Подключение внешних библиотек в среде Code::Blocks

Библиотека (*англ.* library) в программировании — сборник подпрограмм или объектов, используемых для разработки программного обеспечения *(определение из «Википедии»)*. Часто библиотека физически представляет собой исполняемый модуль (`*.dll` или `*.so`), код которого вызывается исполняемым файлом программы (`*.exe`). При разработке программ с использованием библиотеки требуются также заголовочные файлы, в которых описаны функции библиотеки (их реализация — в исполняемом модуле), а также статические библиотеки для связывания программы с библиотекой (`*.lib` или `*.a`): например, код загрузки соответствующей DLL. Схематично изложенное представлено на рис. 1.

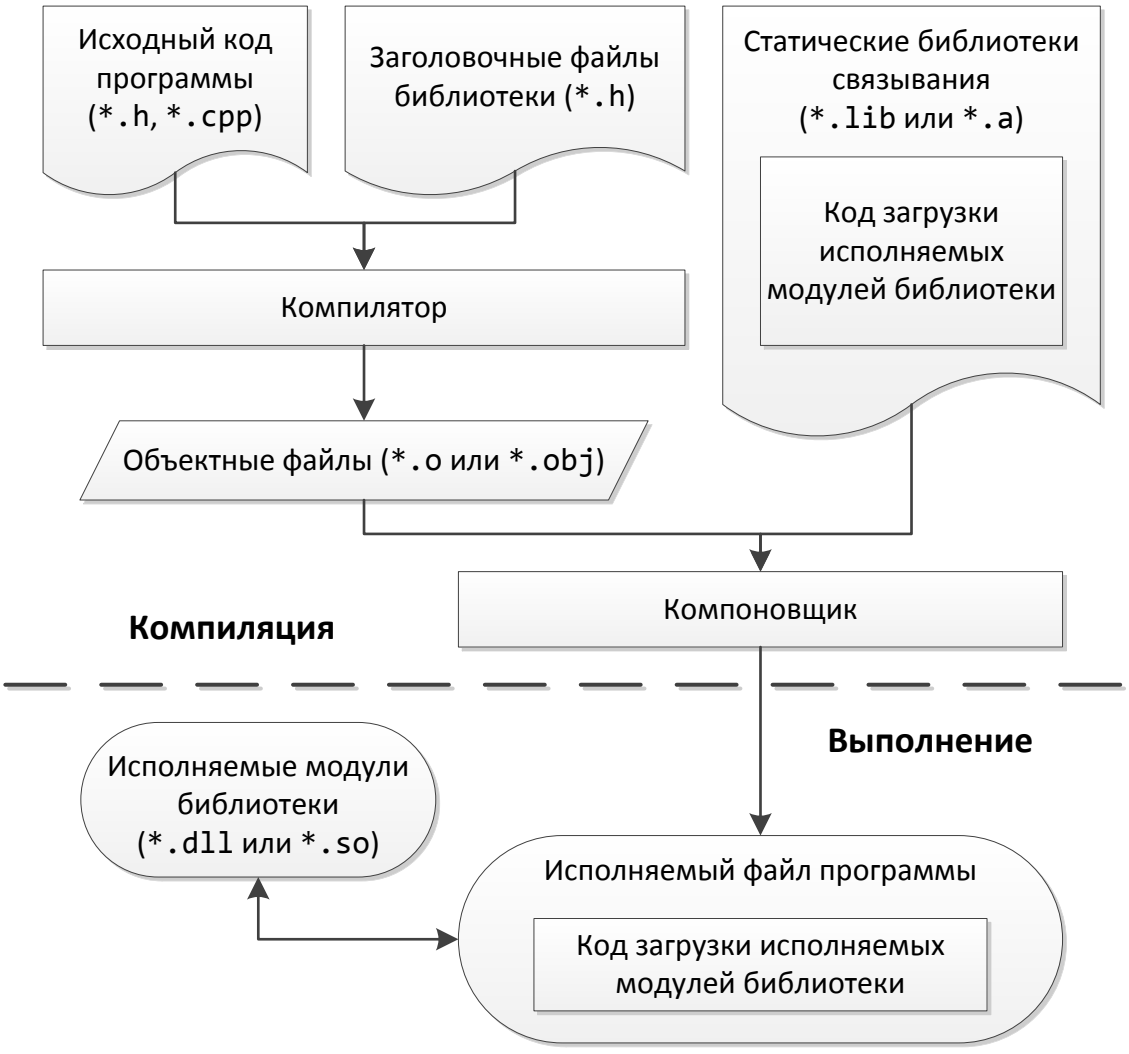

Рисунок 1 — Сборка и выполнение программ с внешними библиотеками

Видно, что для разработки требуется получить заголовочные файлы библиотеки, статические библиотеки связывания и исполняемый модуль библиотеки. Последний потребуется распространять вместе с программой. Перечисленное представляет собой т. н. набор для разработчика (*англ.* developer files, *сокр.* «devel»).

При разработке необходимо:

1) Указать компилятору, где следует искать заголовочные файлы библиотеки.
2) Указать компоновщику использовать статические библиотеки связывания при создании исполняемого файла.
3) Поместить исполняемые модули библиотеки в каталог с исполняемым файлом программы или в системный каталог для динамических библиотек.

При этом становится возможно задействовать функциональность библиотеки в программе, просто включая её заголовочные файлы директивой `#include`.

Изложенное относится к наиболее распространенному случаю, хотя существуют иные виды библиотек и немало нюансов, выходящих за пределы данной инструкции. Обычно среди файлов для разработчика имеется руководство (README), помогающее подключить данную конкретную библиотеку.

Рассмотрим подключение библиотеки на примере `libxml2`, предназначенной для работы с файлами на расширяемом языке разметки (XML) в ОС Windows. Создадим новый проект Code::Blocks в отдельном каталоге.

1. Получение необходимых файлов

Официальный сайт библиотеки (http://www.xmlsoft.org/) имеет раздел Downloads, где нас интересуют готовые сборки (*англ.* binaries) для платформы Win32. Пройдя по ссылкам, можно загрузить devel-архив со всем необходимым и распаковать его в каталог проекта.

В архиве содержатся каталоги `bin` (исполняемые модули), `lib` (статические библиотеки связывания) и `include` (заголовочные файлы). Это достаточно типично и, как правило, достаточно для работы.

2. Настройка компилятора

В свойствах сборки проекта (контекстное меню проекта, пункт «Build options…») на вкладке «Search directories» (пути поиска) нужно для компилятора добавить путь к каталогу include библиотеки (кнопкой «Add»). На вопрос среды, использовать ли относительный путь, рекомендуется ответить утвердительно, чтобы сохранить возможность перемещать каталог проекта. Диалог настройки показан на рис. 2.

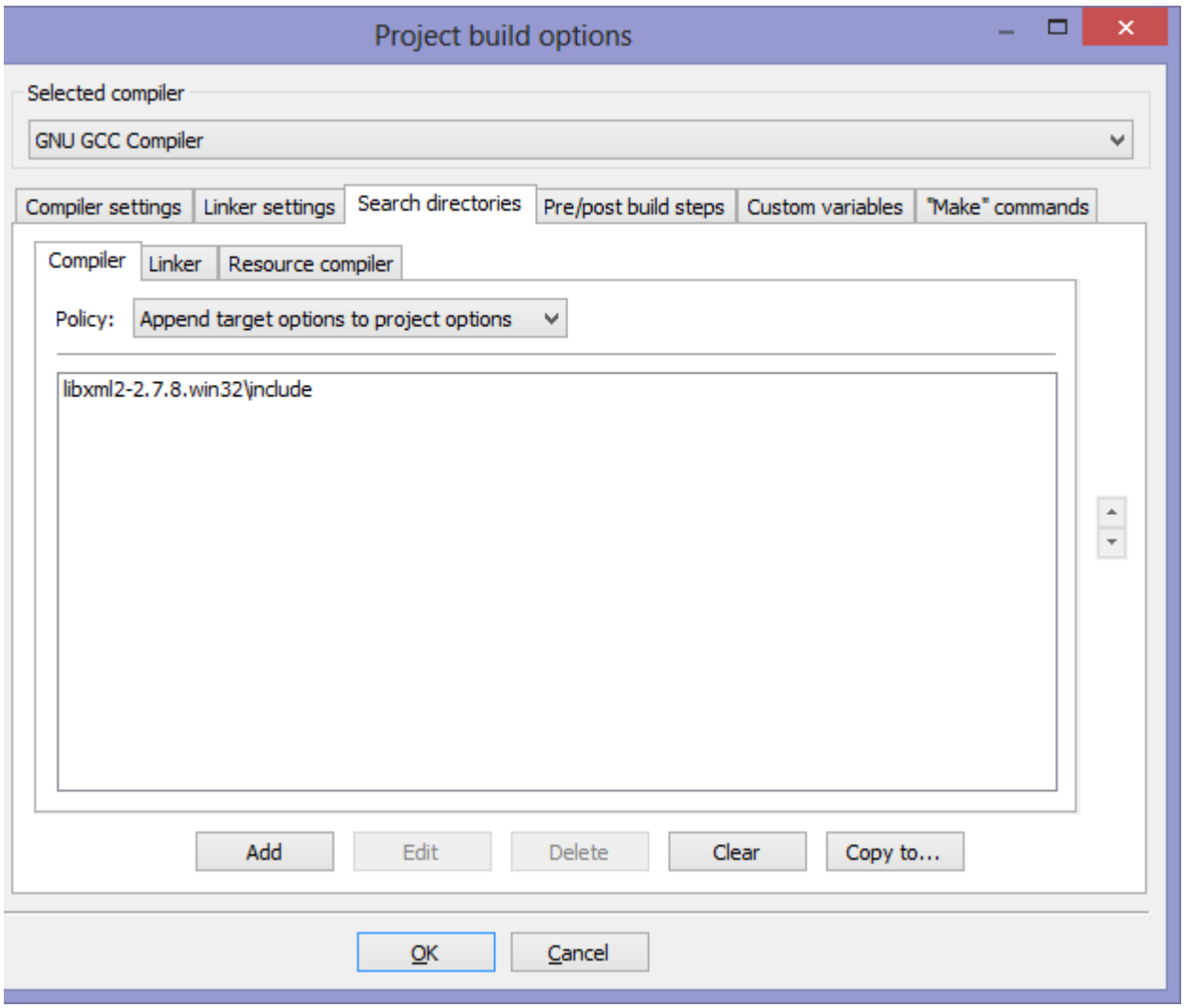

Рисунок 2 — Диалог настройки компилятора

3. Настройка компоновщика

Указать компоновщику подключать статические библиотеки связывания можно аналогично в том же диалоге на другой вкладки (рис. 3). Часто требуется указать несколько пунктов. Вообще говоря, следует заполнять этот список в зависимости от того, какая функциональность библиотеки используется, в соответствии с руководством, но сугубо для простоты в данном случае можно указать все библиотеки связывания.

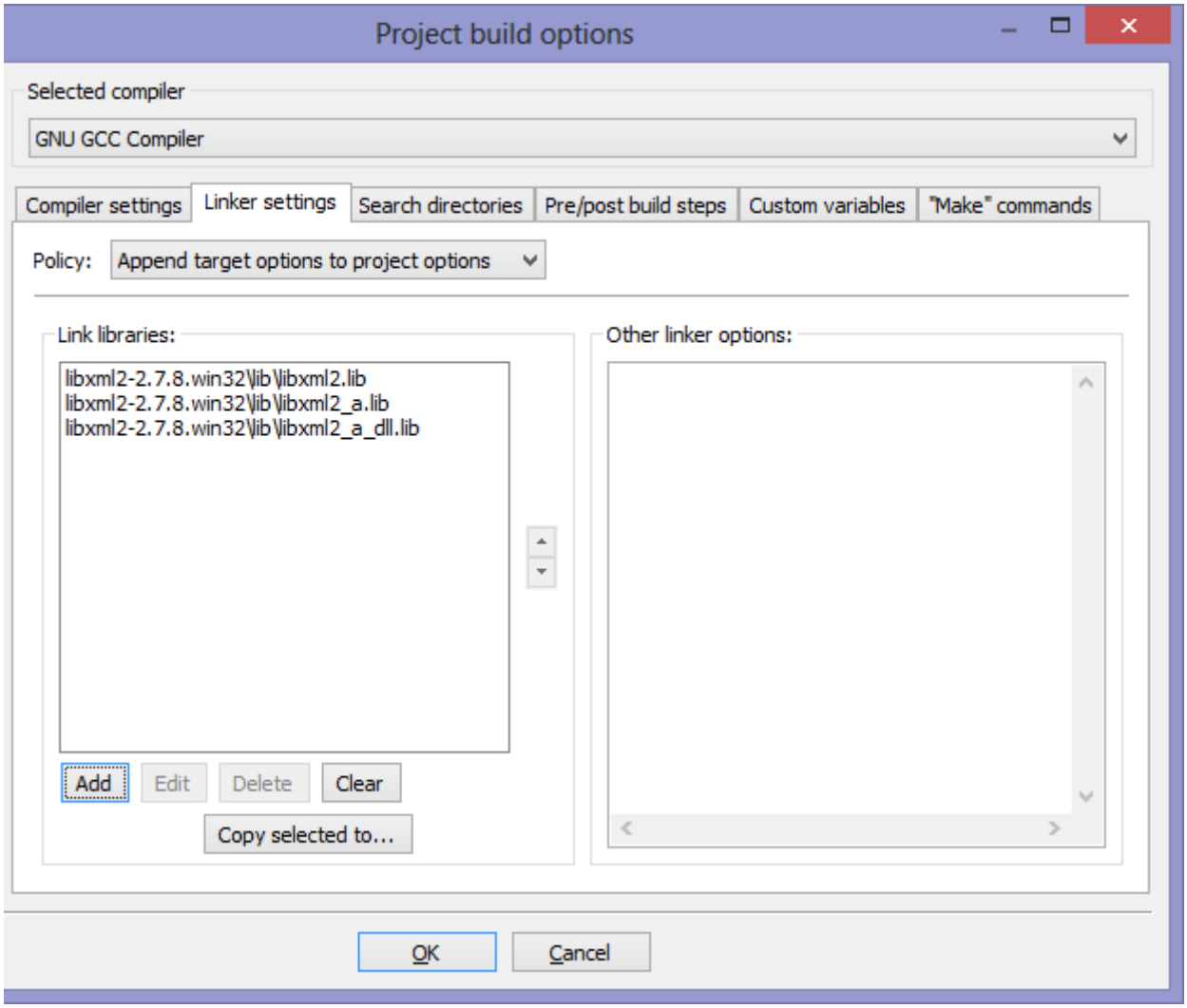

Рисунок 3 — Диалог настройки компоновщика

4. Сборка проекта

Напишем пробный код, приведенный в листинге 1, и выполним сборку проекта.

```
1  #include <libxml/xmlversion.h>
2  int main() {
3      xmlCheckVersion(LIBXML_VERSION);
4      return 0;
5  }
```

Листинг 1 — Программа для проверки поддержки нужной версии библиотеки

В первой строке подключается один из заголовочных файлов библиотеки. Имя файла указывается в угловых скобках, чтобы его поиск осуществлялся в системном каталоге и в каталогах, указанных в п. 2. В третьей строке вызывается одна из функций библиотеки для проверки поддержки исполняемым модулем той версии библиотеки, для которой программа компилировалась. Вызов важен, так как, если не пользоваться библиотекой в программе, компоновщик не задействует статические библиотеки связывания, и узнать об ошибках настройки не удастся.

Если проследить за журналом компиляции, можно видеть, как настройки, указанные в п. п. 2—3 передаются сначала компилятору, а затем компоновщику:

```
mingw32-g++.exe -Wall -fexceptions -pedantic-errors -std=c++11
     -Wall -g -pedantic-errors -std=c++11 -Wall
     -Ilibxml2-2.7.8.win32\include -c main.cpp -o obj\Debug\main.o
mingw32-g++.exe -Lc:\mingw\lib -o bin\Debug\library_demo.exe
     obj\Debug\main.o libxml2-2.7.8.win32\lib\libxml2.lib
     libxml2-2.7.8.win32\lib\libxml2_a.lib
     libxml2-2.7.8.win32\lib\libxml2_a_dll.lib
```

5. Запуск программы

Если попытаться теперь запустить программу, она аварийно завершится, так как код загрузки исполняемых модулей, внедренный компоновщиком в программу, согласно п. 3, ожидает, что в рабочем каталоге имеется файл библиотеки `libxml2.dll`. Следует скопировать этот файл в каталог проекта, а если программу потребуется распространить — передавать файл библиотеки вместе с исполняемым (из каталога `bin/Debug` или `bin/Release`).

Однако, если прочесть внимательно документацию на сайте библиотеки, можно заметить, что сама `libxml2` для своей работы нуждается в `zlib` (библиотека для сжатия и распаковки данных). Скачать последнюю можно там же, где и `libxml2`, а файл `zlib1.dll` следует поместить в каталог проекта и обращаться с ней, как с `libxml2.dll`.

При правильном выполнении всех шагов программа успешно собирается и работает, не выводя никаких сообщений о несовместимости версий, чего и требовалось добиться.