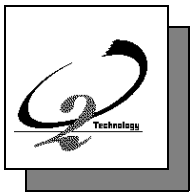


O_2 Makegen
User Manual

Release 5.0 - April 1998

Information in this document is subject to change without notice and should not be construed as a commitment by O_2 Technology.

The software described in this document is delivered under a license or nondisclosure agreement.

The software can only be used or copied in accordance with the terms of the agreement. It is against the law to copy this software to magnetic tape, disk, or any other medium for any purpose other than the purchaser's own use.

O_2, O_2Engine API, O_2C, O_2DBAccess, O_2Engine, O_2Graph, O_2Kit, O_2Look, O_2Store, O_2Tools, and O_2Web are registered trademarks of O_2 Technology.

SQL and AIX are registered trademarks of International Business Machines Corporation.

Sun, SunOS, and SOLARIS are registered trademarks of Sun Microsystems, Inc.

X Window System is a registered trademark of the Massachusetts Institute of Technology.

Unix is a registered trademark of Unix System Laboratories, Inc.

HPUX is a registered trademark of Hewlett-Packard Company.

BOSX is a registered trademark of Bull S.A.

IRIX is a registered trademark of Siemens Nixdorf, A.G.

NeXTStep is a registered trademark of the NeXT Computer, Inc.

Purify, Quantify are registered trademarks of Pure Software Inc.

Windows is a registered trademark of Microsoft Corporation.

All other company or product names quoted are trademarks or registered trademarks of their respective trademark holders.

Who should read this manual

This manual describes how to build O_2 applications by creating makefiles which invoke O_2 and system tools. O_2Makegen generates a platform dependent makefile from platform independent information stored in a configuration file. O_2Makegen may be utilized for building applications using C++, O_2C, C and O_2Engine API.

Other documents available are outlined, click below.

See O2 Documentation set.

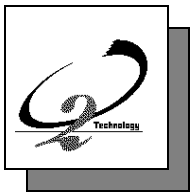


TABLE OF CONTENTS

This manual is divided into the following chapters:

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

1 The Build Process

GENERAL OVERVIEW OF THE O_2MAKEGEN TOOL

Congratulations! You are now a user of the O_2Makegen tool!

This chapter gives an overview of the O_2 system and describes the different steps in binding together the layers of an O_2 application to build an executable.

The chapter is divided into the following sections:

1 The Build Process

1.1 System Overview

The system architecture of O_2 is illustrated in Figure 1.1.

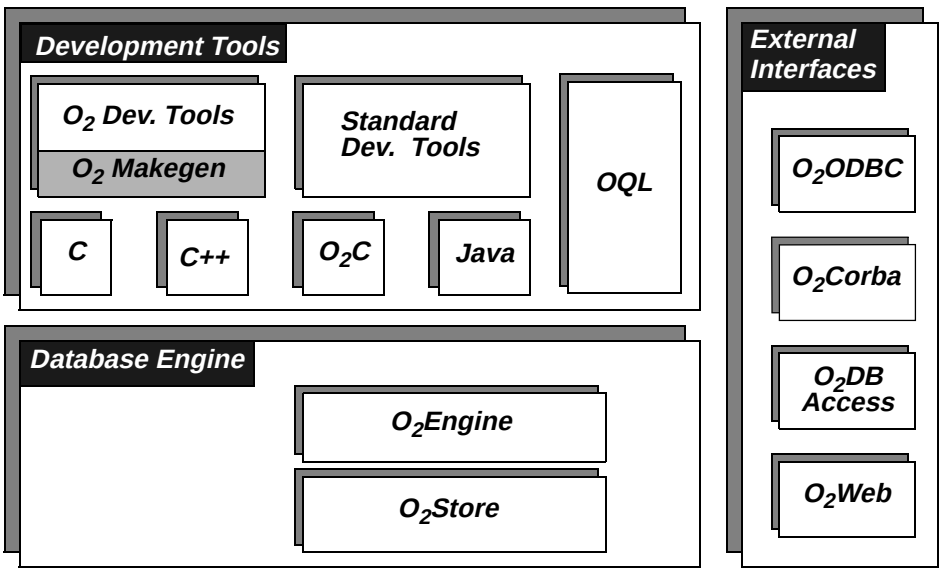


Figure 1.1: O_2 System Architecture

The O_2 system can be viewed as consisting of three components. The Database Engine provides all the features of a Database system and an object-oriented system. This engine is accessed with Development Tools, such as various programming languages, O_2 development tools and any standard development tool. Numerous External Interfaces are provided. All encompassing, O_2 is a versatile, portable, distributed, high-performance dynamic object-oriented database system.

Database Engine:

- O_2Store — The database management system provides low level facilities, through O_2Store API, to access and manage a database: disk volumes, files, records, indices and transactions.
- O_2Engine — The object database engine provides direct control of schemas, classes, objects and transactions, through O_2Engine API. It provides full text indexing and search capabilities with O_2Search and spatial indexing and retrieval capabilities with O_2Spatial. It includes a Notification manager for informing other clients connected to the same O_2 server that an event has occurred, a Version manager for handling multiple object versions and a Replication API for synchronizing multiple copies of an O_2 system.

Programming Languages:

O_2 objects may be created and managed using the following programming languages, utilizing all the features available with O_2 (persistence, collection management, transaction management, OQL queries, etc.)

- C — O_2 functions can be invoked by C programs.
- C++ — ODMG compliant C++ binding.
- Java — ODMG compliant Java binding.
- O2C — A powerful and elegant object-oriented fourth generation language specialized for easy development of object database applications.
- OQL — ODMG standard, easy-to-use SQL-like object query language with special features for dealing with complex O_2 objects and methods.

O_2 Development Tools:

- O_2Graph — Create, modify and edit any type of object graph.
- O_2Look — Design and develop graphical user interfaces, provides interactive manipulation of complex and multimedia objects.
- O_2Kit — Library of predefined classes and methods for faster development of user applications.
- O_2Tools — Complete graphical programming environment to design and develop O_2 database applications.

Standard Development Tools:

All standard programming languages can be used with standard environments (e.g. Visual C++, Sun Sparcworks).

External Interfaces:

- O_2Corba — Create an O_2/Orbix server to access an O_2 database with CORBA.
- O_2DBAccess — Connect O_2 applications to relational databases on remote hosts and invoke SQL statements.
- O_2ODBC — Connect remote ODBC client applications to O_2 databases.
- O_2Web — Create an O_2 World Wide Web server to access an O_2 database through the internet network.

1.2 The O_2Makegen tool

Most software are built using the same underlying procedures:

- obtain user source code files
- create automatically generated source code files (if any)
- use compiler to convert source code to machine code
- supply any additional compiled objects and libraries
- use a linker to create an executable
- install executable and other files needed at run-time.

There are many ways of building an application. Software development environments vary widely. Tools that support the build process, such as O_2Makegen, must therefore be highly flexible to allow for the largest range of situations.

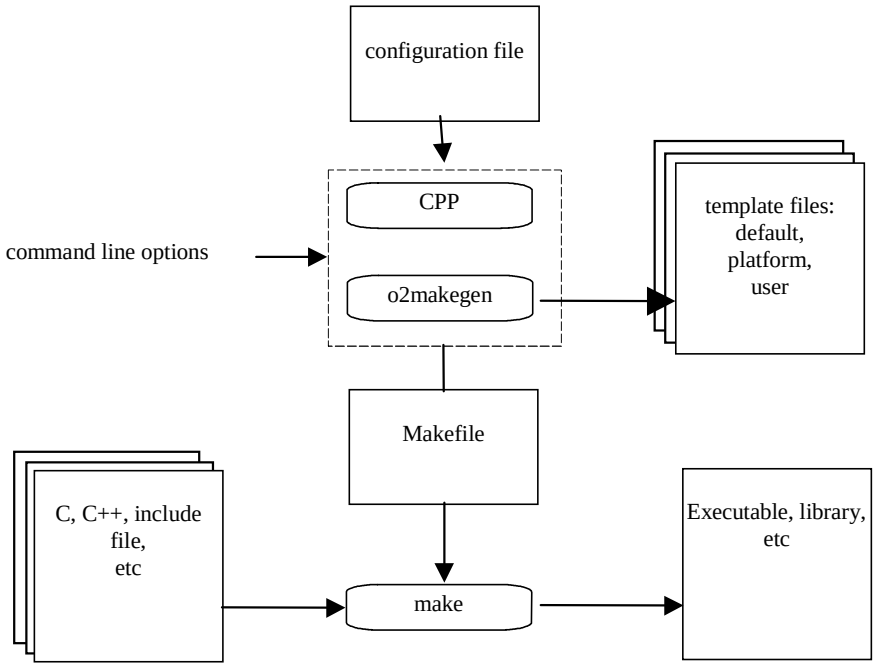
On UNIX and MS-Windows systems, application building is supported by the "make" utility, which uses input file known as "makefile" to invoke system utilities like compiler and linker to build application. The "make" utility contains logic to minimize the steps needed to build applications by avoiding unnecessary steps.

Makefiles contain entries known as "targets", which in turn have "rules" which are lists of commands to execute to construct the target. Each target has one or more "dependencies", which can be other targets that must be executed first. By recursively executing dependencies, the "make" tool avoids unnecessary actions or compilations. Makefiles may contain macros, which are text symbols expanded when targets are executed. These macros allow for more compact and readable makefiles.

The O_2Makegen tool assists in building O_2 applications by creating makefiles that invoke O_2 and system tools to perform the build. O_2Makegen reduces the time spent creating makefiles by reducing the information you provide to control what "make" does.

A powerful feature of O_2Makegen is the capability to generate a platform-dependent makefile from platform-independent information stored in a configuration file. All information you supply is independent of the platform and O_2Makegen generates all platform-dependent stuff.

You do not have to worry about platform dependencies. O_2Makegen can be used to build applications using O_2 ODMG C++ binding, O_2C, C Interface to O_2, O_2Engine API runtime from O_2 Technology.



Part Name	Description
configuration file	provides application-dependent information such as the name of source files, header files which must be imported, libraries to link, ...
command line options	gives information about platforms, makefile name,...
cpp (C preprocessor)	preprocesses the configuration file.
default template file	gives the default for tools and commands used in the makefile (located in O2HOME/config)
platform template file	gives the tools and commands used in the makefile when different from the default (located in O2HOME/config)
user template file	gives the tools and commands used in the makefile for a particular installation when different from the default- and from the platform-specific ones.
O2Makegen tool	generates a makefile based on all input provided.
make tool	builds the targets specified in the generated makefile.

1.3 Using O_2Makegen

The model enforced by O_2Makegen is that within a directory, only one program, library or re-locatable object can be created. If you have multiple directories, you can define one makefile per directory, and a master makefile in the root directory which triggers the other sub-makefiles in the sub-directories.

When you run "make", you usually must tell it what to build. The generated makefile allows you to invoke several targets:

- Building And Installing An Application :

  These entry points are created in the makefile to build and install an application:

Targets	Description
all	Creates the executable or the library
install	Installs the executable or the library and the include files necessary when using the library

Whenever you run "make", all products of the build process are placed in the directory in which "make" is run.

- Removing Generated Files :

  These entry points are created in the makefile to remove generated files:

Targets	Description
clean	removes all the temporary files (e.g. object files) and the executable or library
clobber	performs clean, and removes the files generated by O_2 tools

- Generating Dependencies :

  The following entry point is created in the makefile to generate dependencies:

Targets	Description
depend	adds dependency rules to the makefile

  The default target (i.e. the one chosen when you run "make" without indicating a target) is "all".

  Not all entry points are created in the makefile:

- " all ", " clean ", " clobber " are always created
- " install " is created only if the destination is given in the configuration file
- " depend " is created only if all sources are given in the configuration file.

  Depending on the type of executable, some targets are added to the generated makefile. For example, when you build a O_2 C++ ODMG application, "import" and "unimport" targets are created to import and unimport C++ classes definitions in the O_2 database.

  You may wish to have additional actions performed by the generated makefile. This can be done by adding a user makefile which is invoked by the generated makefile.

  The makefile generated by O_2Makegen contains definitions related to the machine type, the operating system and its version. These definitions allow portions of your source code to be machine-dependent. They are passed to the compiler by means of -D flags.

  These definitions are described below:

  HP machines :

- DHP800
- DHPUX
- DHPUX_9x
- DHPUX_10x

IBM or Bull machines

- DIBMRS600
- DAIX
- DAIX_32
- DAIX_42

Sun machines

- DSPARC
- DSOLARIS
- DSOLARIS_2x
- DSUNOS
- DSUNOS_41

Silicon Graphic machines

- DSGI
- DIRIX
- DIRIX_5.x

Digital Alpha Machines

- DALPHA
- DOSF1

Intel machines

- DX86
- DSCO
- DSCO_32
- DSOLARIS
- DSOLARIS_2x
- DWIN32

x is the release number.

1.4 Invoking O_2Makegen

The syntax is the following:

```
o2makegen [options] configuration_file
```

where "options" are :

Invoking O2Makegen :

Name	Description
-help	displays a help facility and exits
-version	prints version number of o2makegen and exits
-verbose	enables verbose mode
-output *makefile_name*	specifies the name of the generated makefile (Make-file by default)
-arch *machine*	specifies the target architecture name
-os *os*	specifies the target operating system name
-deffile *file_name*	specifies the name of a private template file
-pushbacksize *int-size*	sets pushback and argument collection size (default 4096)

The default values for "arch" and "os" options are the architecture and the operating system on which o2makegen is running.

The "arch" option recognizes the following machine type:

SPARC

HP800

SGI

IBMRS6000

DECALPHA

RM

X86

The "os" option recognizes the values:

SUNOS_41

SOLARIS_2x

HPUX_9x

HPUX_10x

IRIX_5x

AIX_32

AIX_42

OSF1

SINIX_54

SCO_32
WINDOWS

(x is the release number).

1.4.1. External influences

You can use the TMPDIR environment variable to set the directory in which O_2Makegen creates temporary files.

On Sparc with SUNOS_41 O_2Makegen uses the Sun executable /usr/5bin/m4 available with the system V software installation option.

1.4.2. Return value

O_2Makegen exits with one of the following values:

0	if makefile generation is successful.
>0	if aborted due to processing errors.
<0	if makefile generation carried out but with warning messages.

The following chapter describes the configuration file contents in more details.

2 O_2Makegen Configuration File

This chapter is divided into the following sections:

2 O2Makegen Configuration File

2.1 Introduction

To build an application, you need to provide a configuration file to supply information to O_2Makegen to create a makefile. The configuration file contains all the necessary information to build your application.

This information includes the names of source files and precompiled objects, executables names,...

The configuration file can have any name, but generally the extension ".cf" is used.

The configuration file is used by O_2Makegen and specifies :

- ***Application source files (.C, .cc and .h)***
- ***Application library/object files (archive library, shared library and .o)***
- ***The library environment***
- ***Macro definitions and include directories***

The C preprocessor "cpp" is run on the configuration file to maximize flexibility. You can use "cpp" macros to customize your configuration file.

2.2 The configuration file syntax

A configuration file contains comment lines, blank lines and directives. A comment line is introduced by a semicolon (;) going to the end of the line. C comment syntax is also accepted.

A directive line contains one of a predefined set of options as the first word, followed by user supplied information as in:

```
ExpClasses= Person City
```

If the user-supplied information is a list of items, each item must be separated by one or more spaces.

Warning !

If the same label (first word of a directive) appears multiple times in the same file, only the last appearance is relevant. All other directives with the same label are ignored.

If the user-supplied information is an option, preceding the label with the plus sign (+) indicates that the option is active. The minus sign (-) indicates that the option is inactive. For example,

```
+Debug
```

indicates that the debug mode is active.

All options are disabled by default, except "O2C++Target"

As the C preprocessor is run on your configuration file before it is used, you can place "cpp" directives anywhere in the configuration file to achieve effects such as conditional building.

Spaces and tabs can be inserted freely between elements without altering semantics. Since the configuration file is line-oriented, line feeds cannot be inserted freely. A '\' followed by a new line is treated as a continuation line and permits to break a long line into many shorter lines.

Unprotected $ characters cause a search in the environment for the following name. In this way, $HOME is expanded to the string in the process environment. If $ characters are to be literal, protect the $ character using a back slash as in \$HOME.

2.3 Basic options

This section is divided up as follows :

2.3.1. Building an application

If you want to obtain an executable, you must use the following options:

Options	Description
ProgramName = executable_name	specifies the generation of the program called executable-name. (no default).
ProgramObjs = object files name list	specifies the object files needed to build the executable (no default).
ProgramLib = application libraries name list	specifies the application specific libraries that should be linked with the application (none by default). For a library libx.a or libx.so (sl), you must give in the name list only the part x (without the "lib" prefix and the suffix)
UserLdFlags = flags for the link editor	specifies flags added to the link command
ProgramLibDir = library directories list	specifies the directories containing the application specific libraries (none by default).

These options generate a target whose name is executable_name. This target is added as dependency to the "all" target which is the default target.

The action associated with this target is the link edition of all object files and libraries for obtaining the executable.

If the options Debug, Profile, Purify or Quantify are set, the name of the executable is changed and is postfixed by "_d", "_p", "_pure" and "_quant".

Warning !

O_2 runtime libraries must not appear in the "ProgramLib" list.

The following option is specific to AIX:

```
ExportLibFile = file name list
```

Export files are ASCII files identifying external symbols that are made available for another object executable to import.

See the AIX documentation set for more information.

2.3.2. Building a library

If you want to construct an archive library, you must use the following options:

Options	Description
LibName = library name	specifies library to build (no default).
UserLdFlags = flags for the link editor	specifies flags added to the link commands
Objs = LibObjs	specifies object files needed to build the library (no default). The object files generated from the source files by an o2cpp_import or o2cpp_export command must not be in this list.
+- CreateArchiveLib	specifies the type of library to be built. +CreateArchiveLib permits to build an archive library. Archive library is the default.
+-CreateSharedLib	specifies the type of library to be built. +CreateSharedLib permits to build a shared library. Indicating -CreateSharedLib builds an archive lib.

These options generate a target whose name is library_name. This target is added as dependency to the "all" target which is the default target.

Using +CreateSharedLib and +CreateArchiveLib constructs a shared library (and not a shared library and an archive library within one makefile). If none is used, an archive library is built by default.

The action associated with this target is the archival of all object files in a library.

2.3.3. Building a relocatable

If you want to construct a relocatable object, you must use the following options:

Options	Description
RelocatableName = relocatable_name	generation of a relocatable called relocatable_name (no default).

Options	Description
UserLdFlags = flags for the link editor	flags added to the link commands
RelocatableObjs = object_files	Object files needed to build the relocatable object.

These options generate a target whose name is relocatable_name. This target is added as dependency to the "all" target which is the default target.

The action associated with this target is the construction of one object from all object files.

2.3.4. Compiling sources

To compile your source files, you can supply the following information:

Options	Description
Define=string_list	adds or renames macro definitions to the compilation or link phase.None by default.
Undefine =string_list	removes macro definitions to the compilation or link phase. None by default.
Include=directory_list	adds directory containing files to be included at compilation or link phase. None by default.
Repositories=directory_list	list of directories used as a C++ template repository-during compilation or link phase. None by default.

2.3.5. Debugging

If you want to obtain an executable with the debug information, you must set:

```
+Debug
```

in the configuration file. The default value is false. The generated executable is postfixed by "_d". You can use your favorite debugger to run your application.

Using the following keyword

```
+Profile
```

permits to generate an executable or library which contains code for profiling (using the OS supplied profiler). The name of the executable or of the library is postfixed by "_p". The default value is false.

Warning !

If Debug and Profile are set, only Debug is relevant, Profile is ignored. The generated executable is postfixed by "_d".

Purify TM, a product of Pure Software, is a tool that you can use to track down memory leaks and errors in your application. When you build your application, set :

```
+Purify
+Debug
```

in the configuration file. The generated executable is postfixed by "_pure". When you run your executable, memory leakage and access errors are tracked down. For more information on Purify, see the Purify documentation set. The default value is false.

You can also use Quantify TM, another product of Pure Software. It is a tool that identifies the portions of your application that dominate its execution time. When you build your application, set :

```
+Quantify
+Debug
```

in the configuration file. The generated executable is postfixed by "_quant". After running your application, the profile of execution is displayed. For more information on Quantify, see the Quantify documentation set. The default value is false.

2.3.6. Computing dependencies

If you set the following in your configuration file:

Options	Description
Sources=source_files	source files making up the application or library

a "depend" target is generated. Launching "make depend" modifies the makefile and adds all dependencies between source and include files.

2.3.7. Installing an application

The following installation information can be set in the configuration file to install the generated program, or generated library elsewhere than in the current directory :

Options	Description
ProgramDestDir=directory	directory in which the generated program must be installed (none by default).
LibDestDir=directory	directory where the generated library must be installed (default none).
LibHeaders=file_name	file to be treated as the include file of the generated library (default none).
HeadersDestDir=directory	directory where all include files needed to use the generated library must be installed (none by default).

If this information is given, an "install" target is generated in the makefile.

2.3.8. Triggering makefiles in sub directories

The model enforced by o2makegen is that within a directory, only one program, library or re-locatable object can be created. If you have multiple directories, you can define one makefile per directory, and a master makefile in the root directory which triggers the other sub-makefiles.

This master makefile can be generated by giving the following information in the configuration file :

Options	Description
SubDirs=directory_list	list of directories in which a make must be triggered.

The generated makefile contains four targets :

Targets	Description
all	triggers "all" targets of all makefiles found in the sub-directories
clean	triggers "clean" targets of all makefiles found in the subdirectories
clobber	triggers "clobber" targets of all makefiles found in the subdirectories
install	triggers "install" targets of all makefiles found in the subdirectories

Choosing one of these targets generates a call to each makefile found in each listed directory with the chosen target. All submakefiles are executed once in the order listed.

2.3.9. Example

In a first directory named "Lib", this first configuration file will generate a makefile to construct the archive library "my_lib" from the object files "pragma.o" and "collection.o". These object files are created from C++ files "pragma.cc" and "collection.cc" by the C++ compiler.

```
LibName= my_lib
LibObjs= pragma.o \
         collection.o

Sources= pragma.cc \
         collection.cc
```

In a second directory named "Prog", this second configuration file will generate a Makefile to construct the executable "my_prog" from the object files "main.o", "o2connect.o" and "foo.o" and from the archive library "my_lib". These object files are created from C++ file "main..cc" by the C++ compiler and C files "o2connect.c" and "foo.c" by the C compiler.

```
ProgramName= my_prog
ProgramObjs= main.o o2connect.o foo.o
ProgramLib= my_lib
ProgramLibDir= ../Lib
```

```
Sources= main.cc o2connect.c foo.c
```

You can create a master makefile for these two directories in the parent directory :

```
SubDirs= Lib Prog
```

The generated makefile triggers first the makefile in the "Lib" directory to construct the "my_lib" library, and then the makefile in the "Prog" directory to construct the executable "my_prog" which uses the library "my_lib".

2.4 O_2 specific options

These options specify information about O_2 environment used to generate the makefile. These options must be valued if you want to use tools and/or libraries from the O_2 environment.

This includes the following items:

Options	Description
O2Home=directory	O_2 installation directory. Default can be positioned in a .site.cf file (see Customizing O_2Makegen section).

By default, O_2 applications are built using shared version of the libraries from O_2 runtime. For debugging or delivery purposes, you can use the archive versions of the O_2 runtime libraries by setting the following option:

Options	Description
+-Use ArchiveLib	specifies which archive versions of O_2 runtime libraries must be used in place of the shared libraries (default is false)

The following options indicate which runtime libraries should be used to build your application:

Options	Description	Runtime libraries
+-UseOql	indicates whether OQL is used or not (default is false)	o2sql
+-UseLook	indicates whether O_2Look is used or not (default is false)	o2look, Xm, Xt, X11
+-UseLkBrowser	indicates whether O_2Look browser editor is used or not (default is false)	o2look, o2look_browser, Xm, Xt, X11
+-UselkDialog	indicates whether O_2Look dialog box editor is used or not (default is false)	o2look, o2look_dialog, Xm, Xt, X11
+-UseLkGraph	indicates whether O_2Look graph editor is used or not (default is false)	o2look, o2look_graph, Xm, Xt, X11
+-UseLkPict	indicates whether O_2Look picture editor is used or not (default is false)	o2look, o2look_pict, Xm, Xt, X11
+-UseLkText	indicates whether O_2Look Text editor is used or not (default is false)	o2look, o2look_text, Xm, Xt, X11
+-UseO2xt	indicates whether O_2Xt is used or not (default is false)	o2xt
+-UseVersion	indicates whether O_2Version mechanism is used or not (default is false)	o2vm
+-UseMeta	indicates whether O_2 meta service is used or not (default is false)	o2compiler, o2syntax, o2cruntime.

Refer to the O_2Look documentation set for more information about specific editors and o2xts.

2.5 Building a C++ application

If you use the ODMG C++ binding, you can use **`o2makegen`** to create a makefile which allows you to invoke new targets:

- import
- unimport
- export
- unexport

These targets invoke the o2cpp_import, o2cpp_unimport, o2cpp_export and o2cpp_unexport tools.

These new targets are added to the dependency list of the target "all". So when you build your executable by calling **`make`**, these new targets are triggered if necessary.

All necessary O_2 runtime libraries are automatically added to your executable.

To create this type of makefile, you must set the following option in the configuration file :

```
±O2C++Target
```

To use the import and/or export tools from O_2, the makefile must know the following information :

Options	Description
O2Schema = schema name	O_2 schema where classes will be imported (no default)
O2System = system name	O_2 system in which the schema resides (no default)
O2Server = server name	O_2 server which must be used by import and export tools (no default)

To import C++ classes in O_2, you can use the following options in addition to the one described in the above sections :

Options	Description
ImpFiles = file_name(s)	C++ files where imported classes are defined (no default).
ImpList = type_name(s)	list of classes or atomic types to be imported.
ImpBag = type_name (s)	bag of classes or atomic types to be imported.
ImpSet = type_name(s)	set of classes or atomic types to be imported
ImpVarray = type_name(s)	varray of classes or atomic types to be imported.

For each file given in the "ImpFiles" options, you must give the following information :

Options	Description
[FileName]ImpClasses= class_name(s)	list of classes to be imported. The definition of this class is found in FileName (no default).
[FileName]ImpOutputDir= directory	path and directory in which C++ files are generated for the classes in FileName (current directory by default).
[FileName]ImpForwardClasses= class_name(s)	All classes forwarded during the processing of FileName (none by default).
[FileName][ClassName]ImpAsClass= class_name	Import class_name as ClassName (see *C++ Binding Guide*).
[FileName][ClassName]ImpForwardFile= file_name	Specifies the file where the forwarded class is defined (none by default).
[FileName][ClassName]ImpMemberFunc= function_member(s)	Member function(s) to be imported (none by default)
±[FileName][ClassName]ImpAllPublicMemberFunc	Imports all public member functions of the class (false by default).
+-[FileName][ClassName]ImpAccessPrivateMember	Automatically generates access methods for private members (false by default)
[FileName]ImpUseFiles= file_name(s)	Files which are required to parse FileName. For example, an include file not explicitly included in the FileName file (none by default).
[FileName]ImpUseDir= directory_name(s)	Directory path used to find files specified in [FileName]ImpUseFiles option (none by default).
±[FileName]ImpNoModification	Pointers to objects of an imported class are not modified inside FileName (by default they are transformed to persistent pointers).
[FileName]ImpLibClasses= class_name(s)	Classes belonging to an external C++ library and used as a superclass of an imported class (none by default).
±UseConfirmClasses	Indicates whether classes are automatically confirmed or not after importation in O_2. (Default is false).

2 O2Makegen Configuration File

To export O_2 classes, you must provide the following information:

Options	Description
ExpOutputDir=directory	Directory path in which C++ files are generated (current directory by default).
ExpClasses= class_name(s)	O_2 classes to be exported (no default)
±[ExpClassName]ExpType	Type structure of the ExpClassName is also made available to C++ (False by default).
±[ExpClassName]Exp-NoVirtual	Specify that the exported methods are not virtual (false by default)
[ExpClassName]Exp-Methods= method_name(s)	O_2C methods of class ExpClassName to be exported (none by default).

For more information, refer to the *ODMG C++ Binding* documentation set.

When building an ODMG C++ application, you can use the following options:

- UseMeta
- UseVersion
- UseOQL
- UseLook and all specific editors

Warning !

The object files generated from the source files created by an o2cpp_import or o2export command must not be in the ProgramObjs list nor in the LibObjs list.

The generated makefile contains 8 targets :

Options	Description
all	triggers export, import and finally creation of the executable or of the library
clean	removes all object, core, ...
clobber	triggers clean, unexport and unimport
install	installs executable or library with include files
export	exports all indicated O_2 classes
import	imports all indicated C++ classes
unexport	destroys code generated by export
unimport	destroys O_2 classes, generated code and removes patch of the C++ classes

Example:

```
+O2C++Target

ProgramName= odmg
ProgramObjs= main.o o2connect.o pragma.o \
             bag.o list.o set.o array.o \
             collection.o  collection_int.o
collection_real.o \
             collection_string.o

Sources= main.cc o2connect.c pragma.cc \
         bag.cc list.cc set.cc array.cc \
        collection.cc collection_int.cc collection_real.cc
\
         collection_string.cc

O2Home= $O2MK_HOME
O2System= $O2MK_SYSTEM
O2Server= $O2MK_HOST
O2Schema= odmg_s

+UseOql

ImpFiles= schema.hxx
[schema.hxx]ImpClasses= A B node subnode
```

```
ImpSet= A B node int char double "char*" d_String short
float
ImpList= A B node int char double "char*" d_String short
float
ImpBag= A B node int char double "char*" d_String short
float
ImpVarray= A B node int char double "char*" d_String short
float
```

To build the "odmg" executable, we must import the C++ classes A, B, node, subnode defined in the file schema.hxx. We also import collection (Set, List, Bag and Varray) for the type :

A, B, node, int, char, double, "char *", d_String, short, float

All are imported in the schema "odmg_s" in the system "$O2MK_SYSTEM" using the server "$O2MK_HOST". You can note that $O2MK_HOME, $O2MK_SYSTEM, and $O2MK_HOST are environment variables and are evaluated during the makefile creation.

Because the "Sources" options have a value, a "depend" target is created. When you type "make depend" all dependencies of all files given in "Sources" are added at the end of the makefile.

The program name is odmg and is composed of all objects found in the "ProgramObjs" option list.

The program uses the "oql_execute" service. So we set the option "+UseOql".

The '\' character is used to break a directive in multiple lines.

2.6 Building a C application

If you use the C interface to O_2 in your application, you can use o2makegen to generate a makefile to build your application. You must set the following option:

```
±O2LinkCTarget
```

For more information, refer to the *C Interface to O_2* documentation set.

When building a C application, you can use the following options:

- UseMeta
- UseVersion
- UseOQL
- UseLook and all specific editors

Options	Description
all	constructs the application
clean	removes all objects, core,...
install	installs the application

2.7 Building an O_2C application

If you want to build an executable from an O_2C application, you must write a C++ file containing the main procedure of your application. This main procedure must make a connection to an O_2 server, trigger the O_2C application using the "o2_run_application" service, and at the end disconnect from the server.

To build your application, you can generate a makefile using the following option:

```
±O2CTarget
```

Do not forget to supply the name of the main object in the ProgramObj option.

For more information, refer to the *O_2C* documentation set.

When building an O_2C application, you can use the following options:

- UseMeta
- UseOQL
- UseLook and all specific editors

UseVersion is not allowed. In O_2C, you must use the schema O_2Version and not a library.

Targets	Description
all	constructs the application
clean	removes all objects, core, ...
install	installs the application

2.8 Building an O_2Engine API application

If you use the O2Engine API interface, you can also use O_2Makegen. If you set the option:

```
±O2APITarget
```

The generated makefile contains the necessary libraries to your executable.

For more information, refer to the *O_2Engine API* documentation set.

When building an O_2Engine API application, you can use the following options:

- UseVersion
- UseOQL
- UseLook and all specific editors

UseMeta is not allowed.

Targets	Description
all	constructs the application
clean	removes all objects, core,...
install	installs the application

2.9 Adding your own makefile

This option permits to add additional targets or actions which will be performed by the generated makefile. This can be done by writing your own makefile and having up to two targets executed when the generated makefile runs.

Options	Description
PreTarget= string list	Specifies the targets in the user-written makefile that will be executed before any rules of the generated makefile. All targets are executed once in the order listed.
PostTarget= string list	Specifies the targets in the user-written makefile that will be executed after linking rules and before install rules of the generated makefile. All targets are executed once in the order listed.
UserMakefile= file name	Specifies the user-written makefile. This makefile must contain all targets that are named in the PreTarget and PostTarget options. It can also contain the targets "all", "clean" and "install". In this case, the actions defined for these targets are executed after the corresponding actions defined in the generated makefile.

In your makefile, you can use any of the Makefile macros defined in the generated makefile.

If you modify your makefile, you must rerun O_2Makegen to regenerate the makefile.

For example:

UserMakefile= mymakefile

PreTarget= serverlaunch

PostTarget= servershut

The user-written makefile, named "mymakefile", is specified by the option UserMakefile. "mymakefile" contains the targets "serverlaunch" and "servershut". The target "serverlaunch" will be executed before anything. The target "servershut" will be executed after the construction of the executable.

2 O2Makegen Configuration File

3 Customizing O_2Makegen

This chapter is divided into the following sections :

3 Customizing O2Makegen

3.1 Template files

When O_2Makegen is used, it obtains definitions for the target system from template files. Template files are supplied with O_2Makegen in the directory $O2HOME/config.

A template file defines symbols which can be names of tools, operating system commands, options, etc.

Default template files contain all default definitions. These files have the extension ".mak". They must not be modified.

Platform template files override default definition found in the default template files. The platform template file is named using the machine and operating system name (for example: sparc_solaris_24.cf is the template file for Sparc machines with Solaris 2.4). The extension of these files is ".cf". Normally this file will not be modified.

But you can modify the site template files. The content of this file overrides the content of the platform template file. The file is named after the machine and operating system name with the ".site.cf" extension (for example: sparc_solaris_24.site.cf is the site template file for Sparc machine with Solaris 2.4). This file can be modified to adapt o2makegen to your site configuration.

A last template file is used to override definitions: the user template file. This file is introduced by the "deffile" option of the O_2Makegen command. This template file permits to have particular definitions for one user or for one project.

3.2 Syntax of a template file

A template file consists of a set of "m4" macros. "m4" is a standard UNIX macro processor. All symbols used in the generated makefile are defined in template files as m4 macros.

The primary function of m4 used in template file is "define". This is used to define and redefine macros. The following input:

```
define(name, stuff)
```

causes the string "name" to be defined as "stuff".

The left parenthesis must immediately follow the word "define" to signal that "define" has arguments.

To redefine N, the evaluation must be delayed by quoting :

```
define(N, 100)
...
define(‘N’, 200)
```

The N in the second definition is replaced by 100. The result is equivalent to the following statement:

```
define(100,200)
```

This statement causes an error since only things that look like names can be defined.

In m4, it is often wise to quote the first argument of a macro. The following example will not redefine N:

```
define(N, 100)
...
define(N, 200)
```

Each occurrence of $n in the replacement text, where n is a digit, is replaced by the n-th argument. Argument 0 is the name of the macro; missing arguments are replaced by the null string; $# is replaced by the number of arguments; $* is replaced by a list of all the arguments separated by commas; $@ is equivalent to $*, but each argument is quoted.

Comments and examples are enclosed between " # " and new-line characters. If you want to discard characters up to and including the next new-line use the macro dnl (...).

For more information consult the *m4* documentation.

For example, we put here the contents of a ".site.cf" file:

```
define(‘CCCmd’,            /usr/bin/CC)
define(‘InstallCmd’,       /etc/install -i)
define(‘PurifyCmd’,       purify)
```

```
define('StandardDefines', -DHP800 -DHPUX -DHPUX_90)

define('OptimizedCFlag',    +O1)
define('PicCFlag',          +Z)

define('OptimizedCCFlag',    +O1)
define('PicCCFlag',          +Z)
define('StdCCIncludes',      -I'/usr/'include'/CC')
define('DefaultCCOptions',   -z +a1 -pta -ptn -ptb)

define('StaticLDFlag', '-Wl,-a,archive')
define('RelocLDFlag',        -r -E)
define('DefaultLDOptions', '+a1 -Wl,-E')
define('SpecialLDOptions','$(PTREPOSITORIES)')

define('XDir',     /usr/lib/X11R5)
define('MotifDir', /usr/lib/Motif1.2)
```

In this template file, we redefine the macro CCCmd which gives the location of the C++ compiler. Note that all macro names are quoted, as explained above.

In the following section we describe each macro used in the template files.

3.3 List of modifiable macros

You can redefine macros in the "*.site.cf" template file. You can also redefine macros in the file introduced by the "deffile" option of the O_2Makegen command.

Warning !

Modifying these macros can lead to unexpected behavior of O_2Makegen.

3.3.1. Default Command Definitions

Name	Default Value	Contents
ArCmd	/bin/ar clq	to create libraries
CcCmd	/bin/cc	to run C compiler
CCCmd	CC	to run C++ compiler
ChmodCmd	/bin/chmod	to change mode of file
InstallCmd	/bin/install	to install files
LdCmd	/bin/ld	to run loader
MakeCmd	/bin/make	to run make
MkdirCmd	/bin/mkdir	to make directory
MvCmd	/bin/mv -f	to move files
CpCmd	/bin/cp	to copy files
RanlibCmd	/bin/true	to clean up libraries
RmCmd	/bin/rm -f	to delete files
TouchCmd	/bin/touch	to touch files
PurifyCmd	none	to purify files
QuantifyCmd	none	to quantify files

3.3.2. Default cc compiler flags

Name	Default Value	Contents
OptimizedCFlag	-O	cc compiler flags to turn on optimization
DebuggableCFlag	-g	cc compiler flags to turn on debug info
ProfilingCFlag	-G	cc compiler flags to turn on profiling info
PicCFlag	none	cc compiler flags to turn on pic code generation
DefaultCOptions	none	default special cc compiler options
SpecialCOptions	none	specific cc compiler options

3.3.3. Default C++ compiler flags

Name	Default Value	Contents
OptimizedCCFlag	-O	CC compiler flags to turn on optimization
DebuggableCCFlag	-g	CC compiler to turn on debug info
ProfilingCCFlag	-G	CC compiler to turn on profiling info
PicCCFlag	none	CC compiler to turn on pic code generation
DefaultCCOptions	none	default special CC compiler options
SpecialCCOptions	none	specific CC compiler options
CCExtension	cc	extension of CC source files

3.3.4. Miscellaneous flags

Name	Default Value	Contents
StdCCIncludes	none	location of specific C++ includes
StandardIncludes	none	-I's for compiler
StandardDefines	none	-D's for compiler
Xdir	none	directory containing the X Libraries
MotifDir	none	directory containing the Motif Libraries

3.3.5. Linker flags

Name	Default Value	Contents
RelocLDFlag	-r	Linker flag to create a object from a list of object
StaticLDFlag	none	Linker flag to use the static version of a library
DefaultLDOptions	none	Linker default options

3.3.6. Install flags

Name	Default Value	Contents
InstBinFlag	0755	File mode of installed binary
InstLibFlag	0664	File mode of installed library
InstIncFlag	0444	File mode of installed include

3.3.7. Files used in clean target

Name	Default Value	Contents
FilesToClean	*.o core *~ *.bak	Files to be deleted

3 Customizing O2Makegen

4 Troubleshooting Guidelines

This chapter gives advice to correct problems you may encounter when using O_2Makegen. Here is the list of error messages with their explanations.

4 Troubleshooting Guidelines

Code	Message	Explanation
27512	Configuration file is mandatory	You have not given a configuration file to O_2Makegen. Retry with a configuration file name.
27522	Unknown arguments	One or more options given to O_2Makegen are not recognized. Consult the list of O_2Makegen options.
27532	Cannot create temporary file	O_2Makegen cannot create its temporary file. Maybe the file system where temporary files are created is full, or you do not have sufficient access rights to the directory where temporary files are created. By default, this directory is /usr/tmp (on UNIX, there is no default value on windows). You can modify it by using the environment variable TMPDIR.
27542	Error during configuration file analysis	O_2Makegen has not recognized one or more options found in the configuration file. Another message gives you the unrecognized options and the line number in the configuration file. Verify the spelling of the options in the configuration file.
27552	Error during make file creation	O_2Makegen cannot create the makefile. Maybe one template file used is wrong. If you have created or modified a template file, maybe you have forgotten to quote the macro name. You can also retry to launch O_2Makegen using the pushbacksize option. This option permits to resize structures used by the analyzer of O_2Makegen. The default value is 4096. Try 8192.
27572	Cannot find machine or os type	O_2Makegen does not know the machine type or the operating system. Verify the spelling of the values of the options.
27582	Cannot find template installation directory	O_2Makegen cannot retrieve the O2HOME/config directory or you do not have read access to it. Verify your installation of O_2
27592	Cannot access template files	O_2Makegen cannot retrieve the template file. Verify the content of the O_2HOME/config directory. Verify if you have read access to the template files and to the directory. Verify your installation of O_2.
27602	Cannot find (name) executable	O_2Makegen uses other programs and cannot find one or more of these programs. Verify the existence of the program and the execution right.

5 Reference Guide

This chapter is divided into the following sections:

5 Reference Guide

5.1 Options of the O_2Makegen tool

Name	Description
-help	Displays a help facility and exit
-version	Prints version number of o2makegen and exit
-verbose	Enables verbose mode
-output *makefile_name*	Specifies the name of the generated makefile (Makefile by default)
-arch *machine*	Specifies the target architecture name
-os *os*	Specifies the target operating system name
-deffile *file_name*	Specifies the name of a private template file
-pushbacksize *int_size*	Changes pushback and argument collection size from the default size 4096.

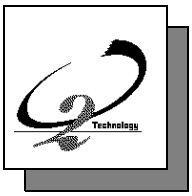
5.2 Options of configuration file

Options	Description
±[ExpClassName]Exp-NoVirtual	Specifies that the exported methods are not virtual (false by default)
[ExpClassName]Exp-Methods= method_name(s)	O_2C methods of class ExpClassName are to be exported (none by default).
[FileName][Class-Name]ImpAsClass= class_name	Imports C++ class_name as ClassName (see C++ Binding Guide)
[FileName][Class-Name]ImpForward-File=file_name	Specifies the file where the forwarded class is defined (none by default).
[FileName][Class-Name]ImpMember-Func= function_member(s)	Member function(s) to be imported (none by default)
[FileName]ImpClasses= class_name(s)	Classes to be imported. The definition of this class is found in FileName (no default).
[FileName]ImpForward-Classes= class_name(s)	All classes forwarded during the processing of File-Name (none by default).
[FileName]ImpLib-Classes=class_name(s)	Classes belonging to an external C++ library and used as a superclass of an imported class (none by default).
[FileName]ImpOutput-Dir= directory	Path and directory in which C++ files are generated for the classes in FileName. (current directory by default).
[FileName]ImpUseDi-rectory= directory_name(s)	Directory path used to find files specified in [File-Name]ImpUseFiles option (none by default).
[FileName]ImpUse-Files= file_name(s)	Files which are required to parse FileName. For example, an include file not explicitly included in the FileName file (none by default).
±[ExpClassName]Exp-Type	Type structure of the ExpClassName is also made available to C++ (False by default).
±[FileName][Class-Name]ImpAccessPri-vateMember	Automatically generates access methods for private members (false by default).
±[FileName][Class-Name]ImpAllPublic-MemberFunc	Imports all public member functions of the class (false by default)
±[FileName]ImpNo-Modification	Pointers to objects of an imported class are not modi-fied inside FileName (by default they are transformed to persistent pointers).

Options	Description
±Debug	Permits generation of debuggable executable and add _d to the executable name
±O2C++Target	Used to find which O_2 runtime libraries are necessary to build a ODMG C++ application (true by default)
±O2APITarget	Used to find which O_2 runtime libraries are necessary to build a C application using the O_2Engine API interface (false by default)
±O2CTarget	Use to find which O_2 runtime libraries are necessary to build an executable from an O_2C program (false by default)
±O2LinkCTarget	Use to find which O_2 runtime libraries are necessary to build a C application using the C interface to O_2 (false by default)
±Profile	Generates profiled executable and add _p to the executable name
±Purify	Applies Purify on the executable and add _pure to the executable name
±Quantify	Applies Quantify on the executable and add _quant to the executable name
±UseArchiveLib	Specifies that the archive version of o2 runtime library must be used instead of the shared libraries (default is false).
±UseConfirmClasses	Indicates whether classes are automatically confirmed or not after importation in o2 (default is false).
±UseLkBrowser	Indicates whether O_2Look browser editor is used or not (default is false).
±UseLkDialog	Indicates whether O_2Look dialog box editor is used or not (default is false).
±UseLkGraph	Indicates whether O_2Look graph editor is used or not (default is false).
±UseLkPict	Indicates whether O_2Look picture editor is used or not (default is false)
±UseLkText	Indicates whether O_2Look Text editor is used or not (default is false).
±UseLook	Indicates whether O_2Look is used or not (default is false).
±UseMeta	Indicates whether O_2 meta service is used or not (default is false).
±UseO2xt	Indicates whether O_2Xt is used or not (default is false).
±UseOql	Indicates whether OQL is used or not (default is false).

Options	Description
±UseVersion	Indicates whether O_2Version mechanism is used or not (default is false).
Define= string_list	Adds or renames macro definitions to the compilation or link phase. None by default.
ExpClasses= class_name(s)	O_2 classes to be exported (no default)
ExportLibFile= file name list	For AIX only. Specifies export files for shared libraries
ExpOutputDir= directory	Directory path in which C++ files are generated (current directory by default).
HeadersDestDir= directory	Directory where all include files needed to use the generated library must be installed (none by default).
ImpBag= type_name(s)	Bag of classes or atomic types are imported.
ImpFiles= file_name(s)	The C++ files where imported classes are defined (no default).
ImpList= type_name(s)	List of classes or atomic type are imported.
ImpSet= type_name(s)	Set of classes or atomic type are imported.
ImpVarray= type_name(s)	Varray of classes or atomic type are imported.
Include= directory_list	Adds directories containing files to be included at compilation or link. None by default.
LibDestDir= directory	Directory where the generated library must be installed (default none).
LibHeaders= file_name	Files to be treated as the include file of the generated library (default none).
LibName= library name	Library to build (none by default).
LibObjs= object files	Object files needed to build the library (no default).
O2Home= directory	O_2 installation directory. Default can be positioned in a .site.cf file (see CUSTOMIZING o2makegen section).
O2Schema= schema name	O_2 schema where classes will be imported (no default).
O2Server= server name	O_2 server which must be used by import and export tools (no default).
O2System= system name	O_2 system in which the schema resides(no default).
PostTarget= string list	Specifies the targets in the user-written makefile that will be executed after linking rules and before install rules of the generated makefile are executed. All targets are executed once in the order listed.

Options	Description
PreTarget= string list	Specifies the targets in the user-written makefile that will be executed before any rules of the generated makefile are executed. All targets are executed once in the order listed.
ProgramDestDir= directory	Directory in which the generated program must be installed (none by default).
ProgramLib= application libraries name list	Specifies the application specific libraries that should be linked with the application (none by default). For a library libx.a or libx.so(sl), you must give in the name list only the part x (without lib and extension).
ProgramLibDir= library directories list	Specifies the directories containing the application specific libraries (none by default).
ProgramName= executable name	Specifies the generation of the program called executable_name. (no default).
ProgramObjs= object files name list	Specifies the object files needed to build the executable (no default).
RelocatableName= relocatable_name	Generates a relocatable called relocatable_name (no default).
RelocatableObjs= object_files	Object files needed to build the relocatable object. The object files generated from the source files by an o2cpp_import or o2cpp_export command must not be in this list.
Repositories= directory_list	List of directories used as a C++ template repository during compilation or link phase. None by default.
Sources= source_files	Source files making up the application or library.
SubDirs=directory_list	List of directories in which a make must be triggered
Undefine= string_list	Removes macro definitions to the compilation or link phase. None by default.
UserLdFlags= flags for the link editor	Flags added to the link commands
UserMakefile= file name	Specifies the user-written makefile. This makefile must contain all targets that are named in the PreTarget and PostTarget options. It can also contains the targets "all", "clean" and "install". In this case, the actions defined for these targets are executed after the corresponding actions defined in the generated makefile.



INDEX

INDEX

A

C

D

F

I

J

L

M

O

P

R

S

T

U