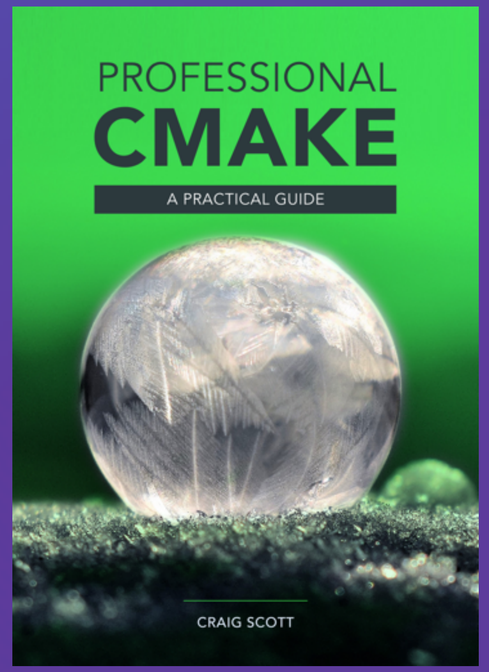
# Deep CMake For Library Authors

Craig Scott

CppCon 2019

# About Me

- Cross-platform C++ developer since 2001
- CMake co-maintainer (volunteer)
- Author of *Professional CMake: A Practical Guide*
- Consulting services available through Crascit Pty Ltd




https://crascit.com

@crascit

# Focus of Talk

Libraries (mostly shared)

Cross-platform considerations

Highlight CMake features

# Key Questions For Library Authors

## API Control

What does the library provide?

## Library Consumers

How might the library be used?

## API Compatibility

How does the library evolve?

## Package Maintainers

How might the library be packaged?

# API Control

- Be clear about what is included in the API
- Don’t expose things that are not part of the API

# API Control

- Be clear about what is included in the API
- Don’t expose things that are not part of the API

Documentation


Headers

Symbol visibility

# API Control

- Be clear about what is included in the API
- Don’t expose things that are not part of the API

Documentation


Headers

Symbol visibility

# How To Control Visibility

```
class MyGenerator
{
public:
    int nextValue();
};
```

# How To Control Visibility

```
class MyGenerator
{
public:
    int nextValue();
};
```

Visual Studio hides symbols by default

GCC and Clang do NOT hide symbols by default

# Visual Studio Visibility Control

```
class __declspec(dllexport) MyGenerator
{
public:
    int nextValue();
};
```

# Visual Studio Visibility Control

```
class __declspec(dllexport) MyGenerator
{
public:
    int nextValue();
};
```

# Visual Studio Visibility Control

```
class __declspec(dllexport) MyGenerator
{
public:
    int nextValue();
};
```

Building

```
class __declspec(dllimport) MyGenerator
{
public:
    int nextValue();
};
```

Using

# Visual Studio Visibility Control

```
class __declspec(dllexport) MyGenerator
{
public:
    int nextValue();
};
```

# Visual Studio Visibility Control

```
class MYTGT_EXPORT MyGenerator
{
public:
    int nextValue();
};
```

# Visual Studio Visibility Control

```
#include “mytgt_export.h”


class MYTGT_EXPORT MyGenerator
{
public:
    int nextValue();
};
```

Header defines MYTGT_EXPORT

# Visual Studio Visibility Control

```
#include “mytgt_export.h”


class MYTGT_EXPORT MyGenerator
{
public:
    int nextValue();
};
```

Header defines MYTGT_EXPORT

```
#ifndef MYTGT_EXPORT
#  ifdef MyTgt_EXPORTS
#    define MYTGT_EXPORT __declspec(dllexport)
#  else
#    define MYTGT_EXPORT __declspec(dllimport)
#  endif
#endif
```

# Visual Studio Visibility Control

```
#include “mytgt_export.h”


class MYTGT_EXPORT MyGenerator
{
public:
    int nextValue();
};
```

Only define MyTgt_EXPORTS when building the library

```
#ifndef MYTGT_EXPORT
#  ifdef MyTgt_EXPORTS
#    define MYTGT_EXPORT __declspec(dllexport)
#  else
#    define MYTGT_EXPORT __declspec(dllimport)
#  endif
#endif
```

# GCC/Clang Visibility Control

- Change default visibility to hidden

  `-fvisibility=hidden`

- Change visibility of inlined code (including templates)

  `-fvisibility-inlines-hidden`

# GCC/Clang Visibility Control

```
class __attribute__((visibility("default"))) MyGenerator
{
public:
    int nextValue();
};
```

# GCC/Clang Visibility Control

```
class MYTGT_EXPORT MyGenerator
{
public:
    int nextValue();
};
```

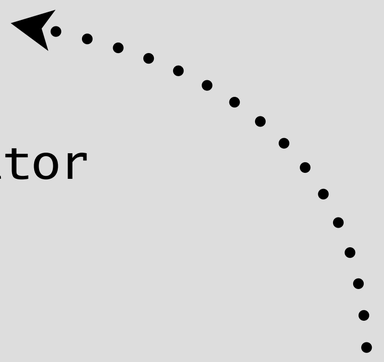
# GCC/Clang Visibility Control

```
#include “mytgt_export.h”


class MYTGT_EXPORT MyGenerator
{
public:
    int nextValue();
};
```

Header defines MYTGT_EXPORT

# GCC/Clang Visibility Control

```
#include “mytgt_export.h”


class MYTGT_EXPORT MyGenerator
{
public:
    int nextValue();
};
```



Header defines MYTGT_EXPORT

```
#ifndef MYTGT_EXPORT
#  define MYTGT_EXPORT __attribute__((visibility(“default”)))
#endif
```

# CMake Visibility Control

```
set(CMAKE_CXX_VISIBILITY_PRESET     hidden)
set(CMAKE_VISIBILITY_INLINES_HIDDEN YES)

add_library(MyTgt ...)

include(GenerateExportHeader)
generate_export_header(MyTgt)
```

# CMake Visibility Control

```
set(CMAKE_CXX_VISIBILITY_PRESET     hidden)
set(CMAKE_VISIBILITY_INLINES_HIDDEN YES)

add_library(MyTgt ...)

include(GenerateExportHeader)
generate_export_header(MyTgt)
```

Set default visibility to hidden for all targets

# CMake Visibility Control

```
set(CMAKE_CXX_VISIBILITY_PRESET     hidden)
set(CMAKE_VISIBILITY_INLINES_HIDDEN YES)

add_library(MyTgt ...)

include(GenerateExportHeader)
generate_export_header(MyTgt)
```

Generates a suitable **mytgt_export.h**

Ensures **MYTGT_EXPORT** is defined

Adds **MyTgt_EXPORTS** definition to **MyTgt**

# Export Examples

```
#include “mytgt_export.h”


class MYTGT_EXPORT MyGenerator
{
public:
    int nextValue();
};


MYTGT_EXPORT double computeSomething();


MYTGT_EXPORT extern int naughtyGlobal;
```

# API Compatibility

Communicating what sort of changes were made since last release

# API Compatibility

Communicating what sort of changes were made since last release

- Use a conventional versioning strategy

# API Compatibility

Communicating what sort of changes were made since last release

- Use a conventional versioning strategy
- Consider semantic versioning

https://semver.org

# API Compatibility

Communicating what sort of changes were made since last release

- Use a conventional versioning strategy
- Consider semantic versioning

https://semver.org

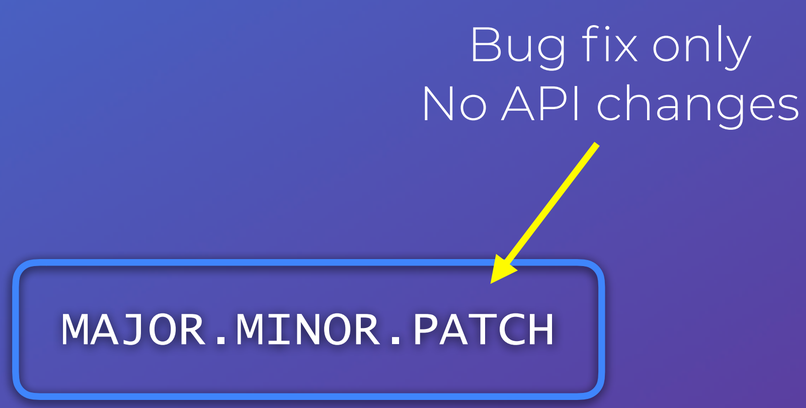
```
MAJOR.MINOR.PATCH
```

# API Compatibility

Communicating what sort of changes were made since last release

- Use a conventional versioning strategy
- Consider semantic versioning

https://semver.org

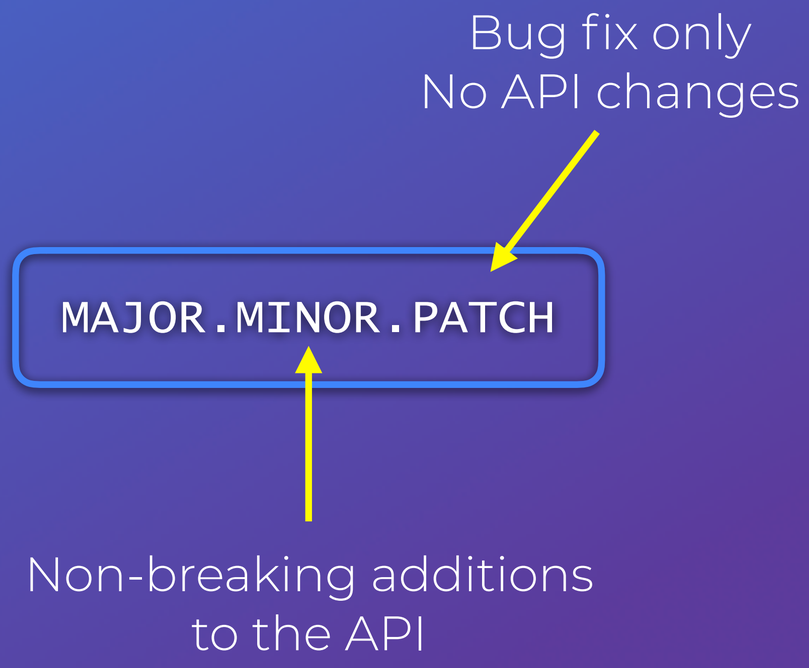


# API Compatibility

Communicating what sort of changes were made since last release

- Use a conventional versioning strategy
- Consider semantic versioning

https://semver.org

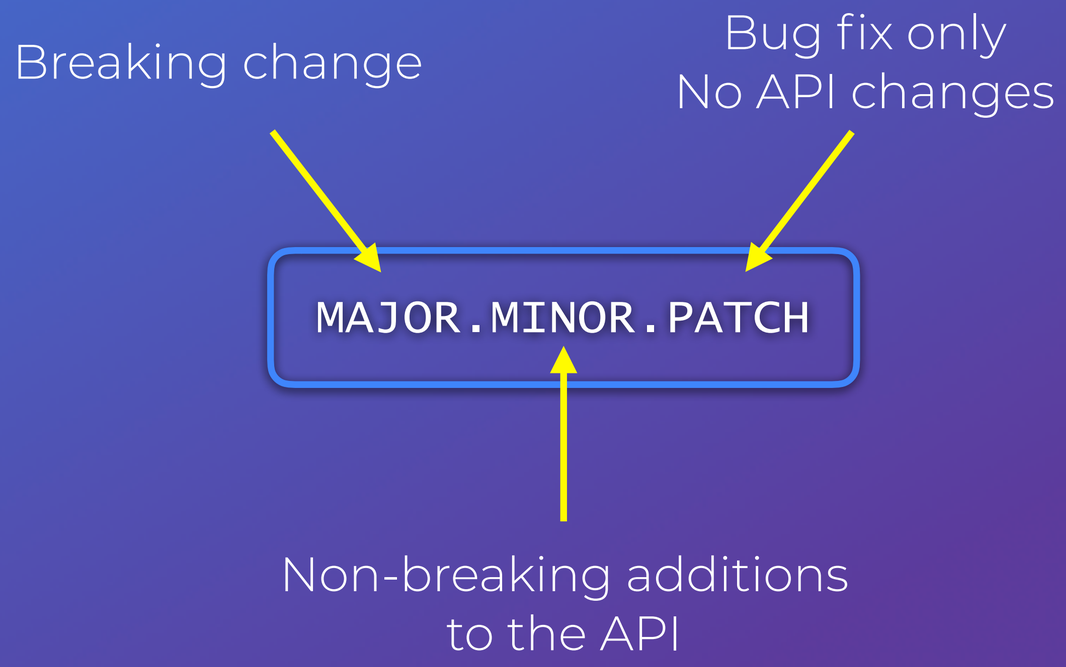


# API Compatibility

Communicating what sort of changes were made since last release

- Use a conventional versioning strategy
- Consider semantic versioning

https://semver.org



# Shared Library Symlinks

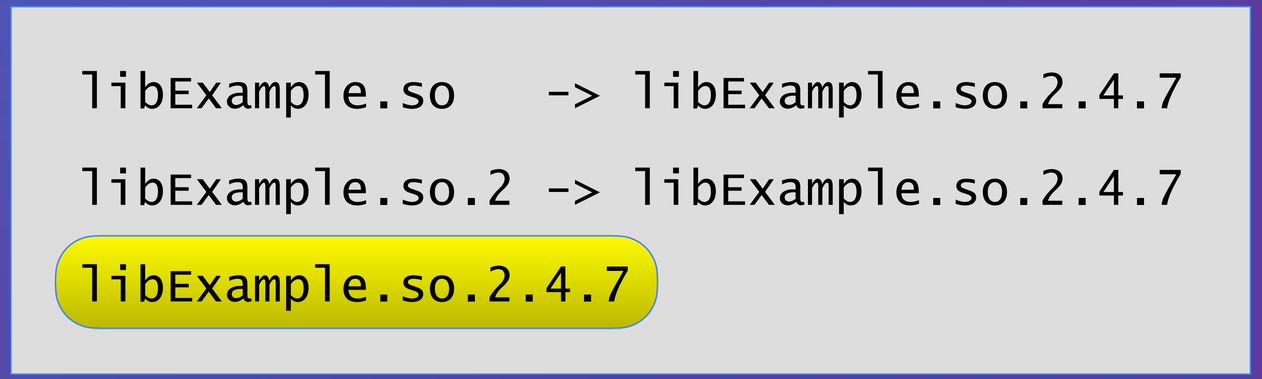
Common convention used on Unix and Unix-like operating systems

Ordering of suffix and version number may vary, but principle is the same

```
libExample.so   -> libExample.so.2.4.7
libExample.so.2 -> libExample.so.2.4.7
libExample.so.2.4.7
```

# Shared Library Symlinks

REAL LIBRARY



# Shared Library Symlinks

Humans, packages

REAL LIBRARY

```
libExample.so   -> libExample.so.2.4.7
libExample.so.2 -> libExample.so.2.4.7
libExample.so.2.4.7
```

# Shared Library Symlinks

Humans, packages

SONAME

REAL LIBRARY

```
libExample.so   -> libExample.so.2.4.7
libExample.so.2 -> libExample.so.2.4.7
libExample.so.2.4.7
```

# Shared Library Symlinks

Run-time loader — SONAME

Humans, packages — REAL LIBRARY

```
libExample.so   -> libExample.so.2.4.7
libExample.so.2 -> libExample.so.2.4.7
libExample.so.2.4.7
```

Check with commands like `ldd` or `otool -L`

# Shared Library Symlinks

Run-time loader

Humans, packages

NAME LINK

SONAME

REAL LIBRARY

```
libExample.so   -> libExample.so.2.4.7
libExample.so.2 -> libExample.so.2.4.7
libExample.so.2.4.7
```

# Shared Library Symlinks

| | | |
|---|---|---|
| Build-time linker | NAME LINK | `libExample.so -> libExample.so.2.4.7` |
| Run-time loader | SONAME | `libExample.so.2 -> libExample.so.2.4.7` |
| Humans, packages | REAL LIBRARY | `libExample.so.2.4.7` |

Specified on linker command line as `-lExample`

# Shared Library Symlinks

| | | |
|---|---|---|
| Build-time linker | NAME LINK | `libExample.so -> libExample.so.2.4.7` |
| Run-time loader | SONAME | `libExample.so.2 -> libExample.so.2.4.7` |
| Humans, packages | REAL LIBRARY | `libExample.so.2.4.7` |

**SONAME** is most critical from a compatibility perspective

# Library Versioning In CMake

```
add_library(Example ...)

set_target_properties(
  Example PROPERTIES
    SOVERSION  2
    VERSION    2.4.7
)
```

```
libExample.so   -> libExample.so.2.4.7
libExample.so.2 -> libExample.so.2.4.7
libExample.so.2.4.7
```

# Library Versioning In CMake

```
add_library(Example ...)

set_target_properties(
  Example PROPERTIES
    SOVERSION  2
    VERSION    2.4.7
)
```

```
libExample.so   -> libExample.so.2.4.7
libExample.so.2 -> libExample.so.2.4.7
libExample.so.2.4.7
```

# Library Versioning In CMake

```
add_library(Example ...)

set_target_properties(
  Example PROPERTIES
    SOVERSION  2
    VERSION    2.4.7
)
```

```
libExample.so   -> libExample.so.2.4.7
libExample.so.2 -> libExample.so.2.4.7
libExample.so.2.4.7
```

# Library Versioning In CMake

```
add_library(Example ...)

set_target_properties(
  Example PROPERTIES
    SOVERSION  2
    VERSION    2.4.7
)
```

```
libExample.so   -> libExample.so.2.4.7
libExample.so.2 -> libExample.so.2.4.7
libExample.so.2.4.7
```

Always created

# Library Versioning In CMake

```
add_library(Example ...)

set_target_properties(
  Example PROPERTIES
    SOVERSION  2
    VERSION    2.4.7
)
```

Missing SONAME

?

# Library Versioning In CMake

```
add_library(Example ...)

set_target_properties(
  Example PROPERTIES
    SOVERSION  2
    VERSION    2.4.7
)
```

```
libExample.so   -> libExample.so.2.4.7
libExample.so.2 -> libExample.so.2.4.7

libExample.so.2.4.7
```

If **SOVERSION** is missing, it defaults to same as **VERSION**

# Library Versioning In CMake

```
add_library(Example ...)

set_target_properties(
  Example PROPERTIES
    SOVERSION   2
    VERSION     2.4.7
)
```

```
libExample.so    -> libExample.so.2.4.7
libExample.so.2  -> libExample.so.2.4.7
libExample.so.2.4.7
```

SONAME

If **SOVERSION** is missing, it defaults to same as **VERSION**

# Library Versioning In CMake

```
add_library(Example ...)

set_target_properties(
  Example PROPERTIES
    SOVERSION  2
    VERSION    2.4.7
)
```

```
libExample.so    -> libExample.so.2.4.7
libExample.so.2  -> libExample.so.2.4.7
libExample.so.2.4.7
```

SONAME

If **SOVERSION** is missing, it defaults to same as **VERSION**

**PROBABLY WRONG!**

# Library Versioning In CMake

```
add_library(Example ...)

set_target_properties(
  Example PROPERTIES
    SOVERSION  9
    VERSION    2.4.7
)
```

Independent SONAME

?

# Library Versioning In CMake

```
add_library(Example ...)

set_target_properties(
  Example PROPERTIES
    SOVERSION  9
    VERSION    2.4.7
)
```

```
libExample.so   -> libExample.so.2.4.7
libExample.so.9 -> libExample.so.2.4.7
libExample.so.2.4.7
```

Is this valid?

# Library Versioning In CMake

```
add_library(Example ...)

set_target_properties(
  Example PROPERTIES
    SOVERSION  9
    VERSION    2.4.7
)
```

```
libExample.so   -> libExample.so.2.4.7
libExample.so.9 -> libExample.so.2.4.7
libExample.so.2.4.7
```

Is this valid?  **YES!**

# Library Versioning In CMake

```
add_library(Example ...)

set_target_properties(
  Example PROPERTIES
    SOVERSION  2
    VERSION    2.4.7
)
```

**Windows?**

# Library Versioning In CMake

```
add_library(Example ...)

set_target_properties(
  Example PROPERTIES
    SOVERSION  2
    VERSION    2.4.7
)
```

```
Example.dll

Example.lib
```

# Library Versioning In CMake

```
add_library(Example ...)

set_target_properties(
  Example PROPERTIES
    SOVERSION  2
    VERSION    2.4.7
)
```

```
Example.dll
Example.lib
```

Acts like SONAME

# Library Versioning In CMake

```
add_library(Example ...)

set_target_properties(
  Example PROPERTIES
    SOVERSION  2
    VERSION    2.4.7
)
```

Example.dll — Acts like SONAME

Example.lib — Acts like NAME LINK

# Library Versioning In CMake

```
add_library(Example ...)

set_target_properties(
  Example PROPERTIES
    SOVERSION  2
    VERSION    2.4.7
)
```

Example.dll — Acts like SONAME

Example.lib — Acts like NAME LINK

Some version details may be encoded into the binaries, but not into the file names

# Package Versioning

```
find_package(SomeProj 2.3)
```

# Package Versioning

```
find_package(SomeProj 2.3)
```

- SomeProjConfig.cmake
- SomeProjConfigVersion.cmake

- someproj-config.cmake
- someproj-config-version.cmake

# Package Versioning

```
find_package(SomeProj 2.3)
```

- SomeProjConfig.cmake
- SomeProjConfigVersion.cmake

# Package Versioning

```
find_package(SomeProj 2.3)
```

- SomeProjConfig.cmake
- SomeProjConfigVersion.cmake

# Package Versioning

```
find_package(SomeProj 2.3)
```

- SomeProjConfig.cmake
- SomeProjConfigVersion.cmake

# Package Versioning

```
find_package(SomeProj 2.3)
```

- SomeProjConfig.cmake
- SomeProjConfigVersion.cmake

```
include(CMakePackageConfigHelpers)

write_basic_package_version_file(
  SomeProjConfigVersion.cmake
  VERSION 2.4.7
  COMPATIBILITY SameMajorVersion
)
```

# How Might A Library Be Packaged?

- By you in your own dedicated package
- As part of some other package (i.e. an embedded dependency)
- By a distribution maintainer
- By a packaging system not part of the OS

# Installing Libraries With CMake

```
install(TARGETS Example DESTINATION lib)
```

# Installing Libraries With CMake

```
install(TARGETS Example DESTINATION lib)
```

2/10

# Installing Libraries With CMake

```
include(GNUInstallDirs)
install(TARGETS Example
  RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
  LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
  ARCHIVE DESTINATION ${CMAKE_INSTALL_LIBDIR}
)
```

# Installing Libraries With CMake

```
include(GNUInstallDirs)
install(TARGETS Example
  RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
  LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
  ARCHIVE DESTINATION ${CMAKE_INSTALL_LIBDIR}
)
```

Windows DLLs

# Installing Libraries With CMake

```
include(GNUInstallDirs)
install(TARGETS Example
  RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
  LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
  ARCHIVE DESTINATION ${CMAKE_INSTALL_LIBDIR}
)
```

Non-Windows shared libraries
(including symlinks)

# Installing Libraries With CMake

```
include(GNUInstallDirs)
install(TARGETS Example
  RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
  LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
  ARCHIVE DESTINATION ${CMAKE_INSTALL_LIBDIR}
)
```

Static libraries (all platforms)
Windows import libraries

# Installing Libraries With CMake

```
include(GNUInstallDirs)
install(TARGETS Example
  RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
  LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
  ARCHIVE DESTINATION ${CMAKE_INSTALL_LIBDIR}
)
```

# Installing Libraries With CMake

```
include(GNUInstallDirs)
install(TARGETS Example
  RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
  LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
  ARCHIVE DESTINATION ${CMAKE_INSTALL_LIBDIR}
)
```

```
/usr/lib
/usr/lib64
/usr/lib/x86_64-linux-gnu
 ...
```

# Installing Libraries With CMake

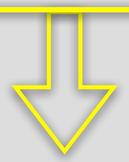
```
include(GNUInstallDirs)
install(TARGETS Example
  RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
  LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
  ARCHIVE DESTINATION ${CMAKE_INSTALL_LIBDIR}
)
```

```
install(TARGETS Example)
```

Requires at least CMake 3.14

# Installing Libraries With CMake

```
include(GNUInstallDirs)
install(TARGETS Example
  RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
  LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
  ARCHIVE DESTINATION ${CMAKE_INSTALL_LIBDIR}
)
```



```
install(TARGETS Example)
```

Requires at least CMake 3.14

5/10

# Installing Libraries With CMake

```
include(GNUInstallDirs)
install(TARGETS Example
  RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
          COMPONENT   Runtime
  LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
          COMPONENT           Runtime
          NAMELINK_COMPONENT Development
  ARCHIVE DESTINATION ${CMAKE_INSTALL_LIBDIR}
          COMPONENT   Development
)
```

# Installing Libraries With CMake

```
include(GNUInstallDirs)
install(TARGETS Example
  RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
          COMPONENT   Runtime
  LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
          COMPONENT           Runtime
          NAMELINK_COMPONENT Development
  ARCHIVE DESTINATION ${CMAKE_INSTALL_LIBDIR}
          COMPONENT   Development
)
```

# Installing Libraries With CMake

```
include(GNUInstallDirs)
install(TARGETS Example
  RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
          COMPONENT   Runtime
  LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
          COMPONENT           Runtime
          NAMELINK_COMPONENT Development
  ARCHIVE DESTINATION ${CMAKE_INSTALL_LIBDIR}
          COMPONENT   Development
)
```

Requires at least CMake 3.12

# Installing Libraries With CMake

```
include(GNUInstallDirs)
install(TARGETS Example
  RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
          COMPONENT   Runtime
  LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
          COMPONENT           Runtime
          NAMELINK_COMPONENT Development
  ARCHIVE DESTINATION ${CMAKE_INSTALL_LIBDIR}
          COMPONENT   Development
)
```

# Installing Libraries With CMake

```
include(GNUInstallDirs)
install(TARGETS Example
  RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
          COMPONENT   Runtime
  LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
          COMPONENT           Runtime
          NAMELINK_COMPONENT Development
  ARCHIVE DESTINATION ${CMAKE_INSTALL_LIBDIR}
          COMPONENT   Development
)
```

# Installing Libraries With CMake

```
include(GNUInstallDirs)
install(TARGETS Example
  RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
          COMPONENT   Runtime
  LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
          COMPONENT           Runtime
          NAMELINK_COMPONENT  Development
  ARCHIVE DESTINATION ${CMAKE_INSTALL_LIBDIR}
          COMPONENT   Development
)
```

8/10

# Installing Libraries With CMake

```
include(GNUInstallDirs)
install(TARGETS Example
  RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
          COMPONENT   SomeProj_RunTime
  LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
          COMPONENT           SomeProj_RunTime
          NAMELINK_COMPONENT  SomeProj_Development
  ARCHIVE DESTINATION ${CMAKE_INSTALL_LIBDIR}
          COMPONENT   SomeProj_Development
)
```

# Installing Libraries With CMake

```
include(GNUInstallDirs)
install(TARGETS Example
  RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
          COMPONENT   SomeProj_RunTime
  LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
          COMPONENT           SomeProj_RunTime
          NAMELINK_COMPONENT SomeProj_Development
  ARCHIVE DESTINATION ${CMAKE_INSTALL_LIBDIR}
          COMPONENT   SomeProj_Development
)
```

# Installing Libraries With CMake

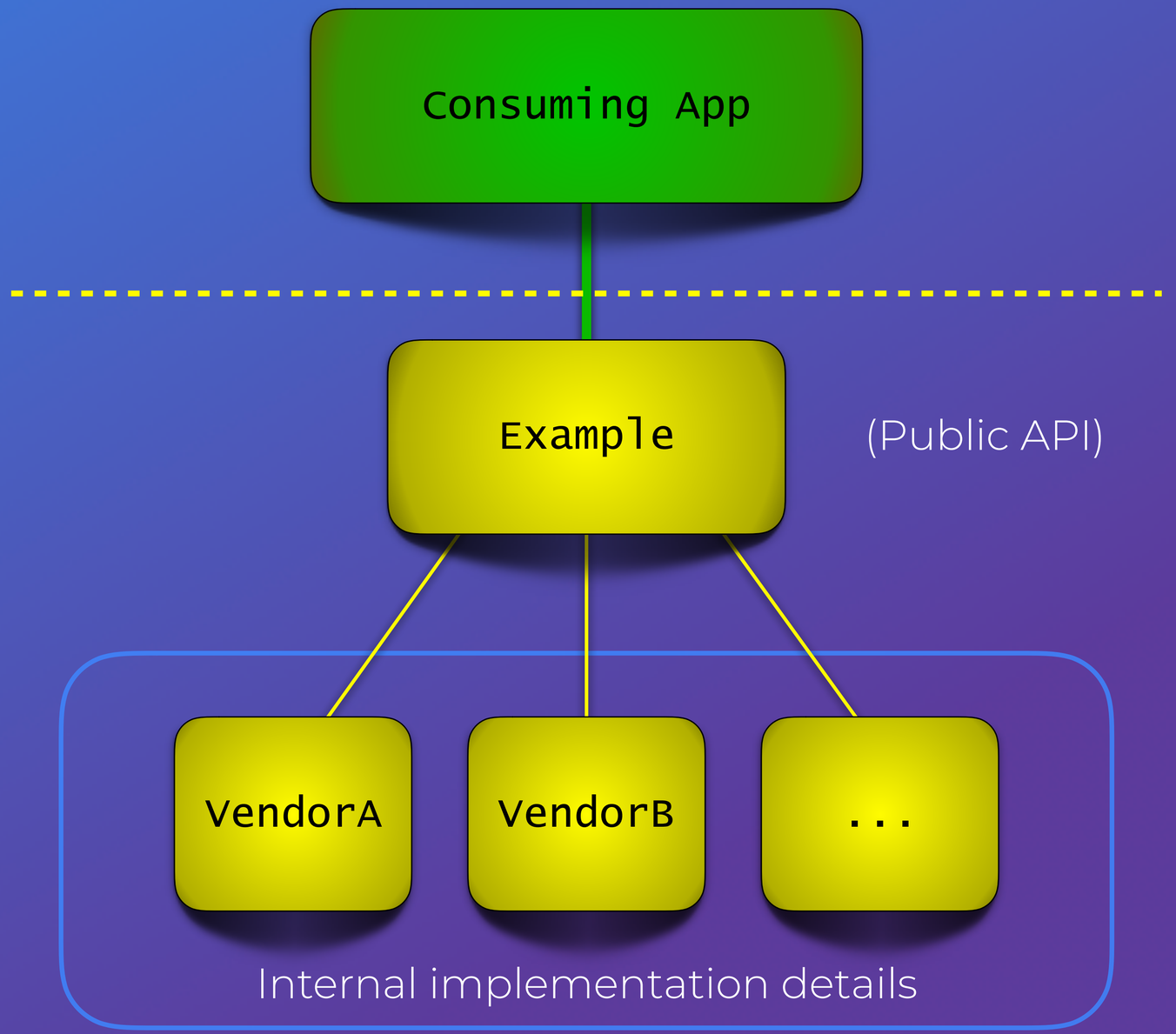
```
include(GNUInstallDirs)
install(TARGETS Example
  RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
          COMPONENT   SomeProj_RunTime
  LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
          COMPONENT           SomeProj_RunTime
          NAMELINK_COMPONENT SomeProj_Development
  ARCHIVE DESTINATION ${CMAKE_INSTALL_LIBDIR}
          COMPONENT   SomeProj_Development
)
```

# EXAMPLE SCENARIO

(LINUX)

The `Example` shared library is the only thing consumers link to

The public API contains nothing from any internal implementation library



# EXAMPLE SCENARIO

(LINUX)

The `Example` shared library is the only thing consumers link to

The public API contains nothing from any internal implementation library

Build your libraries

Run your test apps against the libraries in your build tree

Package and install your libraries

Someone else builds against the installed libraries

They run the app they just built

# EXAMPLE SCENARIO

(LINUX)

The `Example` shared library is the only thing consumers link to

The public API contains nothing from any internal implementation library

Build your libraries

Run your test apps against the libraries in your build tree

Package and install your libraries

Someone else builds against the installed libraries

They run the app they just built

# EXAMPLE SCENARIO

(LINUX)

The `Example` shared library is the only thing consumers link to

The public API contains nothing from any internal implementation library

Build your libraries

Run your test apps against the libraries in your build tree

Package and install your libraries

Someone else builds against the installed libraries

They run the app they just built

# EXAMPLE SCENARIO

(LINUX)

The `Example` shared library is the only thing consumers link to

The public API contains nothing from any internal implementation library

Build your libraries

Run your test apps against the libraries in your build tree

Package and install your libraries

Someone else builds against the installed libraries

They run the app they just built

# EXAMPLE SCENARIO

(LINUX)

The `Example` shared library is the only thing consumers link to

The public API contains nothing from any internal implementation library

Build your libraries

Run your test apps against the libraries in your build tree

Package and install your libraries

Someone else builds against the installed libraries

They run the app they just built

# EXAMPLE SCENARIO

(LINUX)

The `Example` shared library is the only thing consumers link to

The public API contains nothing from any internal implementation library


Build your libraries


Run your test apps against the libraries in your build tree


Package and install your libraries


Someone else builds against the installed libraries


They run the app they just built

# EXAMPLE SCENARIO

(LINUX)

The `Example` shared library is the only thing consumers link to

The public API contains nothing from any internal implementation library


Build your libraries


Run your test apps against the libraries in your build tree


Package and install your libraries


Someone else builds against the installed libraries


They run the app they just built

```
./myapp: error while loading shared libraries: libVendorA.so.3: cannot open shared object file:
No such file or directory
```

# EXAMPLE SCENARIO

(LINUX)

Build your libraries

Run your test apps against the libraries in your build tree

Package and install your libraries

Someone else builds against the installed libraries

They run the app they just built

# EXAMPLE SCENARIO

(LINUX)

Build your libraries

Run your test apps against the libraries in your build tree

Package and install your libraries

Someone else builds against the installed libraries

They run the app they just built

CMake embedded **RPATH** information into the libraries and the app executable

# EXAMPLE SCENARIO

(LINUX)

Build your libraries

Run your test apps against the libraries in your build tree

Package and install your libraries

Someone else builds against the installed libraries

They run the app they just built

CMake embedded **RPATH** information into the libraries and the app executable

**RPATH** is supported on all major platforms except Windows

# EXAMPLE SCENARIO

(LINUX)

Build your libraries

Run your test apps against the libraries in your build tree

Package and install your libraries

Someone else builds against the installed libraries

They run the app they just built

CMake replaces the **RPATH** information it recorded for the build tree with a different set which is *empty by default*.

Libraries lose their **RPATH** connection to their dependencies unless you specify them.

# EXAMPLE SCENARIO

(LINUX)

Build your libraries

Run your test apps against the libraries in your build tree

Package and install your libraries

Someone else builds against the installed libraries

They run the app they just built

Because CMake again provides **RPATH**s for the build, dependency libraries in the same directory as libraries linked to the app will also be found at link time

# EXAMPLE SCENARIO

(LINUX)

Build your libraries

Run your test apps against the libraries in your build tree

Package and install your libraries

Someone else builds against the installed libraries

They run the app they just built

Behavior determined by entries in the binary's dynamic section:

- **DT_RPATH**
- **DT_RUNPATH**

If both are present, **DT_RPATH** is ignored.

# EXAMPLE SCENARIO

(LINUX)

Build your libraries

Run your test apps against the libraries in your build tree

Package and install your libraries

Someone else builds against the installed libraries

They run the app they just built

Behavior determined by entries in the binary's dynamic section:

- DT_RPATH
- DT_RUNPATH

LD_LIBRARY_PATH

If both are present, **DT_RPATH** is ignored.

# EXAMPLE SCENARIO

(LINUX)

Build your libraries

Run your test apps against the libraries in your build tree

Package and install your libraries

Someone else builds against the installed libraries

They run the app they just built

Behavior determined by entries in the binary's dynamic section:

- DT_RPATH
- DT_RUNPATH

If both are present, **DT_RPATH** is ignored.

```
ld.so man page:

...[DT_RUNPATH] are searched only to find
those objects required by DT_NEEDED (direct
dependencies) entries and do not apply to
those objects' children, which must themselves
have their own DT_RUNPATH entries. This is
unlike DT_RPATH, which is applied to searches
for all children in the dependency tree.
```

# Setting Install RPATH Details

```
if(NOT APPLE)
  set(CMAKE_INSTALL_RPATH $ORIGIN)
endif()

add_library(Example ...)
```

# Setting Install RPATH Details

```
if(NOT APPLE)
  set(CMAKE_INSTALL_RPATH $ORIGIN)
endif()

add_library(Example ...)
```

**$ORIGIN** means the location of the binary requiring the dependency

# Setting Install RPATH Details

```
if(NOT APPLE)
  set(CMAKE_INSTALL_RPATH $ORIGIN)
endif()

add_library(Example ...)
```

`$ORIGIN` means the location of the binary requiring the dependency

Apple has a similar feature, but uses different keywords (e.g. `@loader_path`)

- Checks environment variables first
- Recursive searching like DT_RPATH

# Questions?

- You can also catch me at tonight's **Tool Time Labs** for one-on-one discussions of your specific issues
- Consulting services available

GET IN TOUCH

https://crascit.com

@crascit

# Bonus Material

# Ensure Dependencies Are Found

```
find_package(SomeProj 2.3)
```

- SomeProjConfig.cmake
- SomeProjConfigVersion.cmake

```
find_dependency(...)

# See “Exporting Targets” slide (2 after this one)
include(${CMAKE_CURRENT_LIST_DIR}/SomeProj-Targets.cmake)
```

# Config File Location

```
include(GNUInstallDirs)
set(SomeProj_INSTALL_CMAKEDIR
    ${CMAKE_INSTALL_LIBDIR}/cmake/SomeProj
    CACHE STRING “Path to SomeProj cmake files”
)

install(FILES
    SomeProjConfig.cmake
    ${CMAKE_CURRENT_BINARY_DIR}/SomeProjConfigVersion.cmake
  DESTINATION ${SomeProj_INSTALL_CMAKEDIR}
)
```

# Exporting Targets

```
install(TARGETS Example
  EXPORT SomeProj_Targets
  INCLUDES DESTINATION ${CMAKE_INSTALL_INCLUDEDIR}
  ...   # Other lines as discussed in the main part of the talk
)
install(EXPORT SomeProj_Targets
  DESTINATION ${SomeProj_INSTALL_CMAKEDIR}   # See previous slide
  NAMESPACE   SomeProj::                     # See next slide
  FILE        SomeProj-Targets.cmake
  COMPONENT   SomeProj_Development
)
```

*

# Alias Library Matching Exported Target

```
add_library(SomeProj_Example ...)

# Exported target has SomeProj:: namespace prepended (see previous
# slide), so we drop the project-specific prefix from the exported name
set_target_properties(SomeProj_Example PROPERTIES
    EXPORT_NAME Example
)

# Create alias to match exported name of target, consuming projects can
# use that name whether they use find_package() or add_subdirectory()
add_library(SomeProj::Example ALIAS SomeProj_Example)
```

# Useful References

Symbol visibility

- https://gcc.gnu.org/wiki/Visibility
- http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/p0276r0.html
- http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/p1283r0.html

# Useful References

RPATH

- https://gms.tf/ld_library_path-considered-harmful.html
- https://developercommunity.visualstudio.com/idea/566616/support-rpath-for-binaries-during-development.html

Everything you ever wanted to know about shared libraries (and also things you didn't)

- https://akkadia.org/drepper/dsohowto.pdf