AdaDoc

How to write a module for AdaDoc.

August 28, 2002

Contents

1 Introduction

This manual describes the structure of an AdaDoc module, the given features and the instructions to write your own module.

1.1 What is an AdaDoc module?

AdaDoc modules are the last step to create the output file. The modules content the rules and the way to write HTML, $\LaTeX$, plan text or any other format. A module could write a specification (header) for others programming language, send the specifications directly in a database as well.

The system of module had been created in such way that any user of AdaDoc could write his own modules easily.

1.2 Required knowledge

Any particular knowledge is required. You just have to know the basic of the Ada95 language to write a module.

AdaDoc use the OOP paradigm, but you don't need to know the Ada95 OOP to write a module.

The most important knowledge you must have, are required by the output format.

1.3 Required software

You need to recompile AdaDoc, if you want to add a module to it. For that you need the following software

- XML/Ada[1] version 0.7.1.
- GNAT[2] (XML/Ada depend on GNAT).
- AdaDoc sources[3] and the XSD diagram.

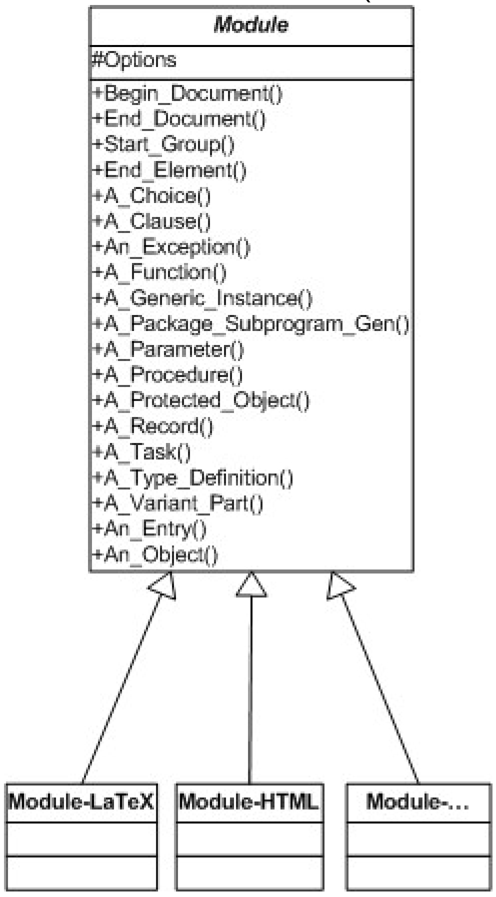
[1]XML/Ada a full XML suite: http://libre.act-europe.fr/xmlada/
[2]GNAT Ada95 compiler: http://www.gnat.com
[3]AdaDoc: http://AdaDoc.sf.net

1.4 Modules implementation

Go to the next section, if you don't want to know the implementation's details.

The last state of AdaDoc (after the XML file creation) is to parse the XML file to write the final output file. The XML parser (api SAX of XML/Ada) sends callbacks every time it reads a XML element. This callback are caught and processed by the AdaDoc interface and finally send to the modules. If a child module had redefined a callback, it would be treated otherwise it would do nothing. This allows to not redefined all the callbacks.

An AdaDoc module is just an inheritance of the Module (Modules.ads) class.



UML reprenstation of module classes.

2 Writing a module

This section describes step by step how to write a module.

We are going to write an example module called *xy*. This module writes information of an Ada95 specification in a text file.

The XSD diagram shows the arborescence of the XML elements. Take look at it to understand the different callbacks.

We are frequently using a dictionary abstract data type in AdaDoc. Its specification (*dict_tda.ads*) is in the tools directory of the AdaDoc sources.

2.1 Writing the module specification

Create a *module-xy.ads* file in directory module. Edit the file and add the lines:

```
package Module.Xy is
   type Object is new Module.Object with null record;

   procedure Begin_Document (Self : Object; Package_Name : String;
                             Atts, Tags : Dict_Type);

   procedure End_Document (Self : Object);

end Module.Xy;
```

It is the specification of the smallest module you could write. You should redefine the **Begin_Document** and the **End_Document** procedures because they are abstract.

2.2 Redefine the callbacks

You have to choose which callback you want to redefine. All the callbacks are in the *module.ads* file.

For your module we are going to write only the procedures and their parameters. We need to redefine the following callbacks:

- **Start_Group** To know when we have begun a group of procedures or a group of parameters.
- **A_Procedure** A procedure.
- **A_Parameter** A parameter.

The *module-xy.ads* file look like that now:

```
package Module.Xy is
   type Object is new Module.Object with null record;

   procedure Begin_Document (Self : Object; Package_Name : String;
                             Atts, Tags : Dict_Type);

   procedure End_Document (Self : Object);

   procedure Start_Group (Self : Object; Group : Element_Type);

   procedure A_Parameter (Self : Object; Atts : Dict_Type; Its_Name,
                          Its_Type, Its_Mode : String);

   procedure A_Procedure (Self : Object; Atts : Dict_Type;
                          Its_Name : String; In_Prot_Obj : Boolean);
end Module.Xy;
```

2.3 Writing the module body

We need to write the body of the package. We are using the *gnatstub* tools for that [4]. After we have run `gnatstub module-xy.ads` on the command line, the file *module-xy.adb* had been created in the same directory that the specification.

[4]gnatstub is a gnat tools to write a package body from a specification.

2.4 Writing of the Begin_Document and the End_Document procedures

These two procedures are going to open and close the output text file. As a module is an abstract machine, we can declare the file directly in the body of the package.

```
with Ada.Text_IO; use Ada.Text_IO;
package body Module.Xy is

   Fichier : File_Type; -- Declaration de notre fichier

   procedure Begin_Document (Self : Object; Package_Name : String;
                             Atts, Tags : Dict_Type) is
   begin
      Create(Fichier, Name => Package_Name & ".txt");
      Put_Line(Fichier, "Paquetage : " & Package_Name);
   end Begin_Document;

   procedure End_Document (self : Object) is
   begin
      New_Line(Fichier);
      Put_Line(Fichier, "Fin.");
      Close(Fichier);
   end End_Document;
   ...
end Module.Xy;
```

2.5 Callbacks' body

2.5.1 Start_Group and End_Element callbacks

We are going to write the **Start_Group** callback, which announce the beginning of any XML element. You can see the elements' list in the XSD diagram or in the **Element_Type** type of *module.ads*.

We need the **Is_Subprograms** callback to know when a subprogram begins and the **Is_Parameters** to know when the parameters begin.

```
...
package body Module.Xy is
   ...
   procedure Start_Group (Self : Object; Group : Element_Type) is
   begin
      case Group is
         when Is_Subprograms =>
            Put_Line(Fichier, "Voici la liste des sous programmes");
            Put_Line(Fichier, "———————————————————————————————————");
         when Is_Parameters =>
            Put_Line(Fichier, "Liste des parametres :");
         when others =>
            null;
      end case;
   end Start_Group;
   ...
end Module.Xy;
```

NB: The **End_Element** callback works the same way, but it is called at the end of a group or an element. We need this call back to close a table (for example </TABLE> in HTML).

2.5.2 Standard callbacks

We are going to write the body of the **A_Procedure** and the **A_Parameter** callbacks and using their parameters.

```
...
   procedure A_Procedure (Self : Object; Atts : Dict_Type;
                          Its_Name : String; In_Prot_Obj : Boolean) is
   begin
      New_Line(Fichier);
      Put_Line(Fichier, Its_Name & " une procedure...");
   end A_Procedure;


   procedure A_Parameter (Self : Object; Atts : Dict_Type; Its_Name,
                          Its_Type, Its_Mode : String) is
   begin
      Put_Line(Fichier, Its_Name & " est de type " & Its_Type &
               " et de mode de passage " & Its_Mode);
   end A_Parameter;
...
```

2.6 Options and the attributes' dictionary

In this state our module is already usable. But to finish it, we need to talk about the options and the attributes' dictionary.

AdaDoc user can give options to module. To catch them, each module has a **Option_Exist** function that takes the options string parameters and return if the user as given this option or not.

Nearly all the callbacks have an attributes' dictionary as parameters. It contains all the optional value of a callback (for example a comment). To know the existence or the absence of an attribute in the dictionary we should use the **Exist** function. The **Value_Of** function returns the value of an attribute.

We are going to modify the **A_Procedure** procedure to make an example. If the user as given the `c` option to AdaDoc and a comment exist for the procedure then we are going to print this comment.

```
...
   procedure A_Procedure (Self : Object; Atts : Dict_Type;
                          Its_Name : String; In_Prot_Obj : Boolean) is
   begin
      New_Line(Fichier);
      if Option_Exist(Self, "c") and
         Exist(Atts, "comment") then
         Put_Line(Fichier, "Commentaire : " &
                           Value_Of(Atts,"comment"));
      end if;
      Put_Line(Fichier, Its_Name & " une procedure...");
   end A_Procedure;
...
```

NB : A comment is a string of character with a ¶ symbol to mark a new line in the attributes' dictionary. You certainly need to replace this symbol (for example in HTML by a < BR >).

2.7 Adding the module in the main program

We need to make four modifications to the main program to add correctly our xy module into AdaDoc. The execution of our module depends on an option in the main program.

2.7.1 Adding the context clause

We are beginning by adding the context clause of our module.

```
...
with Module.Xy; -- Notre module
...
procedure AdaDoc is
...
```

2.7.2 Options type

We need to add an option in the enumerate type **Option_Name**. We call our module **Example**

```
...
   type Option_Name is (Delete_Xml,
                        Extract_Comment,
                        No_HTML,
                        HTML,
                        LaTeX,
                        Exemple, -- our module
                        Show_Help);
...
```

2.7.3 Options' array

Now we add our module in the options' array. This array contains:

- The name of the option (of type **Option_Name**).
- A boolean variable, to know if an action is active by default.
- The main switch of the option. If both characters (here "E") are entering by the user when executing adadoc, the option is active. All the following characters are sent as option of the module (in our module "-Ec" will active the print of the comments)
- *Options* is normally empty but you can use it to force an option.

```
...
   -- Option tab with stat, switch, comment, default options
   Option_Tab   : Option_Tab_Type :=
   (
   Delete_Xml => (False, Switch   => S("-x"),
                         ...
                         Options => S("")),

   -- Debut de l'ajout de notre module
   Exemple     => (False, Switch   => S("-E{a-z}"),
                         Comment => S("Ici notre description."),
                         Options => S("")),
   -- Fin de l'ajout de notre module

   Show_Help   => (False, Switch   => S("-h"),
                         ...
                         Options => S(""))
   );
...
```

NB : Switches are case sensitive.

2.7.4 The processing

Finally we add in Module section of adadoc.adb the instructions to run our module.

```
...
      Modules : declare
         Use_Default : Boolean := True;
      begin
      ...
         if Option_Tab(Exemple).Stat then -- si lé'tat est active
            Parse_XML(XML_File_Name.All,
                      new Module.Xy.Object, -- Instance de notre module
                      Dict_Tags,
                      Option_Tab(Exemple).Options.All); -- Ses options
            Use_Default := False;
         end if;
      ...
      end Modules;
...
```

2.8 Compilation

Now you have to compile AdaDoc. Use the make batch file for Windows system and the makelinux script for Linux system.

3 Feedback

You can send us your module at erreur@users.sourceforge.net and we will add then to the AdaDoc distribution.

4 Code source of the example module

4.1 module-xy.ads

```
--
-- @filename Module.Xy.ads
-- @author Julien Burdy & Vincent Decorges
-- @date 7.7.02
-- @brief Module exemple du document "Comment ecrire un module."
--
-- History :
-- Date       Modification                Author
--
package Module.Xy is

   -- L'objet derive.
   type Object is new Module.Object with null record;

   -- Annonce du debut du document.
   -- Cet evenement est appele une seule fois.
   procedure Begin_Document (Self : Object; Package_Name : String;
                             Atts, Tags : Dict_Type);

   -- Annonce du la fin du document.
   -- Cet evenement est appele une seule fois.
   procedure End_Document (self : Object);

   -- Annonce des édbut de groupes (ou d'Element_Type en general).
   procedure Start_Group (Self : Object; Group : Element_Type);

   -- Annonce d'une procedure.
   procedure A_Parameter (Self : Object; Atts : Dict_Type; Its_Name,
                          Its_Type, Its_Mode : String);

   -- Annonce d'un parametre.
   procedure A_Procedure (Self : Object; Atts : Dict_Type;
                          Its_Name : String; In_Prot_Obj : Boolean);

end Module.Xy;
```

4.2 module-xy.adb

```
with Ada.Text_IO; use Ada.Text_IO;
package body Module.Xy is

   Fichier : File_Type; -- Declaration de notre fichier

   procedure Begin_Document (Self : Object; Package_Name : String;
                             Atts, Tags : Dict_Type) is
   begin
      Create(Fichier, Name => Package_Name & ".txt");
      Put_Line(Fichier, "Paquetage : " & Package_Name);
   end Begin_Document;

   procedure End_Document (self : Object) is
   begin
      New_Line(Fichier);
      Put_Line(Fichier, "Fin.");
      Close(Fichier);
   end End_Document;


   procedure A_Procedure (Self : Object; Atts : Dict_Type;
                          Its_Name : String; In_Prot_Obj : Boolean) is
   begin
      New_Line(Fichier);
      if Option_Exist(Self, "c") and
         Exist(Atts, "comment") then
         Put_Line(Fichier, "Commentaire : " &
                           Value_Of(Atts,"comment"));
      end if;
      Put_Line(Fichier, Its_Name & " une procedure...");
   end A_Procedure;


   procedure A_Parameter (Self : Object; Atts : Dict_Type; Its_Name,
                          Its_Type, Its_Mode : String) is
   begin
      Put_Line(Fichier, Its_Name & " est de type " & Its_Type &
               " et de mode de passage " & Its_Mode);
   end A_Parameter;

   procedure Start_Group (Self : Object; Group : Element_Type) is
   begin
      case Group is
         when Is_Subprograms =>
            Put_Line(Fichier, "Voici la liste des sous programmes");
            Put_Line(Fichier, "----------------------------------");
         when Is_Parameters =>
            Put_Line(Fichier, "Liste des parametres :");
         when others =>
            null;
      end case;
   end Start_Group;

end Module.Xy;
```

4.3 Result file of the example module (Module.Xy.txt)

```
Paquetage : Module.Xy
Voici la liste des sous programmes
----------------------------------

Commentaire : Annonce du debut du document.¶ Cet evenement est appele une seule fois.
Begin_Document une procedure...
Liste des parametres :
Tags est de type Dict_Type et de mode de passage in
Atts est de type Dict_Type et de mode de passage in
Package_Name est de type String et de mode de passage in
Self est de type Object et de mode de passage in

Commentaire : Annonce du la fin du document.¶ Cet evenement est appele une seule fois.
End_Document une procedure...
Liste des parametres :
self est de type Object et de mode de passage in

Commentaire : Annonce des début de groupes (ou d’Element_Type en general).
Start_Group une procedure...
Liste des parametres :
Group est de type Element_Type et de mode de passage in
Self est de type Object et de mode de passage in

Commentaire : Annonce d’une procedure.
A_Parameter une procedure...
Liste des parametres :
Its_Mode est de type String et de mode de passage in
Its_Type est de type String et de mode de passage in
Its_Name est de type String et de mode de passage in
Atts est de type Dict_Type et de mode de passage in
Self est de type Object et de mode de passage in

Commentaire : Annonce d’un parametre.
A_Procedure une procedure...
Liste des parametres :
In_Prot_Obj est de type Boolean et de mode de passage in
Its_Name est de type String et de mode de passage in
Atts est de type Dict_Type et de mode de passage in
Self est de type Object et de mode de passage in

Fin.
```