

IBM Ada/6000

SYNOPSIS

This chapter describes how to compile, bind, and execute IBM Ada/6000 programs as well as how to maintain IBM Ada families and sublibraries on IBM RISC/6000 systems. Note that IBM Ada/6000 is different from Alslys Ada on the IBM RISC/6000 systems; Alslys Ada is covered in a previous chapter.

Note also that Alslys Ada and IBM Ada/6000 use some of the same naming conventions for important files (e.g., "adal i b") which could create conflicts if the compilers are used in the same account. These conflicts can be avoided by making a different working directory for each compiler.

SPECIAL SITE DEPENDENT NOTE	The IBM Ada/6000 compiler exists only on IBM RISC/6000 systems. For a list of these systems, please refer to the "UNIX Systems Available at Cal Poly" chapter in the "Introduction to UNIX" User Guide.
SPECIAL USAGE NOTES	<p>This document describes how to customize your account and maintain IBM Ada/6000 in your account. The Computer Science department also has some procedures which install IBM Ada/6000 specific files in your account on IBM RISC/6000 systems on campus. Please consult your instructor before selecting the proper procedure for using IBM Ada if you are going to be using it for a Computer Science course.</p> <p>This document will not address the specific instructions used in the Computer Science handout.</p> <p>NOTE: In most of the following commands, the term "<i>sublibrary</i>" refers to the Ada library within your account which is usually named "adal i b".</p>
INSTALLING IBM ADA IN YOUR ACCOUNT	You will not have to do anything to install IBM Ada in your account. IBM Ada is installed on the system and is ready for you to use.
CREATE AN IBM ADA SUBLIBRARY	<p>The following command will create an IBM Ada sublibrary named "adal i b" in the current working directory. This must be accomplished before you can compile an IBM Ada program.</p> <pre>% al i b i n i t <CR></pre> <p>The sublibrary is created along with a file named "al i b. l i s t" in the current working directory.</p>
COMPILING IBM ADA SOURCE CODE INTO EXECUTABLE PROGRAMS	<p>With the IBM Ada compiler, a compile and link may occur as a result of a single command. To compile and link an IBM Ada source program enter</p> <pre>% ada [options] [filenames]<CR></pre> <p>where "<i>filenames</i>" are the files to be compiled and where "<i>options</i>" include the following</p> <p>Options for Compiler Output</p> <pre>- v</pre> <p>Display all messages (defaults to not display a copyright notice or progress messages).</p>

- o** *name* Name the executable name (defaults to "a.out").
- l** Produce listing files containing lines from the source files interspersed with any errors the compiler finds. The listing files have a suffix of ".lst". The default is no error listing.
- G** Optimize code (defaults to no optimization).
- O** Highly optimize code (defaults to no optimization). Do not include both **-G** and **-O** as **-O** includes the optimization that **-G** performs.
- p** Produce code that records profile information at run time. This option must be specified before each source unit, whether or not it produces an executable file.

Options for Linking

- b** *unit_name* Produce an executable file using "*unit_name*" as the main program unit. Ada source files are not required with this option when referring to a unit already compiled into the sublibrary.
- m** Compile source files and produce an executable file. The main unit of the program is the last unit in the last source file you specify (defaults to not producing an executable). **NOTE:** This is only used when you are compiling the main routine and all other supporting packages already exist in your ada library.
- e** Produce an object module if either **-b** or **-m** are specified (defaults to producing an executable file if either **-b** or **-m** are specified).
- i** *filename* During linking, include the specified object module (defaults to the compiler deciding which object modules to include).

Options for Libraries

- L** *library_list_file* Use the file "*library_list_file*" as the library list file (defaults to using "alib.lst").
- u** Unlock the working sublibrary so that the compiler can access and update it (defaults to stopping the compile if another compilation stopped without finishing and left the sublibrary inaccessible).
- d** Store information for debugging in the current working directory (defaults to not storing debugging information).

Then to run your program compiled with the "-e" option or linked with the "-b" option, type:

```
% a.out<CR>
```

To execute your program where "a.out" is the name of the executable created by the compiler.

SAMPLE COMPILE AND EXECUTION SESSION

The source being used in this sample is located in the file "hello.ada" and is of the form

A. EXAMPLE WITH TEXT

```
with Text_IO;
procedure Test is
begin
    Text_IO.Put_Line("Hello world!");
end Test;
```

This source may be created in an editor such as "vi" on the AIX system or may be created on another system and transferred to your AIX account.

Before any compilation can take place, we must first initialize the Ada sublibrary, "adali b". This only needs to be done once; the Ada sublibrary should be located in the sub-directory where your Ada compilations will take place. The initialization is completed by entering

```
% alibinit<CR>
```

It is possible to have more than one Ada sublibrary. The amount of disk space this consumes usually makes this unwise in the average user account. Instead the user should select a sub-directory where the majority of ada work will be performed.

Once the file has been created, you may wish to perform an automated syntax check before compiling. This uses less CPU time than a regular compilation. To check the syntax, enter

```
% asyntax hello.ada<CR>
```

When there are no errors, the "asyntax" command responds with

```
hello.ada: Okay.
```

You are now ready to perform a compile and bind in one command by entering

```
% ada -vl hello.ada<CR>
% ada -vb test -o hello<CR>
```

where the "-vl" options instruct the compiler to produce messages on the screen as well as a listing file (*filename.lst*) which contains the error messages relative to the code lines that caused the error. The "-vb" options instruct the linker to produce messages on the screen and bind the program for execution, and the "-o hello" option instructs the linker to produce an executable file named "hello" instead of the usual "a.out".

The program can then be executed and produces its output as follows:

```
% hello<CR>
Hello world!
%
```

B. EXAMPLE USING MATH

Another common example is using multiple units with the math library. Assume that the following source code

```
WITH Text_IO;    -- predefined
WITH My_Long_Float_IO;    -- User defined and in an accessible library
                        -- (NOTE: Pre-compile math_IO.ada
                        --      and check for package in areport.)
WITH Math;      -- IBM Ada/6000 Library
PROCEDURE Test_Math IS

    Area:    Long_Float;
    Radius: Long_Float;
    Pi:      CONSTANT Long_Float :=
                Long_Float( Math.M_Pi );

    USE Math;

BEGIN
    Text_IO.Put ( Item => "Radius is ");
    My_Long_Float_IO.Get ( Item => Radius );
    Area := Pi * Radius**2.0 ; -- Using exponentiation
                                --from math package
    Text_IO.Put ( Item => "Area is ");
    My_Long_Float_IO.Put ( Item => Area, Fore => 1,
                            Aft => 2, Exp => 0 );
    Text_IO.New_line;
END Test_Math;
```

is the file "math.ada". And that the following source code

```
WITH Text_IO;
PACKAGE My_Long_Float_IO IS
    NEW Text_IO.Float_IO(Num => Long_Float);
```

is the file "math_IO.ada".

We can compile and bind the units in the following steps:

```
% ada -vl math_IO.ada<CR>
% ada -vl math.ada<CR>
% ada -vb test_math<CR>
```

The results of these steps will be an executable file named "a.out", that when executed, produces the following

```
% a.out<CR>
Radius is 2.99<CR>
Area is 28.09
%
```

C. EXAMPLE USING MULTIPLE PACKAGES

Multiple packages may be compiled and bound using the following example where the ada source files are named "file1.ada", "file2.ada", "file3.ada", and "file4.ada". Any of the units may be compiled separately, using the example from section A above by using the command

```
% ada -vl filename<CR>    where "filename" is the name of one of the
                           ada source files.
```

Once all of the units have been compiled, the main unit may be bound with the following command

```
% ada -vb main_unit_name -o executable_file_name<CR>
```

This assumes that the main unit has the proper syntax ada statements to call the supporting packages.

MAINTAINING YOUR ADA SUBLIBRARIES

IBM Ada has several commands that are used for the maintenance of IBM Ada sublibraries. For some of the more common tasks the commands are

A. LISTING UNITS IN YOUR SUBLIBRARY

You may list the units contained in your sublibrary by issuing

```
% areport [-B][ -y][ -s sublibrary]<CR>
```

where "*B*" indicates a brief list of names is requested; "*y*" indicates that system library information and any related sublibrary information should also be listed; "*s*" indicates that all parameters following it until another flag or the end of the line are sublibrary names.

This report is useful for determining the unit name, used in the binding step, as well as the names needed for the context clauses of compilation units that use the pre-compiled packages listed in the report.

Using the *adalib* sublibrary from the sample session above, we issue the "*areport*" command and get the following results:

```
% areport<CR>
-- Compilation unit name      -- (Type)      Date      Time      Sublibrary
lib/test                      -- (Prg_S)    1993-Jan-20 08:56:22 /local/home/u7/jdoe/adalib
sec/test                      -- (Prg_B)    1993-Jan-20 08:56:22 /local/home/u7/jdoe/adalib
%
```

B. USING YOUR SUBLIBRARY OUTSIDE THE DIRECTORY ITS IN

Part of your ada library creation is a file at the same location called "*alib.list*". An example of one that gets created when you issue the "*alibinit*" command follows:

```
----- Ada Library List File -----

-- The working sublibrary:
/path/to/your/adalib

-- Other Ada sublibraries:
```

As long as the line after the "*-- The working sublibrary:*" indicates the correct path to your library, you can place a copy of this file in any directory in which you wish to perform compiles. Copying this file (*alib.list*) to any other directory, prepares that directory for becoming a compilation directory using the original ada library.

C. INCLUDING OTHER LIBRARIES IN YOURS

In addition, you may also use the "*alib.list*" file to point to other libraries. The following example "*alib.list*" file points to several additional libraries that

are installed on the system and may be used to point to any IBM Ada library that has been made available on the system:

```
----- Ada Library List File -----
```

```
-- The working sublibrary:  
/path/to/your/adalib
```

```
-- Other Ada sublibraries:  
/usr/lpp/ada/lib/math  
/usr/lpp/ada/lib/nls  
/usr/lpp/ada/lib/X11  
/usr/lpp/ada/lib/xgsl
```

To view the modules in a single library enter:

```
% areport -s libname<CR>
```

where "*libname*" is the name of the library

The library name specified by "*libname*" may either be just the filename of the library ("*adalib*", "*math*", "*nls*", "*X11*", or "*xgsl*" in the above example) or it may specify a library by a fully qualified or relative path (e.g., "*/path/to/your/adalib*").

D. REMOVING UNITS FROM YOUR SUBLIBRARY

To remove a unit from an existing sublibrary enter

```
% aunitrn [-f] sublibrary unit_name<CR>
```

where "*-f*" indicates that the "*unit_name*" refers to the actual file name instead of compilation unit names, the **required** parameter "*sublibrary*" is the name of your Ada sublibrary (usually "*adalib*"), and "*unit_name*" refers to the compilation unit name unless "*-f*" is specified. (**NOTE:** Prefix each name with "*lib/*" or "*sec/*".)

Using the *adalib* sublibrary from the sample session above, we issue the "*aunitrn*" command and get the following results:

```
% aunitrn adalib sec/test<CR>  
%
```

E. REMOVING YOUR SUBLIBRARY

To remove your existing sublibrary enter

```
% alibrn [-F] sublibrary<CR>
```

where "*-F*" indicates that if the sublibrary exists, it should be deleted without prompting; the required parameter "*sublibrary*" is the name of your Ada sublibrary (usually "*adalib*"). If you do not specify the "*-F*" parameter, *alibrn* will prompt you with

```
alibrn: Do you want to remove sublibrary sublibrary?
```

Any answer containing leading or trailing blanks and the string "*y*", "*ye*", or "*yes*" will result in the sublibrary being removed.

Using the *adalib* sublibrary from the sample session above, we issue the "*alibrn*" command and get the following results:

```
% alibrm adalib<CR>
alibrm: Do you want to remove sublibrary adalib? yes<CR>
%
```

F. UNLOCKING YOUR SUBLIBRARY

If your sublibrary becomes locked you may unlock it by entering

```
% ada -u [options] [filenames]<CR>
```

Please refer to section 1 for a description of the options for the *ada* command.

DEBUGGING IBM ADA PROGRAMS

There are two basic places that a user may have problems debugging an IBM Ada program. The first is during the compile phase, and the second is during execution.

A. ERRORS DURING A COMPILE OR BIND

If the compiler aborts during the compilation with an obscure message, the user can usually perform several steps to isolate the problem. Usually, these errors will also lock your Ada sublibrary. Please refer to section 7 on how to unlock your Ada sublibrary.

1. CHECK YOUR ACCOUNT FOR FREE DISK SPACE.

IBM Ada requires a fairly large amount of free disk space in your account in order to build and maintain your sublibrary and executables. Information Technology Services recommends that you try to maintain approximately 200 K bytes of free disk space.

If you are getting an error during either the compile or bind process that isn't clear, check your disk space by entering

```
% quota -v<CR>
```

The command will return how much disk space is in use, the total amount available for you to use, and, if any of the limits are exceeded, what the remaining warnings or grace period is. (On AIX, you need a report of disk space on your home site where your files are located; for example "*on cymbal quota -v*<CR>" would provide you with information on your files if they are located on the AIX site named cymbal.)

If you are over quota or are very close to being over (within 250 K bytes), you should consider deleting files from your account that you no longer need.

NOTE: Creating multiple *adalib*'s in different sub-directories will generally cause the average user to exceed their disk quota. Consider using only a single *adalib* sublibrary or use the "*-p value*" described with the "*-h*" flag with "*alibinit*" to limit the size of the sublibrary.

2. OBSCURE ERRORS DURING COMPILATION

Sometimes, compilers can become confused when compiling a program with strange combinations of syntax errors. If this occurs, you can make a syntax check of only your source code by using the *asyntax* command. To do so, enter

```
% asyntax myprog<CR>
```

where "*myprog*" is the name of your source Ada program. This will cause any syntax error to be displayed without performing a full compilation.

B. ERRORS OCCUR DURING EXECUTION OF YOUR PROGRAM

When an error occurs during execution that generates an obscure execution message, IBM Ada's "*adbg*" can be used to determine the problem. To compile, link, and execute your program under IBM Ada's "*adbg*" enter

```
% ada -lv mysource<CR>  
% ada -bv -o program -d mysource compilation_unit<CR>
```

This compiles the program "*mysource*" with debugging turned on and links the ada program producing an executable file named "*program*". Then enter

```
% adbg [-x program] compilation_unit<CR>
```

If the executable filename "*program*" and the main unit "*compilation_unit*" are the same name, "*-x program*" may be omitted. *adbg* will then prompt you and you must enter the "*run*" sub-command to start execution as follows

```
Debug> run<CR>
```

After observing the behavior of the program, you exit "*adbg*" by entering

```
Debug> exit<CR>
```

Additional help on "*adbg*" is available by typing "*help*" at the *adbg* "Debug> " prompt.

COMMON PROBLEMS

There are a couple of common problems that may occur while using IBM Ada. These are:

A. Unable to open the sublibrary *sublibrary*

Make sure you spelled the sublibrary name correctly. Make sure you have read permissions for the sublibrary. If all else fails, you may have to use the "*alibi*" command to initialize the sublibrary and recompile the units it held.

B. Unable to restore sublibrary *sublibrary*

Try using the compiler with the "*-u*" option to restore the library. If that fails, use the "*alibi*" command to initialize the sublibrary and recompile the units it held.

C. You are unable to use "*alibi*" to initialize your sublibrary "*adali b*".

Check to see if "*adali b*" already exists in the current working directory. If so, remove it with the system command

```
% rm -fr adali b<CR>
```

Once it has been removed, you should be able to use "*alibi*".

IBM ADA COMMAND
SUMMARY

The following is a summary of the IBM Ada/6000 commands and their uses. For a complete description of their parameters, enter

```
% ada_command_name -h<CR>
```

at the system prompt. Some of the common parameters are:

<i>-b unitname</i>	Specifies the main unit name for binding.
<i>-h</i>	Provide help on the command used.
<i>-f</i>	Indicates that unit names are source file names instead of compilation units.
<i>-F</i>	Perform change without prompting for confirmation.
<i>-I</i>	Read a list of sublibrary names from standard input.
<i>-L librarylistfile</i>	Provides an alternate to the default library list file "alib.list".
<i>-o objectname</i>	object output file name to be used instead of default.
<i>-v</i>	Display all messages (verbose output).

The commands are also broken down into their various categories starting with the compiler, and then going on to the unit manipulation, library manipulation, program development, and debugging commands.

The compiler command:

```
ada [-h] [-v] [-o name] [-l] [-a] [-G | -O] [-p] [-qcomment=file]  
[-b unit] [-m] [-e] [-i file] [-L librarylist] [-u] [-d] [-I]  
[-s] [-q time_slice|task_stack|stack_limit|heap_limit=n]  
[-q show_task_exceptions=choice] [-V number] [filename(s)]<CR>  
Compile ada source programs and/or produce executable files.
```

Commands affecting contents of sublibraries:

```
afilemv [-h] [-F] sublibrary current_file_name new_file_name<CR>  
Renames a compiled source file in "sublibrary" from "current_file_name" to "new_file_name".
```

```
alibinit [-h] [-F] [-L librarylistfile] [-p value]<CR>  
Initializes a Ada sublibrary.
```

```
alibchk [-h] [-l] [-w] [-L librarylistfile|sublibraries|-I]<CR>  
Verifies the contents of Ada sublibraries for missing components, extraneous code, and locks.
```

Commands affecting sublibraries:

```
alibmv [-h] [-F] sourcesublibrary targetsublibrary<CR>  
Renames a sublibrary or moves it from one directory to another.
```

```
alibcp [-h] [-F] sourcesublibrary targetsublibrary<CR>  
Copies a sublibrary to another location.
```

alibrm [-h] [-F] *sublibraries*|-I<CR>
Deletes an entire sublibrary.

aunitmv [-h] [-f] *sourcesublibrary targetsublibrary*
unit_name(s)|-I<CR>
Moves one or more units from one sublibrary to another.

aunitcp [-h] [-f] *sourcesublibrary targetsublibrary*
unit_name(s)|-I<CR>
Copies one or more units from one sublibrary to another.

aunitrm [-h] [-f] *sublibrary* *unit_name(s)*|-I<CR>
Removes one or more units from a sublibrary.

areport [-h] [-B] [-y] [-L *librarylistfile*]
[[-s *sublibraries*|-I]|-I]
[-u *unit_name*|-I] | [-f *filename*|-I] <CR>
Generates reports that display different types of information about compiled units and sublibraries.

Commands for Program Development:

afmt [-i *convention*] [-r *convention*] [[-f *controlfile* [-N]] | [[-k] [-n *indentation*] [-w *linewidth*]]] [-h *filename*] -I<CR>
Formats Ada source code to given indentation, line length, and case.

amakedep [-h] *sourcefile(s)*|-I [-F] [[-a] | [-L *librarylistfile*]]
[-i [*lib*|*all*]] [-g [*lib*|*all*]] [-f *makefilename*]
[-b *mainunitname(s)*] [-o *objectname(s)*] <CR>
Invokes AIX "make" command to build your Ada application.

aprof [-b] [-e *name*] [-E *name*] [-f *name*] [-F *name*] [-L *pathname*]
[-s] [-z] [*a.out* [*gmon.out* ...]] <CR>
Run-time performance monitoring of your program by capturing and displaying information during execution.

asrcdep [-F] [-g] [-i]] [-h] [-v] [-c|-m] [-u *unitname*]
[*filename(s)*]|-I <CR>
Determines, from a given list of source files, a reasonable compilation order for those files based on dependencies created by context clauses.

asrcinfo [-h] *filename(s)*|-I <CR>
Provides information about one or more Ada files.

asyntax [-h] *filename(s)*|-I <CR>
Checks the syntax of one or more Ada source files.

axref [-L *librarylistfile*] [-h] [-w] *unitname(s)*|-I <CR>
Provides symbol cross-reference information for one or more Ada compilation units.

Command for debugging:

```
adbg [-h] [-L librarylistfile] [-V spacesize] [-x program] [-R]
[-f filename] compilationunit<CR>
```

Invokes the IBM Ada run-time debugger.

ADA ON-LINE
DOCUMENTATION

On-line Documentation is available with the IBM Ada Help option. To access the information, enter the IBM Ada command name followed by the "-h" flag. For example

```
% ada -h<CR>
```

returns a list of options that may be used with the "ada" command.

There is also a brief man page on IBM Ada which is accessed by entering

```
% man ada<CR>
```

DOCUMENT CODE: UNIX-40105C

DATE REVISED: September 7, 1995

NOTES