CMake による Springhead Library のビルド

第 2.4c 版 – 2021/02/18

本ドキュメントでは、CMake を用いて Springhead Library のセットアップからビルドまでを実施する流れについて説明します。unix においては、この方法が標準となります。

Springhead Library のダウンロードについては “インストールガイド (Windows 版)” をご覧ください (ダウンロード方法は各プラットフォーム共通です)。

改 訂 履 歴

第 3.0 版　全面改訂 (CMake に特化した記述に改訂)
第 2.0 版　全面改訂 及び Gitbook への移行
第 1.0 版　初版

目次

1　セットアップ

Springhead Library は、Windows および unix の両プラットフォームに対応しており、ビルド環境は CMake を用いて生成することができます (Windows では Visual Studio 用の solution/project file、unix では `Makefile`)。
この章では、Springhead Library のセットアップ方法について、Windows と unix のそれぞれについて説明します。
なお Windows については、原則は配布キットに含まれているソリューションファイルを用いたビルドであり、CMake を使用してソリューションファイルを生成・ビルドする方法はあくまでオプションです。

動作を確認しているプラットフォームは、以下のとおりです。

unix:
　Ubuntu 19.04.4 LTS (x86_64)
　cmake version 3.18.0
Windows:
　Windows 10 Enterprise
　Microsoft Visual Studio Community 2017, Version 15.9.4
　Windows SDK version 10.0.18362.0
　cmake version 3.13.32

セットアップでは、

- ビルドに必要なツールの確認
- 拡張版 `swig` のビルド (unix のみ)
- `"CMakeLists.txt"` の準備

などを実施します。
セットアップで得られた情報はセットアップファイル `"C:/Springhead/core/src/setup.conf"` に記録され、CMake の実行時に参照されます。また、セットアップが済んでいるか否かはこのファイルが存在するか否かで判断します。
なお、セットアップファイルを削除することでセットアップ前の状態を復元できます。

Springhead Library のビルドで使用するツールは次のものです。

— `python` 複数のプラットフォームに統一して対応するため。
使用する python script は python version 3 ベースで記述されています。拡張モジュール等を使用することで python 2.7.17 での動作は確認していますが、今後の修正において対応しきれないことも考えられます。Python 3 以降のバージョンが使える環境をお使いください。

— `cmake` Solution file/Makefile の生成を自動化するため。

— `swig` Springhead 用に拡張されたもの。EmbPython ライブラリをビルドするのに必要。unix の場合にはセットアップ時に生成される。

— `devenv`, `nmake` Windows 環境における Visual Studio のビルドツール。

— `gcc`, `gmake` unix 環境のビルドツール。

— `nkf` encoding 変換のため。

次の点にご注意ください。

- unix 環境において、デフォルトで gmake が見つからない場合 (`which gmake` で確認) には、gmake という名前で make にリンクを張ってください。
  例えば `cd /usr/bin; sudo ln -s ‘which make‘ gmake`
  なお、`make --version` としたときに `GNU Make 4.1` などと GNU Make の表示がでないときは、gmake のインストールが必要となります。
- Windows 環境で複数の Visual Studio がインストールされている場合には、使用する Visual Studio のバージョンを選択することができます (後述)。
- Windows の場合、`devenv` 以外のツールは `PATH` に登録しておいてください。
- unix の場合、必要なツールはすべて `PATH` に登録しておいてください。

1.1 Windows でのセットアップ

1.1.1 初めてのセットアップ

ディレクトリ "`C:/Springhead/core/src`" に移動して、スクリプト `setup.bat` を実行します。

```
> chdir C:/Springhead/core/src
> setup.bat
-- found python:  ../../buildtool/win32/python.exe
-- checking python ...  found (version 3.4.0)

setup file ("setup.conf") not exists.

currently available binaries are ...
-- checking devenv ...  selection_number:  None
    found (version:  15.9.28307.222)
-- checking nmake ...  found (version:  14.16.27025.1)
-- checking swig ...  NOT FOUND
-- checking cmake ...  found (version:  3.18.3)
-- checking nkf ...  found (version:  2.1.1 (2010-08-08))

check result is ...
-- setup is required (reason:  setup file "setup.conf" not found).

continue?  [y/n]:
```

ここで y と答えればセットアップファイルが作成されます。セットアップ作業はこれで終わりです。

(注) devenv が複数見つかった場合は次のようになりますので、適切な番号を選択してください。

```
-- checking devenv ...  selection_number:  None
found multiple "devenv"
Please select which one to use
    (1) C:/Program Files (x86)/Microsoft Visual Studio/2017/    (次の行に続く)
            Community/Common7/IDE/devenv.exe (15.9.28307.222)
    (2) C:/Users/someone/Application/Common7/IDE/devenv.exe    (次の行に続く)
            (16.8.30804.86)
enter number:
```

1.1.2 再セットアップ

初めてのセットアップと同様、次のようにしてスクリプトを実行してください。

```
> chdir C:/Springhead/core/src
> setup.bat
```

必要な環境に変更がなければ

```
check result is ...
-- no need to execute 'setup'.
done
```

となり、スクリプトは終了します。

何らかの変更がある場合、たとえば

- 使用する Visual Studio のバージョンを変えたいとき (新しい Visual Studio をインストールした場合など)
- 拡張版 swig の再ビルドが必要となったとき (別途アナウンスをします)

などの場合には

```
check result is ...
-- setup is required (reason:  "devenv" path differs,    (次の行に続く)
           "nmake" path differs).

continue?  [y/n]:
```

などとなりますので、y で答えてください。必要な処理が実行され、セットアップファイルが更新されます。

> 何らかの理由で上記の “continue? [y/n]:” が表示されないとき、または強制的にセットアップファイルを再作成したいときには、setup コマンドに -f オプションを付けて実行してください。

1.2 unix でのセットアップ

1.2.1 初めてのセットアップ

ディレクトリ "C:/Springhead/core/src" に移動して、スクリプト setup.bat を実行します。

```
> chdir C:/Springhead/core/src
> ./setup.sh
found python (Version 3.4.2)
-- checking python ...
setup file ("setup.conf") not exists.

currently available binaries are ...
-- checking python ...  found (version:  3.4.2)
-- checking gcc ...  found (version:  7.5.0)
-- checking swig ...  NOT FOUND
-- checking cmake ...  found (version:  3.18.0)
-- checking gmake ...  found (version:  4.1)
-- checking nkf ...  found (version:  2.1.4 (2015-12-12))

check result is ...
-- setup is required (reason:  setup file "setup.conf" not found,
"CMakeLists.txt" does not exist).

continue?  [y/n]:
```

ここで y と答えればセットアップファイルが作成されます。セットアップ作業はこれで終わりです。

1.2.2 再セットアップ

初めてのセットアップと同様、次のようにしてスクリプトを実行してください。

```
> chdir C:/Springhead/core/src
> ./setup.sh
```

必要な環境に変更がなければ

```
check result is ...
-- no need to execute 'setup'.
done
```

となり、スクリプトは終了します。

何らかの変更がある場合、たとえば

- 使用するツールのバージョンを変更した場合
- 拡張版 swig の再ビルドが必要となったとき (別途アナウンスをします)

などの場合には

```
check result is ...
-- setup is required (reason:  "swig" need to be rebuilt).

continue?  [y/n]:
```

などとなりますので、y で答えてください。必要な処理が実行され、セットアップファイルが更新されます。

何らかの理由で上記の“continue? [y/n]:”が表示されないとき、または強制的にセットアップファイルを再作成したいときには、setup コマンドに -f オプションを付けて実行してください。

2　オプションの変更

この章では、

- 外部パッケージを利用する
- ライブラリとヘッダファイルのインストール先を指定する
- ビルド条件を変更する

などに対する設定方法について説明します。
配布されたデフォルト状態のままで構わない場合には、次の章“3 ビルド”に進んでください。

配布キットの中には、次のファイルが含まれています。

`CMakeConf.txt.dist`	外部パッケージ導入の設定を定義する。 ヘッダファイル/ライブラリファイルのインストール先の設定を定義する。
`CMakeSettings.txt.dist`	ビルドパラメータの設定を行なう。

これらのファイルには各設定の既定値が設定されており、これらの値は cmake 実行時に参照され使用されます。
これらの既定値を変更する場合には、上記の該当するファイルを次のような名前でコピーし、コピーしたファイルを編集します。cmake は実行時にこれらのコピーしたファイルを優先して参照します。

[参考]

変数 *variable* に値 *value* を設定するには `set(`*variable* `"`*value*`")` とします。複数の値を設定するときは、空白文字で区切って並べたものを *value* とします。途中に空白やセミコロンを含まない文字列ならば引用符は省略できます。また、`${`*variable*`}` とすると他の変数の値を、`$ENV{`*variable*`}` とすると環境変数の値を参照できます。文字 ‘`#`’ 以降はコメントです。

2.1　外部パッケージを利用する

別途インストールしたパッケージを使用する場合は、ファイル `"CMakeConf.txt.dist"` の内容を次のように編集します。

変数 `CMALE_PREFIX_PATH` にパッケージを探索するパスを設定する。

```
set(CMAKE_PREFIX_PATH "C:/somewhre/appropreate")
    #           (use absolute path)
    #           (multiple paths must be separated by 'newline' or 'semicolon')
)
```

実際に探索するパスは、<prefix> | <prefix>/(cmake|CMake) | <prefix>/<name>* | <prefix>/<name>*/(cmake|CMake) などです。ここで <prefix> は変数に設定したパス、<name> はパッケージ名を表します。詳細は cmake のドキュメントを参照してください。

2.2 ライブラリとヘッダファイルのインストール先を指定する

ライブラリファイルとヘッダファイルのインストール先を設定する場合は、ファイル "CMakeConf.txt.txt" の内容を次のように編集します。

変数の設定は GUI からでも行なえます。ただし GUI での設定より "CMakeConf.txt" の設定の方が優先されます。また、GUI で CMAKE_INSTALL_PREFIX を空に設定した場合、または "CMakeConf.txt" で DO_NOT_GENERATE_INSTALL_TARGET を指定した場合には、インストールを行なうためのターゲットルールは生成されません。なお、"CMakeConf.txt" で CMAKE_INSTALL_PREFIX に空文字列を設定した場合には、GUI の場合とは異なり、デフォルトのインストールパス (Windows では "C:/Program Files"、unix では "/usr/local") が使用されます。

変数 CMAKE_INSTALL_PREFIX にインストール先を絶対パスで設定します。この設定をすることにより、find_package(Springhaed, REQUIRED) として Springhead Library を簡単に導入することができるようになります。

```
set(CMAKE_INSTALL_PREFIX "C:/where/to/install")
    #           (use absolute path)
```

実際にファイルがインストールされるディレクトリは

```
config files   →   ${CMAKE_INSTALL_PREFIX}/Springhead
header files   →   ${CMAKE_INSTALL_PREFIX}/Springhead/include
library file   →   ${CMAKE_INSTALL_PREFIX}/Springhead/lib
```

です。これらのうち、ヘッダファイルとライブラリファイルのインストール先は次の変数を設定することで変更が可能です。絶対パスまたは ${CMAKE_INSTALL_PREFIX} からの相対パスで指定してください。

```
set(SPR_HEADERS_INSTALL_DIR "header files のインストール先")
set(SPR_LIBRARY_INSTALL_DIR "library file のインストール先")
```

2.3 ビルドパラメータを変更する

ファイル "CMakeSettings.txt.txt" に定義されている変数のうち、次のものは変更することができます。

OOS_BLD_DIR	CMake の生成物を格納する作業場所 (ディレクトリ) の名称。 デフォルト値は build。
VS_VERSION	Visual Studio のバージョン。セットアップを行なっていれば変更は不要です (自動的に適切な値が採用されます)。
COMP_FLAGS_ADD	追加するコンパイルフラグ。 既定値 COMP_FLAGS は "CMakeOpts.dist" に定義されています。
LINK_FLAGS_ADD	追加するリンクフラグ。 既定値 COMP_FLAGS は "CMakeOpts.dist" に定義されています。

その他の変数は変更しないでください。

3 ビルド

Springhead Library をビルドするには

1. CMake を使用して Makefile/Solution file を生成する
2. 生成された Makefile/Solution file を用いてビルドする

の 2 段階が必要です。

以下では、CMake の生成物 (*ビルドの生成物ではありません*) を格納する作業場所 (ディレクトリ) を "*build* " として話を進めます (作業場所の名前は任意で構いません)。

3.1 CMake

CMake には Configure と Generate の 2 段階があります。

コマンドプロンプトの場合は、1 回のコマンドで両方を実行できます。

```
> chdir C:/Springhead
> mkdir build
> cmake -B build [generator]
```

generator の 詳細は、コマンドプロンプトで `cmake --help` とすると確認できます。
generator を省略した場合のデフォルトは `Unix Makefile` が選択されます。

> 注　Windows の場合にはインストールされている Visual Studio の最新バージョンがが選択されるようです。ただし、マシンアーキテクチャは自動的には判定されません。64 ビットマシンの場合には `-A x64` を指定してください。これを忘れると Visual Studio のプラットフォームが `x64` となりません。

---

generator の例
Windows: `-G "Visual Studio 15 2017" -A x64`
unix: `-G "Unix Makefiles"`

---

cmake–gui を利用する場合は、まず、次の画面で Configure ボタンを押します。

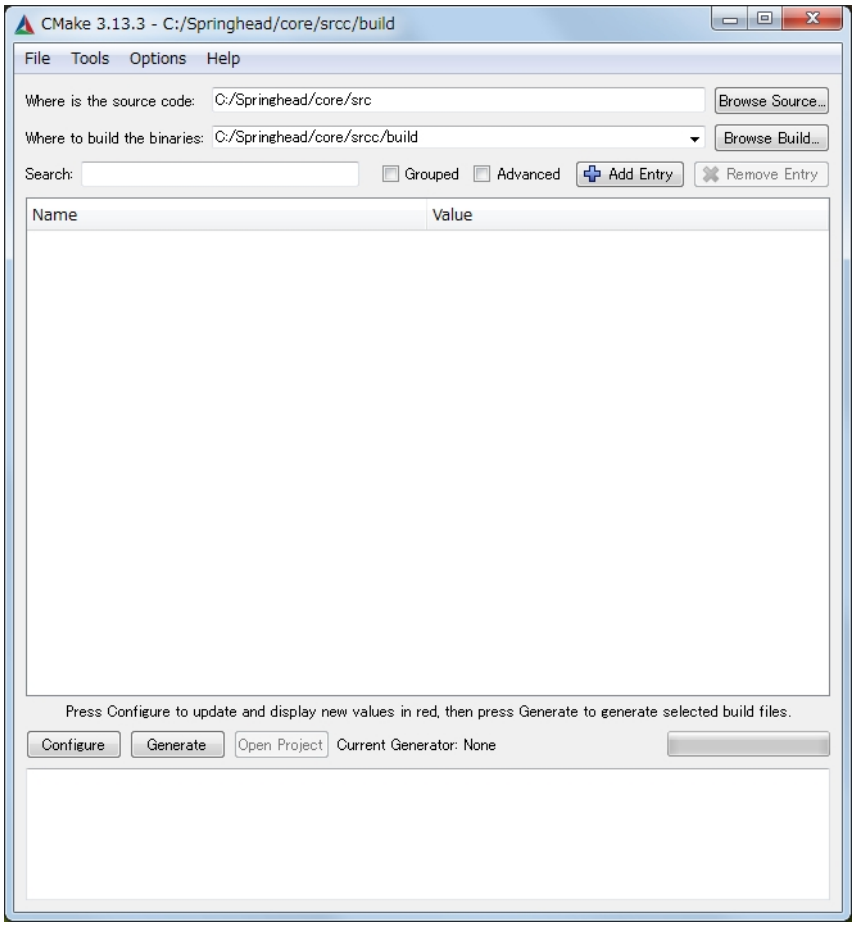

図 1　`cmake` configure

"*build*" ディレクトリがなければ作成するかどうかを尋ねられ、

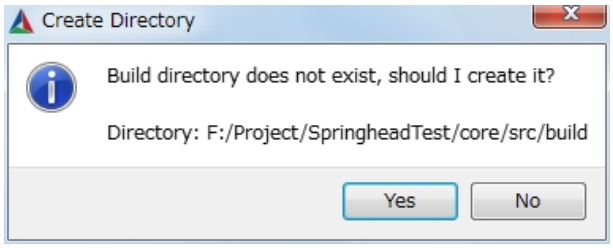

図 2　`cmake` configure

次に generator 指定画面となります。
最後に図 1 の Generate ボタンを押します。

以上で、"*build*" 以下に Makefile (unix の場合) または solution/project file (Windows の場合) が生成されたはずです。

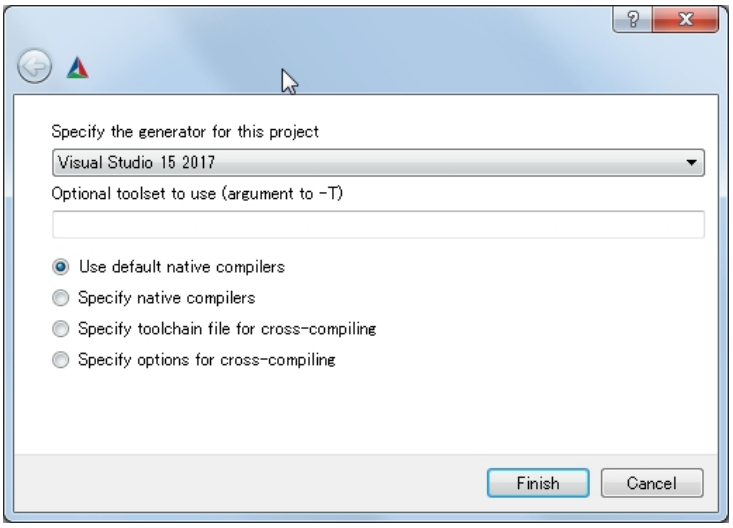

図 3　`cmake` configure

3.2　ビルド

ライブラリのビルドについては特に説明することはありません。

unix の場合

ディレクトリ *build* へ移動して make コマンドを実行してください。“2.2 インストールディレクトリの設定”でライブラリファイルのインストール先を指定していなければ、ライブラリファイルは `"C:/Springhead/generated/lib"` に生成されます。

Windows の場合

ディレクトリ *build* へ移動して `"Springhead.sln"` を Visual Studio で実行し、プロジェクト Springhead をビルドしてください。“2.2 インストールディレクトリの設定”でライブラリファイルのインストール先を指定していなければ、ライブラリファイルは `"C:/Springheadgeneratedlib<`*arch*`>"` に生成されます。`<`*arch*`>` はマシンのアーキテクチャに従い、`"win64"` または `"win32"` のいずれかです。

`INSTALL` 時の注意

ヘッダファイルとライブラリファイルのインストール先を指定している場合、ターゲット `INSTALL` を実行してもライブラリファイルが正しくインストールされない現象が発生しています (インストールされたファイルの内容が不正)。申し訳ございません。続けてもう一度 `INSTALL` を実行していただければ、正しいライブラリファイルがインストールされます。

4　ライブラリとアプリケーションの並行開発 (Windows)

この章では、Windows 上で Springhead Library とそのアプリケーションとを並行して開発する場合について説明します。

ここで述べる事項は Visual Studio に直接依存していますので、Windows 以外のプラットフォームには適用できません。他のプラットフォームの場合は、Springhead のライブラリファイルを作成してリンクするようにしてください。

4.1　重要な事項

CMake を使用してソリューションファイル/プロジェクトファイルを作成した場合には、

1. ソースファイル/ヘッダファイルは配布されたものを直接参照する。
2. ビルドにの成果物 (オブジェクトファイル/ライブラリファイル) は cmake の作業場所の中に生成される。

となります。

したがって、次の事項に注意してください。

ソースファイル/ヘッダファイルの修正

配布されたソースツリーを直接参照していますから、コミットによりリポジトリ (GitHub) に反映されます。

ソースファイル/ヘッダファイルの追加・削除

これらはプロジェクトファイルの変更を伴いますが、プロジェクトファイルは cmake の作業場所に生成されているため、配布されたソースツリーにあるプロジェクトファイルとは別物です。つまりこのような変更は、ソースツリー上にあるプロジェクトファイル (例えば `"C:/SpringheadcoresrcPhysicsPhysics15.0.vcxproj"`) も同様に変更しない限り、コミットしてもリポジトリには反映されないということです (そもそも cmake の作業場所はコミットの対象外です)。

したがって、次の点は重要です。

> ソースファイル/ヘッダファイルの追加・削除を行なったときは、対応する配布ファイル (プロジェクトファイル) も同様に修正してください。

4.2　ビルドの準備

ここでは、アプリケーションを作成するディレクトリを `"C:/Develop/Application"` とし、Springhead Library をインストールしたディレクトリは従来どおり `"C:/Springhead"` として説明を進めます。
なお、アプリケーションを単独で作成する場合 (Springhead ライブラリファイルをリンクする場合) はここでの説明の対象外です。

ディレクトリ "C:/Develop/Application" に移動してください。

配布されている次のファイルを指定した名前でコピーしてください。

CMakeLists.txt.Dev.dist	—	アプリケーション生成用設定ファイルの雛形 CMakeLists.txt という名前でコピーする。
CMakeSettings.txt.Dev.dist	—	ビルドパラメータ変更用ファイルの雛形 CMakeSettings.txt という名前でコピーする。
CMakeTopdir.txt.dist	—	ダウンロードツリー位置指定用ファイル CMakeTopdir.txt という名前でコピーする。

4.2.1 ソースツリーの場所の設定

Springhead Library をダウンロードしたディレクトリを "CMakeTopdir.txt" に設定します。これは、CMake に Springhead のソースツリーの場所を教えるため (Library のソースを add_subdirectory するため) に必要な設定です。

```
> edit CMakeTopdir.txt
  #set(TOPDIR "C:/Springhead")
    ↓
  set(TOPDIR "C:/Springhead")
```

4.2.2 ビルドパラメータの設定

ビルドパラメータを "CMakeSettings.txt" に設定します。各変数の意味は次のとおりです。

ProjectName	プロジェクト名
OOS_BLD_DIR	CMake の作業場所 (ディレクトリ) の名前 本ドキュメントで *build* としているもの。
CMAKE_CONFIGURATION_TYPES	ビルド構成 (OOS_BLD_DIR 参照)。
SPRLIBTYPE	リンクする Springhead Library の種別 STATIC を指定してください。
SRCS	ビルドの対象とするファイル 設定は set(SRCS …) または file(GLOB SRCS …) とします。後者ではワイルドカードが使えます。 SRCS の直後に “RELATIVE <*base-dir*>” を付加すると相対パス指定となります。デフォルトは file(GLOB SRCS RELATIVE ${CMAKE_SOURCE_DIR} *.cpp *.h) です。
EXCLUDE_SRCS	ビルドの対象から外すファイル SRCS でワイルドカードを使用した場合に有用です。上の SRCS で RELATIVE としていないときは絶対パスで指定します。

SPR_PROJS	アプリケーションに組み込む Springhead Library のプロジェクト名 (この中に RunSwig を含めてはいけません)
DEFINE_MACROS_ADD	追加のコンパイルマクロ指定 (/D, -D は不要です)
INCLUDE_PATHS_ADD	追加のインクルードパス指定 (/I, -I は不要です) 現在のディレクトリは ${CMAKE_SOURCE_DIR} で参照できます。
COMP_FLAGS_ADD	追加のコンパイラフラグ指定
LINK_FLAGS_ADD	追加のリンカフラグ指定
LIBRARY_PATHS_ADD	追加のライブラリパス指定 (-L は不要です)
LIBRARY_NAMES_ADD	SPRLIBTYPE が STATIC の場合 追加のライブラリファイル名 (-l は不要です)
STATIC_LIBRARY_NAMES_ADD	SPRLIBTYPE が SHARED の場合 追加の静的ライブラリファイル名 (-l は不要です)
SHARED_LIBRARY_NAMES_ADD	SPRLIBTYPE が SHARED の場合 追加の共有ライブラリファイル名 (-l は不要です)
EXCLUDE_LIBRARY_NAMES	リンクの対象から外すライブラリファイル名 デフォルトで組み込まれてしまうライブラリファイルを排除するために指定します。
VS_VERSION	使用する Visual Studio のバージョン setup を実行してあれば設定する必要はありません。
DEBUGGER_WORKING_DIRECTORY	Visual Studio Debugger の作業ディレクトリ名 デバッガはこのディレクトリで起動されたように振る舞います。
DEBUGGER_COMMAND_ARGUMENTS	Visual Studio Debugger に渡すコマンド引数

4.2.3 その他

別途インストールしているパッケージ (boost, glew, freeglut, glui) を使用する場合には、配布されたファイル "CMakeConf.txt.dist" を "CMakeConf.txt" という名前でコピーして必要な編集をします。

以上で準備作業は終了です。

4.3 CMake から実行まで

アプリケーションの cmake からビルドまでの流れはライブラリの場合と同様です。“3.1 CMake”及び“3.2 ビルド”をご参照ください。

実行時に DLL が見つからないというエラーが発生した場合には、

32 ビット環境のときは —	`"C:/Springhead/dependency/bin/win32"`
64 ビット環境のときは —	`"C:/Springhead/dependecny/bin/win64"` と `"C:/Springhead/dependecny/bin/win32"` の両方

にパスを通してください。Visual Studio から実行するときは、プログラムのプロパティを開き、[構成プロパティ]—[デバッグ]—[環境] に “`path=上記のパス`” とします。