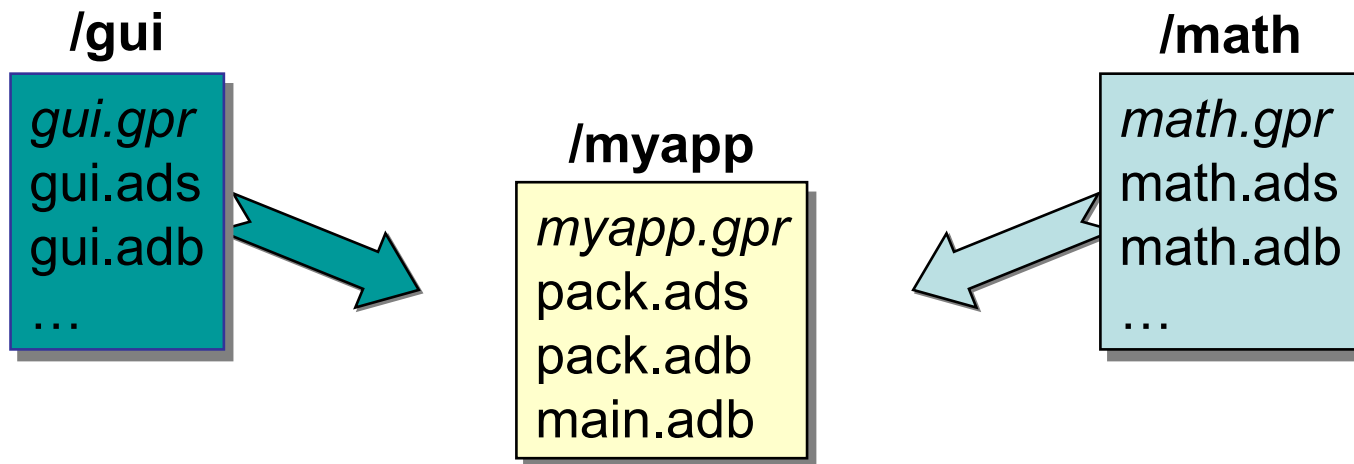
- **Similar to Ada's "with" clauses**
 - But uses strings

```
with literal_string { , literal_string } ;
```

- **String literals are path names of project files**
 - Relative
 - Absolute

Importing Projects Example

```
with GUI, Math;  
package body Pack is ...
```



```
with "/gui/gui.gpr", "/math/math";  
project MyApp is  
...  
end MyApp;
```

Project Path

- **Locates GPR project files referenced by other projects**
- **Combined content of:**
 - Directory containing GPR project file
 - GPR_PROJECT_PATH environment variable (if defined)
- **Note “with” clauses are transitive**
 - Indirectly imported projects are automatically included
 - Unlike Ada “with” clauses
 - A project is responsible for making its own imports work regardless of whether or not it is imported elsewhere

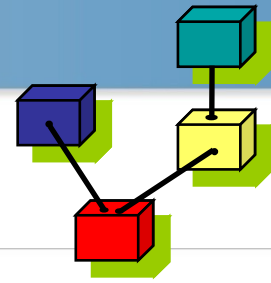
Project Path Examples

- **If** /gui,/math *are on* GPR_PROJECT_PATH

```
with "gui.gpr", "math";  
project MyApp is ...
```

- **If** /gui,/math *are not on* GPR_PROJECT_PATH

```
with "/gui/gui.gpr", "/math/math";  
project MyApp is ...
```

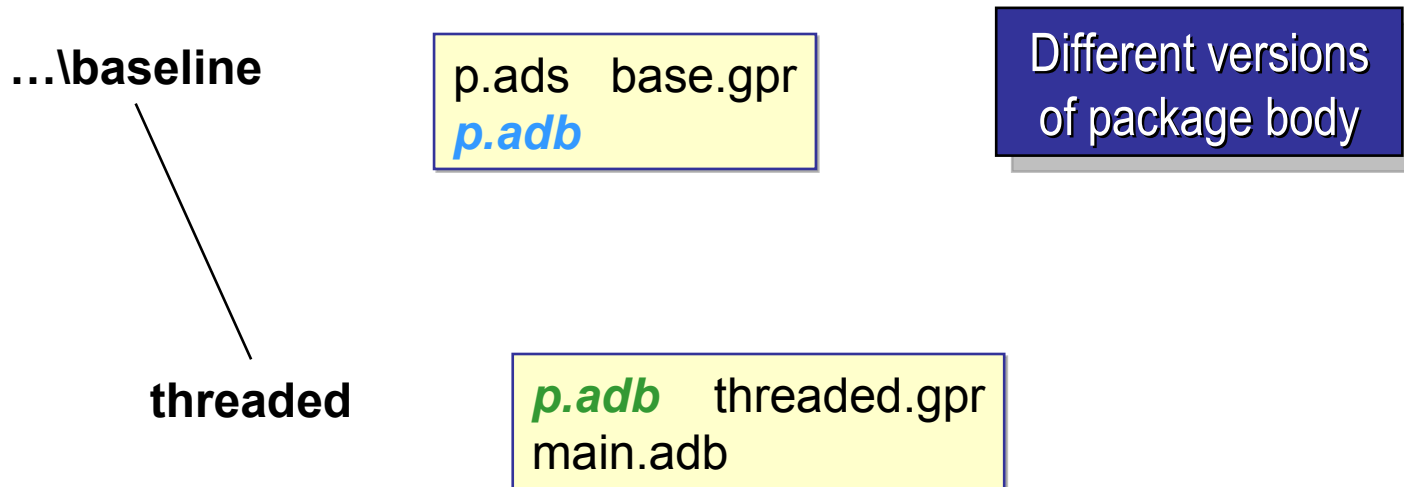


Extending (“Modifying”) Projects

- **Allows using modified versions of source files without changing the original sources**
- **Based on “inheritance” of parent project’s properties**
 - Source files
 - Switch settings
- **Supports localized build decisions and properties**
 - Inherited properties may be overridden with new versions
- **Hierarchies permitted**

Multiple Versions of Unit Bodies Example

- Assume this directory structure & files



- Usage (assume baseline is current directory)

```
gnatmake -Pthreaded/threaded.gpr main
```

Multiple Versions of Unit Bodies File

```
project Baseline is  
end Baseline;
```

```
project Threaded extends "/source/ada/dev/baseline" is  
end Threaded;
```

Source File Naming Schemes

- **Allow arbitrary naming conventions**
 - Other than default convention
- **May be applied to *all* source files in a project**
 - Specified in a package named "Naming"
- **May be applied to *specific* files in a project**
 - Individual attribute specifications

Foreign Default File Naming Example

```
project APEX is  
  for Source_Files use ();  
  package Naming is  
    for Casing use "lowercase";  
    for Dot_Replacement use ".";  
    for Specification_Suffix ("Ada") use ".1.ada";  
    for Implementation_Suffix ("Ada") use ".2.ada";  
  end Naming;  
end APEX;
```

GNAT Default File Naming Example

```
project GNAT is  
  for Source_Files use ();  
  
  package Naming is  
    for Casing use "lowercase";  
    for Dot_Replacement use "-";  
    for Specification_Suffix ("Ada") use ".ads";  
    for Implementation_Suffix ("Ada") use ".adb";  
  end Naming;  
  
end GNAT;
```

Individual (Arbitrary) File Naming

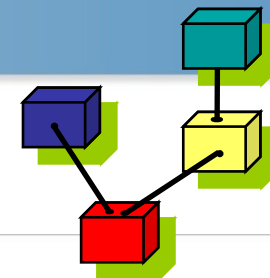
- **Uses associative arrays to specify file names**
 - Index is a string containing the unit name
 - Value is a string
 - Case sensitivity depends on host file system
- **Has distinct attributes for specifications and bodies**

for Specification ("MyPack.MyChild")
use "mypack.mychild.spec";

for Implementation ("MyPack.MyChild")
use "mypack.mychild.body";

GPS Directly Supports Projects

- **The fundamental conceptual basis for GPS**
- **Graphically displays a project hierarchy**
- **Provides a wizard to help you create projects**
- **Displays and navigates project sources**
- **Compiles sources defined by a project**
- **Creates executables for project main programs**
- **Debugs a project's executables**
- **et cetera**



GPR Project Manager Summary

- **Supports hierarchical, localized build decisions**
- **Supports arbitrary file naming conventions**
- **GPS provides direct support**
- **See the GNAT User's Guide for further details!**

