Deep CMake
For Library Authors

Craig Scott
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](https://crascit.com)
- [@crascit](https://twitter.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
API Control

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

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

Visual Studio hides symbols by default

GCC and Clang do NOT hide symbols by default
class __declspec(dllexport) MyGenerator
{
    public:
        int nextValue();
};
Visual Studio Visibility Control

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

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

#ifndef MYTGT_EXPORT
    #ifdef MyTgt_EXPORTS
        #define MYTGT_EXPORT __declspec(dllexport)
    #else
        #define MYTGT_EXPORT __declspec(dllimport)
    #endif
#endif
Visual Studio Visibility Control

```cpp
#include "mytgt_export.h"

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

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

Only define MyTgt_EXPORTS when building the library
GCC/Clang Visibility Control

- Change default visibility to hidden
  - `fvisibility=hidden`

- Change visibility of inlined code (including templates)
  - `fvisibility-inlines-hidden`
class __attribute__((visibility("default"))) MyGenerator
{
    public:
        int nextValue();
};
class MYTGT_EXPORT MyGenerator
{
 public:
  int nextValue();
};
GCC/Clang Visibility Control

```cpp
#include "mytgt_export.h"

class MYTGT_EXPORT MyGenerator
{
public:
    int nextValue();
};
```
#include “mytgt_export.h”

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

#ifndef MYTGT_EXPORT
#define MYTGT_EXPORT __attribute__((visibility("default")))
#endif
CMake Visibility Control

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

add_library(MyTgt ...)

include(GenerateExportHeader)
generate_export_header(MyTgt)
```
CMake Visibility Control

```cmake
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

```cpp
#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
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

Bug fix only
No API changes

Non-breaking additions to the API
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

- Breaking change
- Bug fix only
- No API changes
- Non-breaking additions to the API
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

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

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

<table>
<thead>
<tr>
<th>SONAME</th>
<th>REAL LIBRARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>libExample.so</td>
<td><strong>libExample.so.2.4.7</strong></td>
</tr>
<tr>
<td>libExample.so.2</td>
<td><strong>libExample.so.2.4.7</strong></td>
</tr>
<tr>
<td>libExample.so.2.4.7</td>
<td></td>
</tr>
</tbody>
</table>

Humans, packages
Shared Library Symlinks

Run-time loader  
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

Check with commands like `ldd` or `otool -L`
Shared Library Symlinks

<table>
<thead>
<tr>
<th>NAME LINK</th>
<th>SONAME</th>
<th>REAL LIBRARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>libExample.so</td>
<td>-&gt; libExample.so.2.4.7</td>
<td></td>
</tr>
<tr>
<td>libExample.so.2</td>
<td>-&gt; libExample.so.2.4.7</td>
<td></td>
</tr>
<tr>
<td>libExample.so.2.4.7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Shared Library Symlinks

- **Build-time linker**: NAME LINK
  - `libExample.so` -> `libExample.so.2.4.7`
  - `libExample.so.2` -> `libExample.so.2.4.7`
  - `libExample.so.2.4.7`

- **Run-time loader**: SONAME
  - `libExample.so`

- **Humans, packages**: REAL LIBRARY
  - `libExample.so.2.4.7`

Specified on linker command line as `-lExample`
Shared Library Symlinks

<table>
<thead>
<tr>
<th>Build-time linker</th>
<th>NAME LINK</th>
<th>libExample.so</th>
<th>-&gt;</th>
<th>libExample.so.2.4.7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run-time loader</td>
<td>SONAME</td>
<td>libExample.so.2</td>
<td>-&gt;</td>
<td>libExample.so.2.4.7</td>
</tr>
<tr>
<td>Humans, packages</td>
<td>REAL LIBRARY</td>
<td>libExample.so.2.4.7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**SONAME** is most critical from a compatibility perspective
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
```cpp
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
```
add_library(Example ...)  

set_target_properties(
    Example PROPERTIES
    SOVERSION 2
    VERSION 2.4.7
)
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

```cpp
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

```cpp
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

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

SONAME

Craig Scott  https://crascit.com  CppCon 2019
Library Versioning In CMake

```cpp
add_library(Example ...) 
set_target_properties(
    Example PROPERTIES
    SOVERSION 2
    VERSION 2.4.7
)
```

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

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

`libExample.so.2.4.7`

**SONAME**

**PROBABLY WRONG!**
Library Versioning In CMake

```cpp
add_library(Example ...)
set_target_properties(
    Example PROPERTIES
    SOVERSION 9
    VERSION 2.4.7
)
```

Independent SONAME
Library Versioning In CMake

```cpp
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

```cpp
add_library(Example ...) 
set_target_properties(
    Example PROPERTIES 
        SOVERSION 9 
        VERSION 2.4.7
    )
```

```cpp
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
    )
add_library(Example ...)

set_target_properties(
  Example PROPERTIES
    SOVERSION  2
    VERSION    2.4.7
)

Example.dll
Example.lib
add_library(Example ...)
set_target_properties(
  Example PROPERTIES
    SOVERSION 2
    VERSION  2.4.7
)

Example.dll
Example.lib

Acts like SONAME
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

```cpp
add_library(Example ...) 
set_target_properties(
  Example PROPERTIES
    SOVERSION  2
    VERSION    2.4.7
  )
```

Example.dll

Example.lib

Acts like SONAME

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
find_package(SomeProj 2.3)

- SomeProjConfig.cmake
- SomeProjConfigVersion.cmake
Package Versioning

```cmake
find_package(SomeProj 2.3)
```

- SomeProjConfig.cmake
- SomeProjConfigVersion.cmake
Package Versioning

```cmake
find_package(SomeProj 2.3)
```

- SomeProjConfig.cmake
- SomeProjConfigVersion.cmake
Package Versioning

```cmake
find_package(SomeProj 2.3)

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)
Installing Libraries With CMake

```cpp
include(GNUInstallDirs)
install(TARGETS Example
  RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
  LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
  ARCHIVE DESTINATION ${CMAKE_INSTALL_LIBDIR}
)
```
Installing Libraries With CMake

```cpp
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

```cpp
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

```cpp
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

```cpp
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

```cpp
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}
)
```

Requires at least CMake 3.14
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 COMPONENT Development
  ARCHIVE DESTINATION ${CMAKE_INSTALL_LIBDIR}
  COMPONENT Development
)
```

Requires at least CMake 3.12
Installing Libraries With CMake

```cpp
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

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

```cpp
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

```cpp
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

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

The public API contains nothing from any internal implementation library.

EXAMPLE SCENARIO

(P Linux)

(Public API)
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

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
The **Example** shared library is the only thing consumers link to.

The public API contains nothing from any internal implementation library.

**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
The public API contains nothing from any internal implementation library

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

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)

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

(_L I N U X_)  

The **Example** shared library is the only thing consumers link to.

The public API contains nothing from any internal implementation library.

<table>
<thead>
<tr>
<th>Build your libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run your test apps against the libraries in your build tree</td>
</tr>
<tr>
<td>Package and install your libraries</td>
</tr>
<tr>
<td>Someone else builds against the installed libraries</td>
</tr>
<tr>
<td><strong>X</strong> They run the app they just built</td>
</tr>
</tbody>
</table>
The Example shared library is the only thing consumers link to.

The public API contains nothing from any internal implementation library.

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

./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 RPATHs 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

(UNIX)

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

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

```cpp
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

```cpp
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
Bonus Material
Ensure Dependencies Are Found

find_package(SomeProj 2.3)

find_dependency(...)

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

)
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

- [http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/p0276r0.html](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](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)