

Gem #64: Обработка многоэлементных файлов исходного кода

Автор: AdaCore

Краткое содержание: Gem Ada #64 - В данном Gem демонстрируется, как можно компилировать приложения в GNAT, если в файле исходного кода содержится множество элементов. Лучше всего разбить файл исходного кода на несколько частей, и в данном Gem #64 будет показано, как именно можно этого добиться, но, помимо этого, в GNAT доступен обходной вариант, который позволяет сохранить файл исходного кода.

Давайте начнём...

Во многих устаревших системах, которые были первоначально разработаны с использованием технологий, отличных от GNAT, один исходный файл может включать в себя несколько единиц Ada (как спецификации, так и тела одного пакета).

Там, где это возможно, лучше всего следовать традиционному подходу GNAT и разделить эти файлы на несколько файлов. Основным преимуществом этого является, чтобы ограничить количество перекомпиляции, что `gnatmake` и `gprbuild` будет делать. При обнаружении нескольких единиц в одном файле изменение любого из них приведёт к перекомпиляции всех файлов, зависящих от любого из единиц, найденных в файле. В частности, изменение тела пакета, когда Спецификация находится в одном исходном файле, приведёт к перекомпиляции всех объектов, зависящих от него. Это противоречит всей концепции отдельных спецификаций.

Это также рекомендованный метод в руководстве по качеству и стилю Ada (*Ada Quality and Style manual*), поэтому это должно действительно быть предпочтительным подходом, где возможно.

Такое разделение исходных файлов можно выполнить с помощью инструмента командной строки `gnatchop`. Если вы хотите сохранить исходные (несколько единиц) исходные файлы и продолжать их изменять, мы рекомендуем использовать переключатель `"-r"`, который заставит сообщения об ошибках GNAT ссылаться на исходный (`unsplit` – не разделенный) файл.

Если, однако, вы не намерены держать их, Вы не должны использовать `"-r"`.

Скорее всего, переключатель `"-w"` также пригодится, чтобы принудительно перезаписать ранее разделённые файлы. Кроме того, поскольку вы воссоздаёте исходные файлы, которые видит `gnatmake`, вы должны использовать переключатель `"-m"` для `gnatmake`, чтобы он не обращал внимания на временные метки. В противном случае вы бы в конечном итоге будет перекомпиляция всех файлов всё время.

Вызов `gnatchop` не может быть выполнен автоматически `gnatmake`. Вместо этого у вас должен быть скрипт или файл `Makefile`, который сначала вызывает `gnatchop`, а затем порождает `gnatmake` для выполнения фактической компиляции.

Однако в некоторых случаях невозможно разделить исходные файлы. Чаще всего, это происходит потому, что файлы находятся под контролем конфигурации и необходимо сохранить историю. В других случаях это происходит потому, что вы все ещё хотите скомпилировать со старым компилятором.

GNAT предоставляет обходной путь в случаях, когда вы не можете разделить исходные файлы и не можете или не хотите изменять методы сборки для использования `gnatchop`. Давайте подчеркнём, что это действительно рассматривается, как только обходной путь, и не полностью поддерживается всеми инструментами. Например, коммутаторы `"-gnatD"` и `"-gnatR"` в настоящее время несовместимы с прагмами, описанными ниже. Что ещё более важно, возможно, GPS будет иметь некоторые известные ограничения при использовании нескольких исходных

файлов (перекрёстные ссылки не будут работать во многих случаях, и компиляция текущего файла не удастся, если он содержит несколько единиц)

Вы должны сообщить GNAT о Вашей организации исходного файла, что можно сделать несколькими способами.

Способ 1. Если вы используете файлы проекта, вы можете добавить или создать специальный пакет Naming в отдельном файле внутри вашего проекта и использовать ниже приведённый синтаксис, например:

```
package Naming is
  for Specification ("unit1") use "file1.ada" at 1;
  for Implementation ("unit1") use "file1.ada" at 2;
end Naming;
```

Индекс после at начинается с 1 и указывает положение объекта внутри исходного файла.

Вы можете или создать этот пакет Naming вручную, редактируя файл проекта, или при помощи gnatname инструмента командной строки. Этот инструмент просмотрит все каталоги, Вы передаёте его через его переключатель "-d", проверяет каждый файл в них и определяет, какой модуль или модули они содержат. После этого создаст новый файл проекта, названный my_project_naming.gpr, и изменит проект, который Вы определили в командной строке способом, аналогичным приведённым ниже:

```
with "my_project_naming";
project My_Project is
  package Naming renames My_Project_Naming.Naming;
end My_Project;
```

GPS был недавно изменён, чтобы правильно поддерживать такие файлы проекта, и правильно сохранить существующую информацию " at " при редактировании проекта, а также, чтобы позволить вам создавать новые исключения именования непосредственно из графического интерфейса.

Способ 2. Если вы ещё не используете файлы проекта, вы можете указать ту же информацию, используя прагмы в файле конфигурации. Через свой «-с» переключатель gnatname также может генерировать этот файл прагм. Общая форма файла конфигурации:

```
pragma Source_File_Name
  (Unit1, Spec_File_Name => "file1.ada", Index => 1);
pragma Source_File_Name
  (Unit1, Body_File_Name => "file1.ada", Index => 2);
```

где информация похожа на то, что было указано в примере файла проекта выше.

Существует разница между использованием этих атрибутов проекта или прагм, и использованием "gnatchop -r", как мы описали в первой части: в то время как существует одна команда для порождения, и нет дублирования исходных файлов с прагмами, у вас все еще есть проблема, что gnatmake будет перекомпилировать все, что зависит от любой единицы в файле. При использовании gnatchop, однако, что проблемы не существует, потому что gnatmake видит один-блок исходных файлов. Кроме того, метод на основе pragma ещё не полностью поддерживается всеми инструментами, поэтому предпочтительным подходом является разделение файлов (и предпочтительно заменить исходные файлы с несколькими единицами на результирующие файлы с одной единицей).

Связанный со статьёй текст программы

Attached Files отсутствуют

Файлы примеров Ada Gems распространяются AdaCore и могут быть использованы или изменены для любых целей без ограничений.

Об авторе

Сведения об авторе отсутствуют.

Last Updated: 10/13/2017

Posted on: 5/4/2009

Обсуждение...

Нет