Next: Object and Exec Directory, Up: Building With Projects

11.2.1 Source Files and Directories

When you create a new project, the first thing to describe is how to find the corresponding source files. This is the only settings that are needed by all the tools that will use this project (builder, compiler, binder and linker for the compilation, IDEs to edit the source files,…).

First step is to declare the source directories, which are the directories to be searched to find source files. In the case of the example, the `common` directory is the only source directory.

There are several ways of defining source directories:

- When the attribute **Source_Dirs** is not used, a project contains a single source directory which is the one where the project file itself resides. In our example, if `build.gpr` is placed in the `common` directory, the project has the needed implicit source directory.
- The attribute **Source_Dirs** can be set to a list of path names, one for each of the source directories. Such paths can either be absolute names (for instance `"/usr/local/common/"` on UNIX), or relative to the directory in which the project file resides (for instance "." if `build.gpr` is inside `common/`, or "common" if it is one level up). Each of the source directories must exist and be readable.

  The syntax for directories is platform specific. For portability, however, the project manager will always properly translate UNIX-like path names to the native format of specific platform. For instance, when the same project file is to be used both on Unix and Windows, "/" should be used as the directory separator rather than "\".

- The attribute **Source_Dirs** can automatically include subdirectories using a special syntax inspired by some UNIX shells. If any of the path in the list ends with `"**"`, then that path and all its subdirectories (recursively) are included in the list of source directories. For instance, `**` and `./**` represent the complete directory tree rooted at ".". When using that construct, it can sometimes be convenient to also use the attribute **Excluded_Source_Dirs**, which is also a list of paths. Each entry specifies a directory whose immediate content, not including subdirs, is to be excluded. It is also possible to exclude a complete directory subtree using the "**" notation.

  It is often desirable to remove, from the source directories, directory subtrees rooted at some subdirectories. An example is the subdirectories created by a Version Control System such as Subversion that creates directory subtrees rooted at subdirectories ".svn". To do that, attribute **Ignore_Source_Sub_Dirs** can be used. It specifies the list of simple file names for the roots of these undesirable directory subtrees.

  ```
               for Source_Dirs use ("./**");
               for Ignore_Source_Sub_Dirs use (".svn");
  ```

When applied to the simple example, and because we generally prefer to have the project file at the toplevel directory rather than mixed with the sources, we will create the following file

```
       build.gpr
       project Build is
          for Source_Dirs use ("common");  --  <<<<
       end Build;
```

Once source directories have been specified, one may need to indicate source files of interest. By default, all source files present in the source directories are considered by the project manager. When this is not desired, it is possible to specify the list of sources to consider explicitly. In such a case, only source file base names are indicated and not their absolute or relative path names. The project manager is in charge of locating the specified source files in the specified source directories.

- By default, the project manager search for all source files of all specified languages in all the source directories.

  Since the project manager was initially developed for Ada environments, the default language is usually Ada and the above project file is complete: it defines without ambiguity the sources composing the project: that is to say, all the sources in subdirectory "common" for the default language (Ada) using the default naming convention.

  However, when compiling a multi-language application, or a pure C application, the project manager must be told which languages are of interest, which is done by setting the **Languages** attribute to a list of strings, each of which is the name of a language. Tools like `gnatmake` only know about Ada, while other tools like `gprbuild` know about many more languages such as C, C++, Fortran, assembly and others can be added dynamically.

  Even when using only Ada, the default naming might not be suitable. Indeed, how does the project manager recognizes an "Ada file" from any other file? Project files can describe the naming scheme used for source files, and override the default (see Naming Schemes). The default is the standard GNAT extension (`.adb` for bodies and `.ads` for specs), which is what is used in our example, explaining why no naming scheme is explicitly specified. See Naming Schemes.

- `Source_Files` In some cases, source directories might contain files that should not be included in a project. One can specify the explicit list of file names to be considered through the **Source_Files** attribute. When this attribute is defined, instead of looking at every file in the source directories, the project manager takes only those names into consideration reports errors if they cannot be found in the source directories or does not correspond to the naming scheme.
- For various reasons, it is sometimes useful to have a project with no sources (most of the time because the attributes defined in the project file will be reused in other projects, as explained in see Organizing Projects into Subsystems. To do this, the attribute *Source_Files* is set to the empty list, i.e. `()`. Alternatively, *Source_Dirs* can be set to the empty list, with the same result.
- `Source_List_File` If there is a great number of files, it might be more convenient to use the attribute **Source_List_File**, which specifies the full path of a file. This file must contain a list of source file names (one per line, no directory information) that are searched as if they had been defined through *Source_Files*. Such a file can easily be created through external tools.

  A warning is issued if both attributes `Source_Files` and `Source_List_File` are given explicit values. In this case, the attribute `Source_Files` prevails.

- `Excluded_Source_Files` Specifying an explicit list of files is not always convenient.It might be more convenient to use the default search rules with specific exceptions. This can be done thanks to the attribute **Excluded_Source_Files** (or its synonym **Locally_Removed_Files**). Its value is the list of file names that should not be taken into account. This attribute is often used when extending a project, See Project Extension. A similar attribute **Excluded_Source_List_File** plays the same role but takes the name of file containing file names similarly to `Source_List_File`.

In most simple cases, such as the above example, the default source file search behavior provides the expected result, and we do not need to add anything after setting `Source_Dirs`. The project manager automatically finds `pack.ads`, `pack.adb` and `proc.adb` as source files of the project.

Note that it is considered an error for a project file to have no sources attached to it unless explicitly declared as mentioned above.

If the order of the source directories is known statically, that is if `"**"` is not used in the string list `Source_Dirs`, then there may be several files with the same source file name sitting in different directories of the project. In this case, only the file in the first directory is considered as a source of the project and the others are hidden. If `"**"` is used in the string list `Source_Dirs`, it is an error to have several files with the same source file name in the same directory `"**"` subtree, since there would be an ambiguity as to which one should be used. However, two files with the same source file name may exist in two single directories or directory subtrees. In this case, the one in the first directory or directory subtree is a source of the project.