Gem #60: Генерация связей с заголовочными файлами C++ в Ada.

Автор: Arnaud Charlet, AdaCore

Краткое содержание: Gem Ada #60 - В Gem #59 было показано, как можно легко автоматически генерировать связи с заголовочными файлами C в Ada. В данном Gem будет продемонстрировано, как с не меньшей лёгкостью можно генерировать связи с заголовочными файлами C++.

Давайте начнем…

Создание привязок для заголовков C ++ выполняется с использованием тех же параметров, что и для заголовков C, снова используя драйвер G ++. Это связано с тем, что инструмент генерации привязки использует все готовые C и C ++ фронты, доступные с GCC, а затем переводит его внутреннее представление в соответствующий код Ada.

В этом режиме классы C ++ будут сопоставляться с тегами Ada, конструкторы будут отображаться с использованием прагмы CPP_Constructor, и, когда это возможно, множественное наследование абстрактных классов будет отображаться на интерфейсы Ada.

Полный пример

Например, учитывая следующий заголовочный файл C ++ под названием animals.h:

```
class Carnivore {
public:
  virtual int Number_Of_Teeth () = 0;
};

class Domestic {
public:
  virtual void Set_Owner (char* Name) = 0;
};

class Animal {
public:
  int Age_Count;
  virtual void Set_Age (int New_Age);
};

class Dog : Animal, Carnivore, Domestic {
public:
  int Tooth_Count;
  char *Owner;

  virtual int  Number_Of_Teeth ();
  virtual void Set_Owner (char* Name);

  Dog();
};
```

Мы создаем пакет Ada с:

```
$ g++ -c -fdump-ada-spec animals.h
```

который генерирует файл animals_h.ads, который имеет следующее содержимое:

```
package animals_h is

   package Class_Carnivore is
     type Carnivore is limited interface;
     pragma Import (CPP, Carnivore);
```

```
    function Number_Of_Teeth (this : access Carnivore) return int is abstract;
  end;
  use Class_Carnivore;

  package Class_Domestic is
    type Domestic is limited interface;
    pragma Import (CPP, Domestic);

    procedure Set_Owner
      (this : access Domestic;
       Name : Interfaces.C.Strings.chars_ptr) is abstract;
  end;
  use Class_Domestic;

  package Class_Animal is
    type Animal is tagged limited record
      Age_Count : aliased int;
    end record;
    pragma Import (CPP, Animal);

    procedure Set_Age (this : access Animal; New_Age : int);
    pragma Import (CPP, Set_Age, "_ZN6Animal7Set_AgeEi");
  end;
  use Class_Animal;

  package Class_Dog is
    type Dog is new Animal and Carnivore and Domestic with record
      Tooth_Count : aliased int;
      Owner : Interfaces.C.Strings.chars_ptr;
    end record;
    pragma Import (CPP, Dog);

    function Number_Of_Teeth (this : access Dog) return int;
    pragma Import (CPP, Number_Of_Teeth, "_ZN3Dog15Number_Of_TeethEv");

    procedure Set_Owner
      (this : access Dog; Name : Interfaces.C.Strings.chars_ptr);
    pragma Import (CPP, Set_Owner, "_ZN3Dog9Set_OwnerEPc");

    function New_Dog return Dog&apos;Class;
    pragma CPP_Constructor (New_Dog, "_ZN3DogC1Ev");
  end;
  use Class_Dog;

end animals_h;
```

Как вы можете видеть, классы C ++ сопоставляются с тегами Ada и даже интерфейсами, когда это возможно. Другим преимуществом непосредственного использования компилятора C ++ для генерации привязок является то, что информация о C ++ коверкание (mangling) доступна и может быть легко сгенерирована, в то время как запись вручную может быть утомительной и подверженной ошибкам.

C ++ коверкание (mangling) - это низкоуровневое имя, выбранное компилятором C ++ для данной функции, например, «_ZN3Dog9Set_OwnerEPc» в случае процедуры `Class_Dog.Set_Owner` в приведённом выше примере.

В следующем Gem мы подробно рассмотрим, как работает прагма `CPP_Constructor` и как объединить её с функциями Ada.

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

Данные об авторе отсутствуют.

Last Updated: 10/13/2017
Posted on: 3/9/2009

Обсуждение…