navimatix
we are on IT

A BRIDLE FOR YOUR KITE

3.4.0

Best Practices for Downstream Development with Zephyr

Zephyr® Project Developer Summit 2023

June 26-30, 2023 | Prague, Czech Republic + Virtual

build with Sphinx, presented with Reveal.js

ABOUT US:

Tobias Kästner
tobias.kaestner@ul.com

Staff Engineer, UL Method Park GmbH
System & Software Architect Medical Devices
Agile Coach

Stephan Linz
stephan.linz@navimatix.de

Senior EOSS Engineer, Navimatix GmbH
Embeded Hard- and Software
Scrum Master

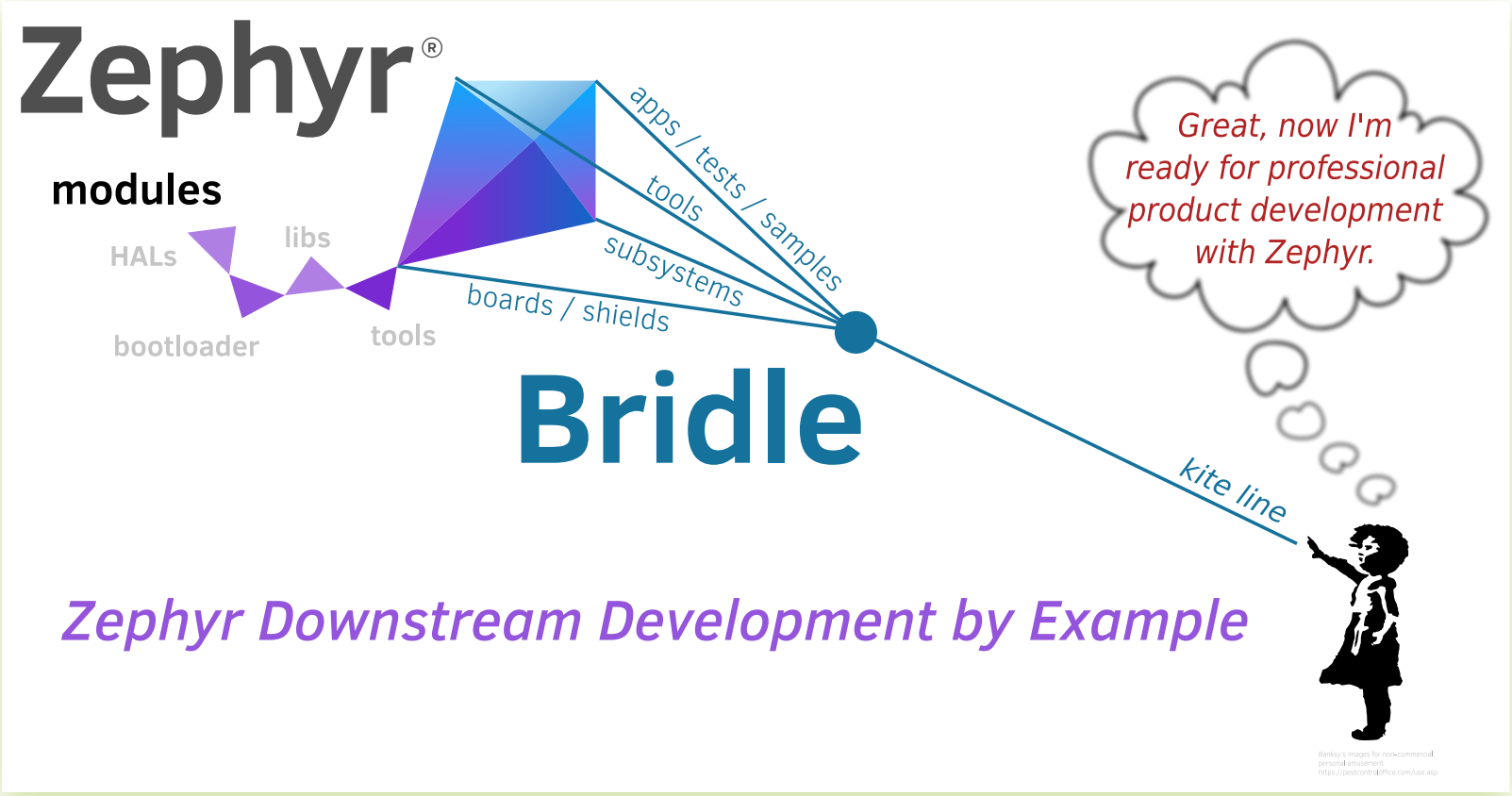
navimatix
we are on IT

WHAT WE DO

- Medical Device software development with (and without) Zephyr
- Connected Health Services development and Life Science Data Analytics
- Maintainers of the **Bridle** project
- Trying to establish Zephyr in the medical device domain
- Zephyr Trainings

#foss4medical

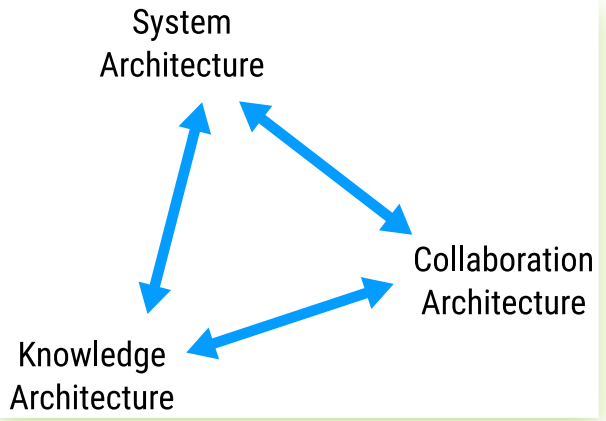
WHAT WE WANT TO TALK ABOUT



Overview what Bridle is

WHAT WE WANT TO TALK ABOUT

- **System architecture**
  - comprises SW & HW interfaces
  - structure, behavior, interfaces
- **Knowledge architecture**
  - big picture entire team knows about
  - propagation of new information
- **Collaboration architecture**
  - how work gets done
  - one's outputs are the other's input



Hopefully you didn't miss our second talk "How I Fell in Love with Zephyr" on Tuesday at 15:50-16:30

WHAT WE WANT TO TALK ABOUT

Architectural needs

- **Collaboration architecture**
  - Collaboration Models
  - Compliant Quality Assurance, e.g. IEC 62304
- **Knowledge architecture**
  - Product Documentation in all domains
  - Support Traceability in all domains, e.g. IEC 62304 or ISO 13485/14971
- **System architecture**
  - System Architecture Integration
  - Configuration Management (SW versions, Toolchain versions)

Bridle gives you a blueprint for

- **Project Logistic on GitHub with HIL**
  - loosely coupled with Zephyr (as module)
  - following your own Life-Cycle (RM)
- **Document Set per Version**
  - different perspectives
  - over multiple domains (potentially)
- **Source and Meta Code**
  - extends the framework (West / CMake / Kconfig)
  - expands the code base (ARCH / SOC / DTS / boards / drivers / tests / apps)

Let’s start with a bit of history for warm up.

A LITTLE BRIEF OF HISTORY

WHERE BRIDLE COMES FROM

2020

Started with a **Proof-Of-Concept on local machine**, inspirited by Nordic's *nRF Connect SDK v1.2*.

Spring 2021

Went to GitHub with a personal project – the project name **Bridle** was born – and established the **GitHub workflow** by reusing best practices (as Zephyr and Nordic did and does).

Note:

Zephyr Example Application was created too.

Summer 2021

Bridle moved on GitHub to a **more collaborative workspace** – the virtual organization **TiaC Systems** was born – and established the central web space for **online documentation** and use Zephyr from an **auto-sync Git mirror**.

First version, **Bridle v2.5.0**, was released.

TiaC Three is (a) Company – based on Tolkien's *The Fellowship of the Ring*

WHERE BRIDLE IS TODAY

since 2021 – MAINTENANCE

- following Zephyr's major and minor releases in V2 and V3
- continuous synchronization with Zephyr upstream regarding tooling (framework)
- closely following the progress in Nordic's *nRF Connect SDK* and the *Zephyr Example Application*

2022 – FOUNDING

- growing up the **project logistics** on GitHub (later more).
- refine Nordic's idea of **doc-sets** (later more).
- start to establish and document Bridle's *Development Model*
- define a first rudimentary *Contribution Workflow*

2023 – CONTENT – Bridle v3.4

- *improving*: support of **new boards and shields**, currently not supported in Zephyr upstream
- *feasibility*: transpose **system architectural definitions** by using **DeviceTree Nexus Nodes** and **Stacked Shields**
- *feasibility*: create **shield test suites** on CMake level
- *teaching*: Bridle is now part in trainings and daily work with colleagues, partners and students

PROJECT LOGISTICS

PROJECT LOGISTICS

(RECALL)

Architectural needs

Bridle gives you a blueprint for

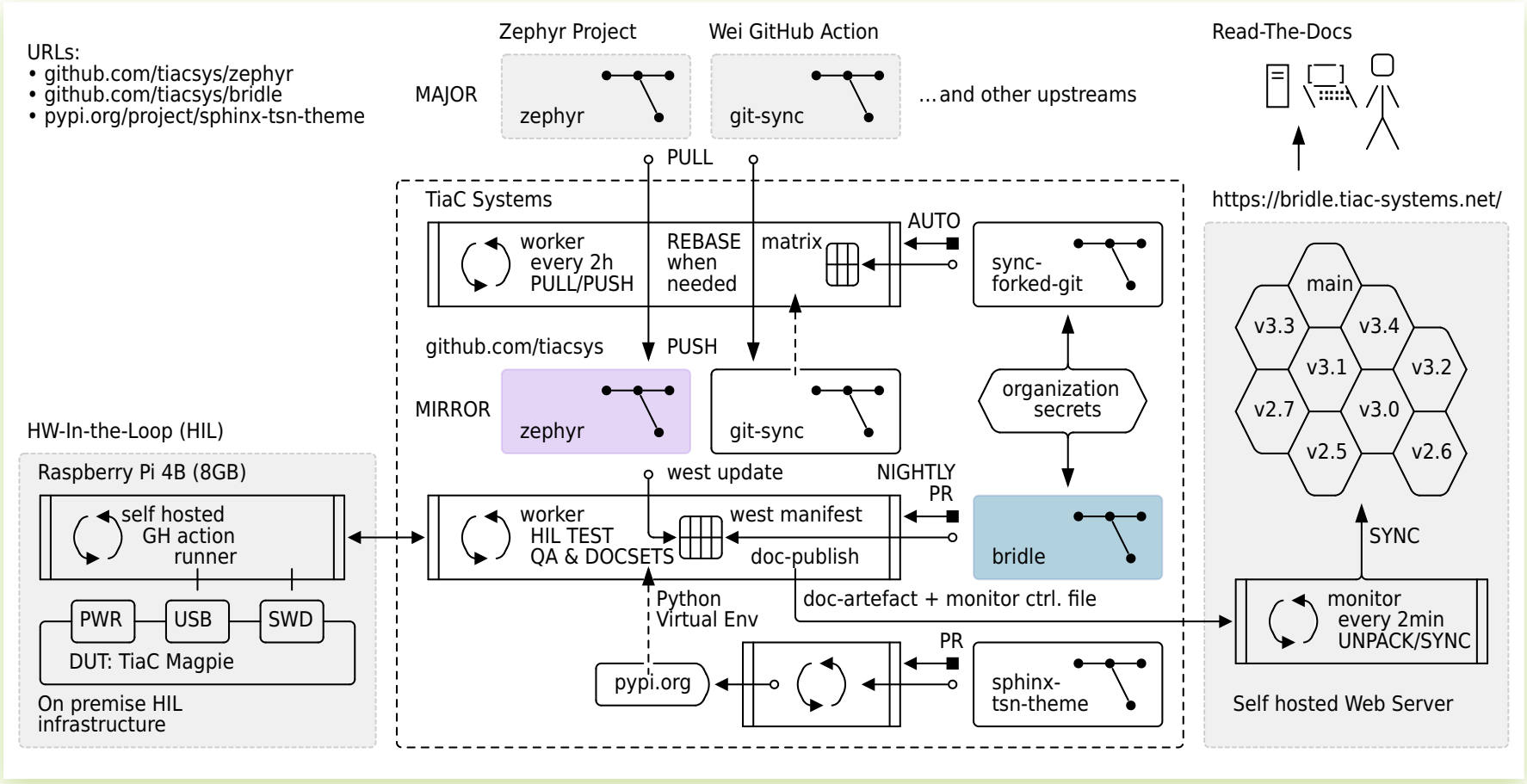
Collaboration architecture

- *Collaboration Models*
- *Compliant Quality Assurance, e.g. IEC 62304*

Project Logistic on GitHub with HIL

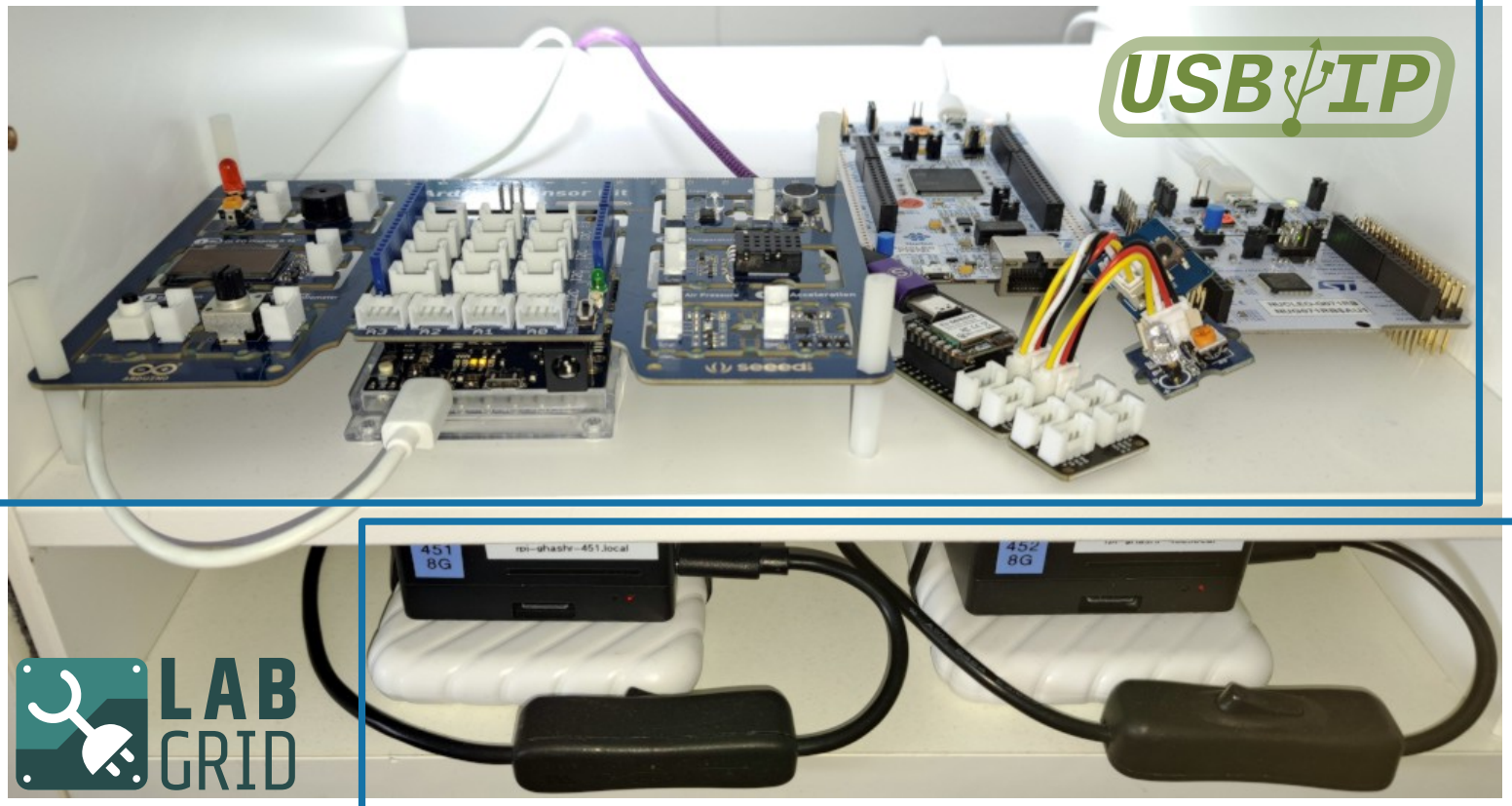
- *loosely coupled with Zephyr (as module)*
- *following your own Life-Cycle (RM)*

PROJECT LOGISTICS ON GITHUB



ON PREMISE HIL INFRASTRUCTURE

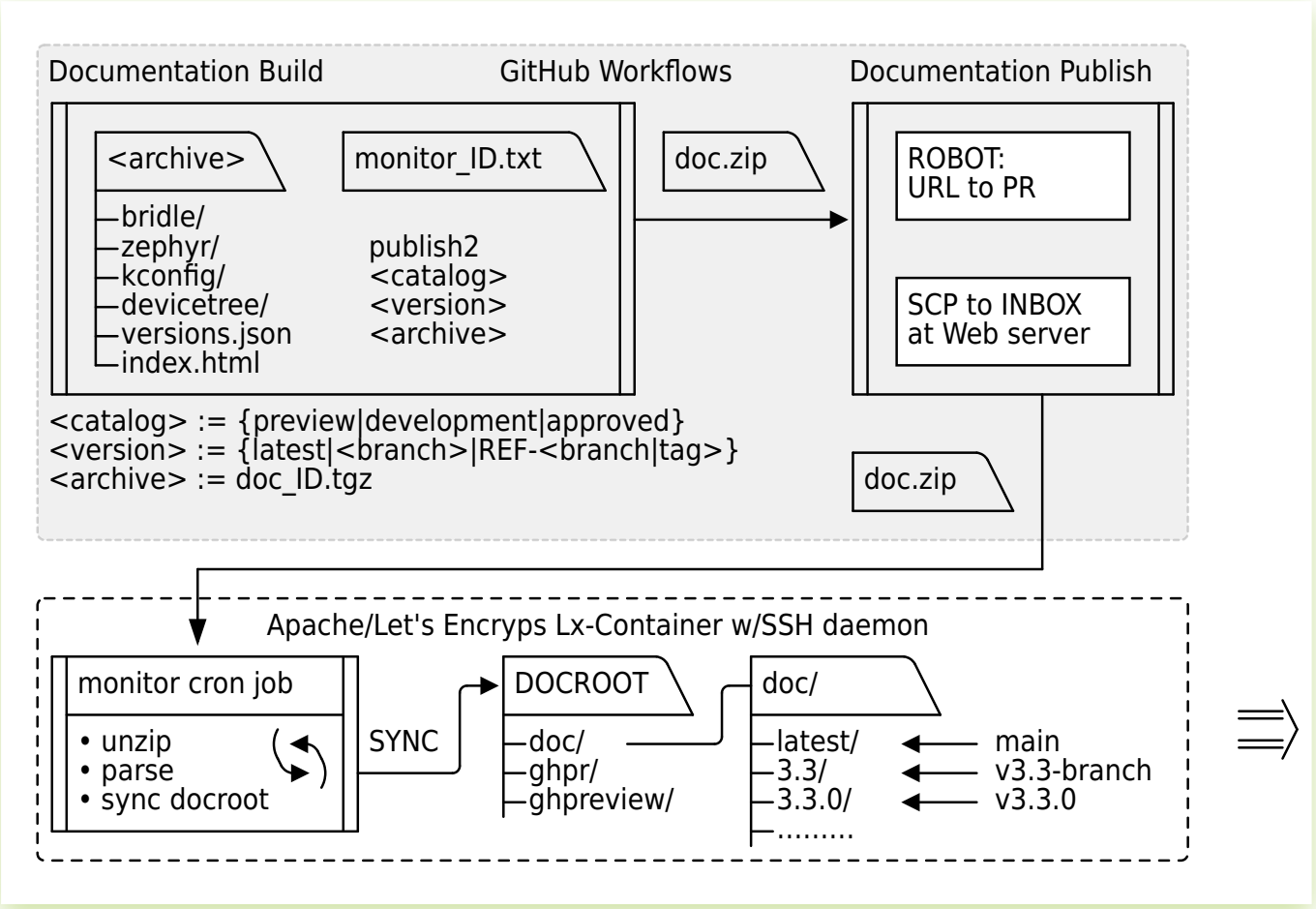
Devices Under Test

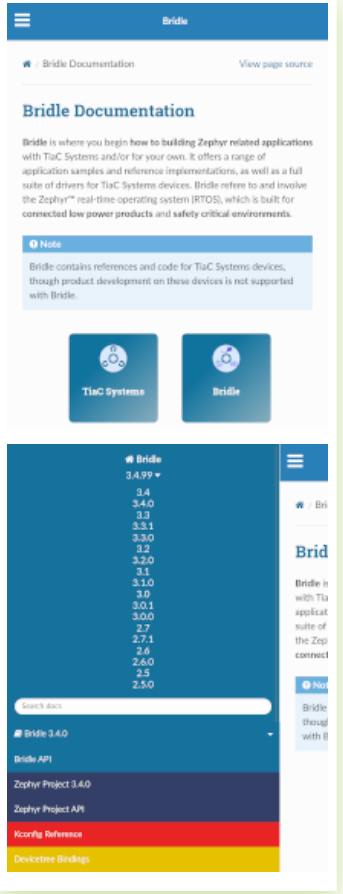


Raspberry Pi 4B (redundant)

Next generation – **currently work in progress** – will use *LabGRID* and *USB/IP*.

SELF HOSTED WEB SERVER





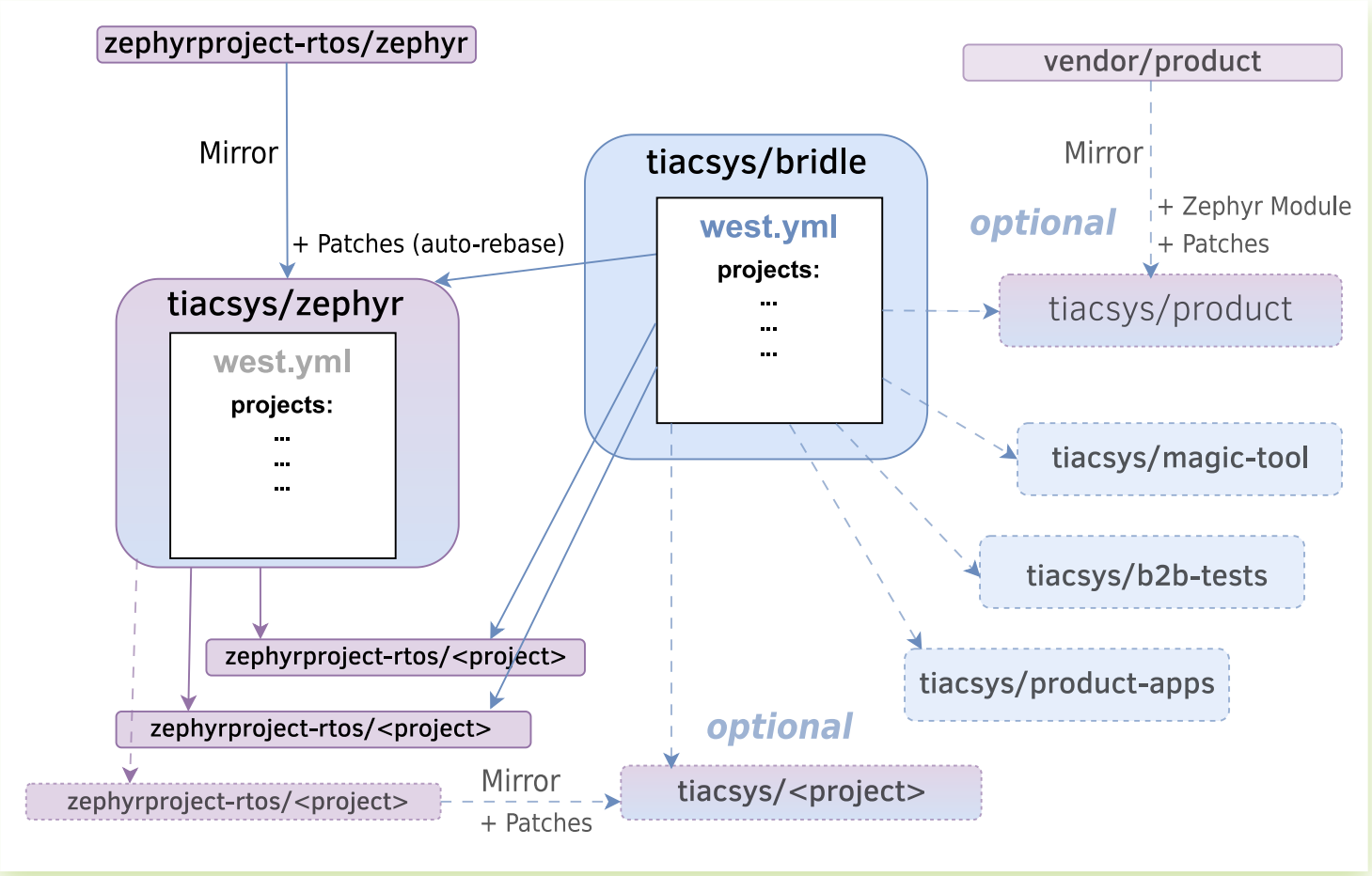
build https://github.com/tiacsys/bridle/blob/main/.github/workflows/doc-build.yml
publish https://github.com/tiacsys/bridle/blob/main/.github/workflows/doc-publish.yml
monitor https://bridle.tiac-systems.net/.inbox/.note-monitor.txt

BRIDLE HAS ITS OWN WEST MANIFEST

https://bridle.tiac-systems.net/doc/3.4/bridle/dm_code_base.html#repository-structure

Bridle v3.4 West Manifest: https://github.com/tiacsys/bridle/blob/v3.4-branch/west.yml

Repository structure → Software Conficuration Management



T2 Star topology in the west documentation – application is the manifest repository.

WEST WORKSPACE SETUP FROM BRIDLE

https://bridle.tiac-systems.net/doc/3.4/zephyr/develop/west/manifest.html

Bridle v3.4 West Manifest: https://github.com/tiacsys/bridle/blob/v3.4-branch/west.yml

West Manifest part for Bridle itself

```
manifest:
  version: "0.13"
  defaults:
    remote: tiacsys
  remotes:
    - name: tiacsys
      url-base: https://github.com/tiacsys
  self:
    path: bridle
    import: submanifests
    west-commands: scripts/west-commands.yml
```

West Manifest part to other projects

```
manifest:
  version: "0.13"

  projects:
    - name: zephyr
      path: zephyr
      remote: tiacsys
      repo-path: zephyr
      revision: tiacsys/v3.4-branch
      clone-depth: 5000
      userdata:
        west-commands-path: scripts/west_commands
```

West Manifest part to nested projects (1)

```
manifest:
  version: "0.13"

  projects:
    - name: zephyr

      import:
        name-allowlist:
          - canopennode
          - chre
          - cmsis
          - edtt
          - fatfs
          - hal_altera
          - hal_atmel
          - hal_espressif
          - hal_nordic
          - hal_nxp
          - hal_rpi_pico
          - hal_st
          - hal_stm32
          - hal_xtensa
```

West Manifest part to nested projects (2)

```
manifest:
  version: "0.13"

  projects:
    - name: zephyr

      import:
        name-allowlist:

          - libmetal
          - liblc3
          - littlefs
          - loramac-node
          - lvgl
          - mbedtls
          - mipi-sys-t
          - net-tools
          - open-amp
          - openthread
          - picolibc
          - segger
          - tinycrypt
```

WEST WORKSPACE SETUP FROM BRIDLE

Bridle v3.4 West Manifest: https://github.com/tiacsys/bridle/blob/v3.4-branch/west.yml

West Manifest from Bridle

```
manifest:
  version: "0.13"
  defaults:
    remote: tiacsys
  remotes:
    - name: tiacsys
      url-base: https://github.com/tiacsys
  self:
    path: bridle
    import: submanifests
    west-commands: scripts/west-commands.yml
  projects:
    - name: zephyr
      path: zephyr
      remote: tiacsys
      repo-path: zephyr
      revision: tiacsys/v3.4-branch
      clone-depth: 5000
      userdata:
        west-commands-path: scripts/west_commands
      import:
        name-allowlist:
          - canopennode
          - chre
          - cmsis
          - edtt
          - fatfs
          - hal_altera
          - hal_atmel
          - hal_espressif
          - hal_nordic
          - hal_nxp
          - hal_rpi_pico
          - hal_st
          - hal_stm32
          - hal_xtensa
          - libmetal
          - liblc3
          - littlefs
          - loramac-node
          - lvgl

          - picolibc
          - segger
          - tinycrypt
```

in a local workspace folder with activated Python Virtual Environment:

Workspace INIT: `west init -m https://github.com/tiacsys/bridle --mr v3.4-branch`

Bridle itself

```
manifest:
  defaults:
    remote: tiacsys
  remotes:
    - name: tiacsys
      url-base: https://github.com/tiacsys
  self:
    path: bridle
    import: submanifests
```

Local West Workspace:

```
workspace/
├── .venv/
├── .west/
│   └── config
├── bridle/
└── ├── west.yml
    ├── submanifests/
    └── zephyr/module.yml
```

```
workspace/.west/config
   [manifest]
   path = bridle
   file = west.yml
```

submanifests/

- Example 2.1: Downstream of a Zephyr release with explicit path
- Example 2.2: Downstream with directory of manifest files
- Example 2.3: Continuous Integration overrides

WEST WORKSPACE SETUP FROM BRIDLE

Bridle v3.4 West Manifest: https://github.com/tiacsys/bridle/blob/v3.4-branch/west.yml

West Manifest from Bridle

```yaml
manifest:
  version: "0.13"
  defaults:
    remote: tiacsys
  remotes:
    - name: tiacsys
      url-base: https://github.com/tiacsys
  self:
    path: bridle
    import: submanifests
    west-commands: scripts/west-commands.yml
  projects:
    - name: zephyr
      path: zephyr
      remote: tiacsys
      repo-path: zephyr
      revision: tiacsys/v3.4-branch
      clone-depth: 5000
      userdata:
        west-commands-path: scripts/west_commands
      import:
        name-allowlist:
          - canopennode
          - chre
          - cmsis
          - edtt
          - fatfs
          - hal_altera
          - hal_atmel
          - hal_espressif
          - hal_nordic
          - hal_nxp
          - hal_rpi_pico
          - hal_st
          - hal_stm32
          - hal_xtensa
          - libmetal
          - liblc3
          - littlefs
          - loramac-node
          - lvgl

          - picolibc
          - segger
          - tinycrypt
```

in a local workspace folder with activated Python Virtual Environment and initialized West Workspace:

Workspace UPDATE: `west update`

Zephyr and Components

```yaml
manifest:
  defaults:
    remote: tiacsys
  remotes:
    - name: tiacsys
      url-base: https://github.com/tiacsys
  projects:
    - name: zephyr
      path: zephyr
      remote: tiacsys
      repo-path: zephyr
      revision: tiacsys/v3.4-branch
      import:
        name-allowlist:
          - canopennode
          - chre
          - cmsis
          - edtt
          - fatfs
          - hal_altera
          - hal_atmel
          - hal_espressif
          - hal_nordic
          - hal_nxp
          - hal_rpi_pico
          - hal_st
          - hal_stm32
          - hal_xtensa

          - picolibc
          - segger
          - tinycrypt
```

Local West Workspace:

```
workspace/
├── .venv/
├── .west/
├── bridle/
├── modules/
│   ├── crypto/
│   ├── debug/
│   ├── fs/
│   ├── hal/
│   └── lib/
├── tools/
│   ├── edtt/
│   └── net-tools/
└── zephyr/
```

```
workspace/bridle/zephyr/module.yml

workspace/modules/crypto/
├── mbedtls/zephyr/module.yml
└── tinycrypt/zephyr/module.yml

workspace/modules/debug/
├── mipi-sys-t/zephyr/module.yml
└── segger/zephyr/module.yml

workspace/modules/fs/
├── fatfs/zephyr/module.yml
└── littlefs/zephyr/module.yml
```

```
workspace/modules/hal/
├── altera/zephyr/module.yml
├── atmel/zephyr/module.yml
├── cmsis/zephyr/module.yml
├── espressif/zephyr/module.yml
├── libmetal/zephyr/module.yml
├── nordic/zephyr/module.yml
├── nxp/zephyr/module.yml
├── rpi_pico/zephyr/module.yml
├── st/zephyr/module.yml
├── stm32/zephyr/module.yml
└── xtensa/zephyr/module.yml
```

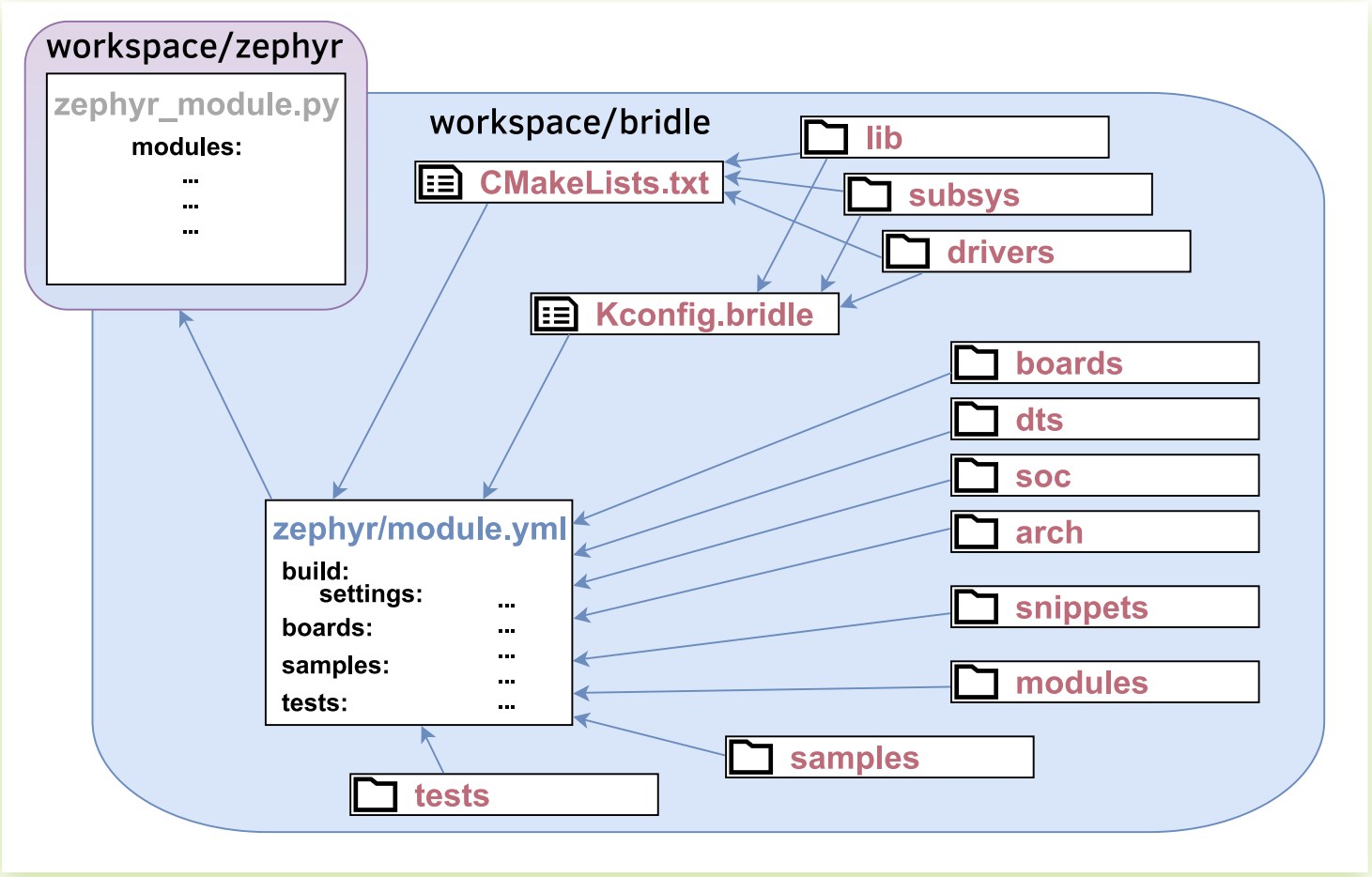
```
workspace/modules/lib/
├── canopennode/zephyr/module.yml
├── chre/zephyr/module.yml
├── gui/lvgl/zephyr/module.yml
├── liblc3/zephyr/module.yml
├── loramac-node/zephyr/module.yml
├── open-amp/zephyr/module.yml
├── openthread/zephyr/module.yml
└── picolibc/zephyr/module.yml
```

BRIDLE IS A ZEPHYR MODULE

https://bridle.tiac-systems.net/doc/3.4/zephyr/develop/modules.html

Bridle v3.4 Zephyr Module: https://github.com/tiacsys/bridle/blob/v3.4-branch/zephyr/module.yml

Workspace structure → Zephyr Expansion



The **SCA tools** implementation or **Binary Blobs** are **not yet used** by Bridle.

ZEPHYR ENTRY POINTS INTO BRIDLE

https://bridle.tiac-systems.net/doc/3.4/zephyr/develop/modules.html

Bridle v3.4 Zephyr Module: https://github.com/tiacsys/bridle/blob/v3.4-branch/zephyr/module.yml

Integration Files

```
build:
  cmake: .
  kconfig: Kconfig.bridle
```

Hint:

Integration Files are **recommended**!

Build Settings

```
build:
  settings:
    board_root: .
    dts_root: .
    soc_root: .
    arch_root: .
    snippet_root: .
    module_ext_root: .
```

Twister Directories

```
build:
  cmake: .
boards:
  - boards
samples:
  - samples
tests:
  - tests
```

Binary Blobs

```
blobs:
  - path: lib/libhonkomat.a
    sha256: deadbeef
    type: lib
    version: '1.0'
```

Zephyr module dependencies are not yet used by Bridle, e.g.:

```
build:
  depends:
    - zephyr
    - hal_nxp
    - littlefs
    - …
```

The SCA tool implementation is not yet used by Bridle, e.g.:

```
build:
  settings:
    sca_root: .
```

Only useful when Zephyr's GitHub workflow will be used, see the multiple steps **Run tests with twister** in: zephyr/v3.4.0/.github/workflows/twister.yaml

```
./zephyr/scripts/zephyr_module.py \
    --twister-out module_tests.args
```

Example only! Binary Blobs are not yet used by Bridle.

ZEPHYR ENTRY POINTS INTO BRIDLE

https://bridle.tiac-systems.net/doc/3.4/zephyr/build/cmake/index.html

Bridle v3.4 Zephyr Module: https://github.com/tiacsys/bridle/blob/v3.4-branch/zephyr/module.yml

Zephyr Module file in Bridle

```
build:
  cmake: .
  kconfig: Kconfig.bridle
  settings:
    board_root: .
    dts_root: .
    soc_root: .
    arch_root: .
    snippet_root: .
    module_ext_root: .
boards:
  - boards
samples:
  - samples
tests:
  - tests
```

CMake expansions through Zephyr Module **Integration Files**.

Integration Files

```
build:
  cmake: .
  kconfig: Kconfig.bridle
```

Bridle source code folder in local West Workspace:

```
workspace/bridle/
├── include/
├── VERSION
├── version.h.in
├── CMakeLists.txt
└── Kconfig.bridle
```

```
workspace/bridle/subsys/CMakeLists.txt
workspace/bridle/subsys/Kconfig
workspace/bridle/drivers/CMakeLists.txt
workspace/bridle/drivers/Kconfig
workspace/bridle/lib/CMakeLists.txt
workspace/bridle/lib/Kconfig
```

Bridle v3.4 CMakeLists.txt: https://github.com/tiacsys/bridle/blob/v3.4-branch/CMakeLists.txt

```
add_custom_target(version_bridle_h
  DEPENDS ${PROJECT_BINARY_DIR}/include/generated/version_bridle.h)
add_dependencies(version_bridle_h version_h)
add_dependencies(zephyr_interface version_bridle_h)

zephyr_include_directories(include)

# Lib is placed early because it defines important common supports
add_subdirectory(lib)
add_subdirectory(subsys)
add_subdirectory(drivers)
```

ZEPHYR ENTRY POINTS INTO BRIDLE

https://bridle.tiac-systems.net/doc/3.4/zephyr/build/kconfig/index.html

Bridle v3.4 Zephyr Module: https://github.com/tiacsys/bridle/blob/v3.4-branch/zephyr/module.yml

Zephyr Module file in Bridle

```
build:
  cmake: .
  kconfig: Kconfig.bridle
  settings:
    board_root: .
    dts_root: .
    soc_root: .
    arch_root: .
    snippet_root: .
    module_ext_root: .
boards:
  - boards
samples:
  - samples
tests:
  - tests
```

Kconfig expansions through Zephyr Module **Integration Files**.

Integration Files

```
build:
  cmake: .
  kconfig: Kconfig.bridle
```

Bridle source code folder in local West Workspace:

```
workspace/bridle/
├── include/
├── VERSION
├── version.h.in
├── CMakeLists.txt
└── Kconfig.bridle
```

```
workspace/bridle/subsys/CMakeLists.txt
workspace/bridle/subsys/Kconfig
workspace/bridle/drivers/CMakeLists.txt
workspace/bridle/drivers/Kconfig
workspace/bridle/lib/CMakeLists.txt
workspace/bridle/lib/Kconfig
```

Bridle v3.4 Kconfig.bridle: https://github.com/tiacsys/bridle/blob/v3.4-branch/Kconfig.bridle

```
mainmenu "Bridle"

rsource "subsys/Kconfig"
rsource "drivers/Kconfig"
rsource "lib/Kconfig"
```

ZEPHYR ENTRY POINTS INTO BRIDLE

Bridle v3.4 Zephyr Module: https://github.com/tiacsys/bridle/blob/v3.4-branch/zephyr/module.yml

Zephyr Module file in Bridle

```
build:
  cmake: .
  kconfig: Kconfig.bridle
  settings:
    board_root: .
    dts_root: .
    soc_root: .
    arch_root: .
    snippet_root: .
    module_ext_root: .
boards:
  - boards
samples:
  - samples
tests:
  - tests
```

Build System expansions through Zephyr Module **Build Settings**.

Build Settings

```
build:
  settings:
    board_root: .
    dts_root: .
    soc_root: .
    arch_root: .
    snippet_root: .
    module_ext_root: .
```

Bridle source code folder in local West Workspace:

workspace/**bridle**/	workspace/**bridle**/**boards**/<arch>/<board>/*
├── **boards**/	workspace/**bridle**/**boards**/**shields**/<shield>/*
├── **dts**/	workspace/**bridle**/**dts**/<arch>/*
├── **soc**/	workspace/**bridle**/**dts**/**bindings**/*
├── **arch**/	workspace/**bridle**/**soc**/<arch>/<soc>/*
├── **snippets**/	workspace/**bridle**/**arch**/<arch>/*
└── **modules**/	workspace/**bridle**/**snippets**/<snippet>/*
	workspace/**bridle**/**modules**/**modules.cmake**
This **folder names**	workspace/**bridle**/**modules**/<module>/CMakeLists.txt
are all **mandatory**!	workspace/**bridle**/**modules**/<module>/Kconfig

boards/<arch>/<board>/*	https://bridle.tiac-systems.net/doc/3.4/zephyr/hardware/porting/board_porting.html
boards/**shields**/<shield>/*	https://bridle.tiac-systems.net/doc/3.4/zephyr/hardware/porting/shields.html
dts/<arch>/*	https://bridle.tiac-systems.net/doc/3.4/zephyr/build/dts/index.html
dts/**bindings**/*	https://bridle.tiac-systems.net/doc/3.4/zephyr/build/dts/bindings.html
soc/<arch>/<soc>/*	https://bridle.tiac-systems.net/doc/3.4/zephyr/hardware/porting/board_porting.html#soc
arch/<arch>/*	https://bridle.tiac-systems.net/doc/3.4/zephyr/hardware/porting/arch.html
snippets/<snippet>/*	https://bridle.tiac-systems.net/doc/3.4/zephyr/build/snippets/index.html
modules/**modules.cmake**	https://bridle.tiac-systems.net/doc/3.4/zephyr/develop/modules.html#modules-module-ext-root

DOCUMENT SETS

DOCUMENT SETS

(RECALL)

Architectural needs

Bridle gives you a blueprint for

Knowledge architecture

- *Product Documentation in all domains*
- *Support Traceability in all domains, e.g. IEC 62304 or ISO 13485/14971*

Document Set per Version

- *different perspectives*
- *over multiple domains (potentially)*

BRIDLE HAS ITS OWN DOCUMENT SET

https://bridle.tiac-systems.net/doc/3.4/bridle/doc_structure.html

Bridle
3.4.99

Search docs

CONTENTS:

- About Bridle
- About this documentation
  - Documentation structure
    - Documentation search
    - Different set versions
  - Building Bridle documentation
  - Documentation style guide
  - Documentation templates
- Getting started
- User guides
- Applications and samples
- Supported Boards
- Supported Snippets
- API Reference
- Release Notes

Bridle 3.4.0

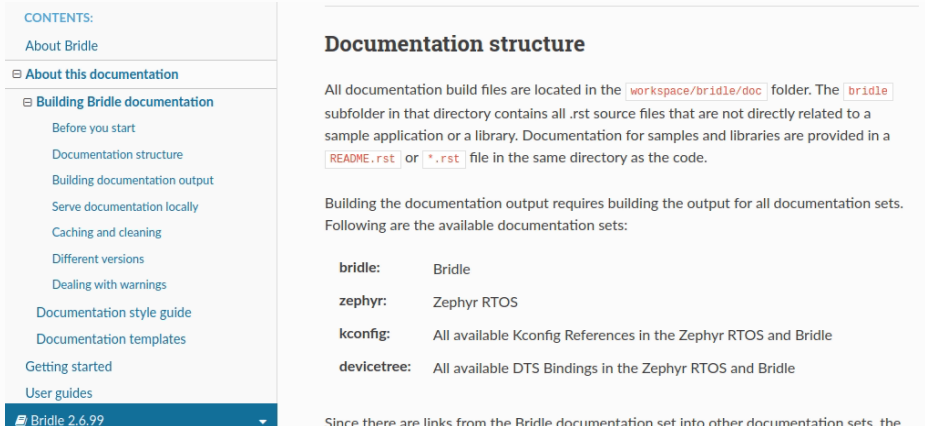
/ About this documentation / Documentation structure

View page source

Documentation structure

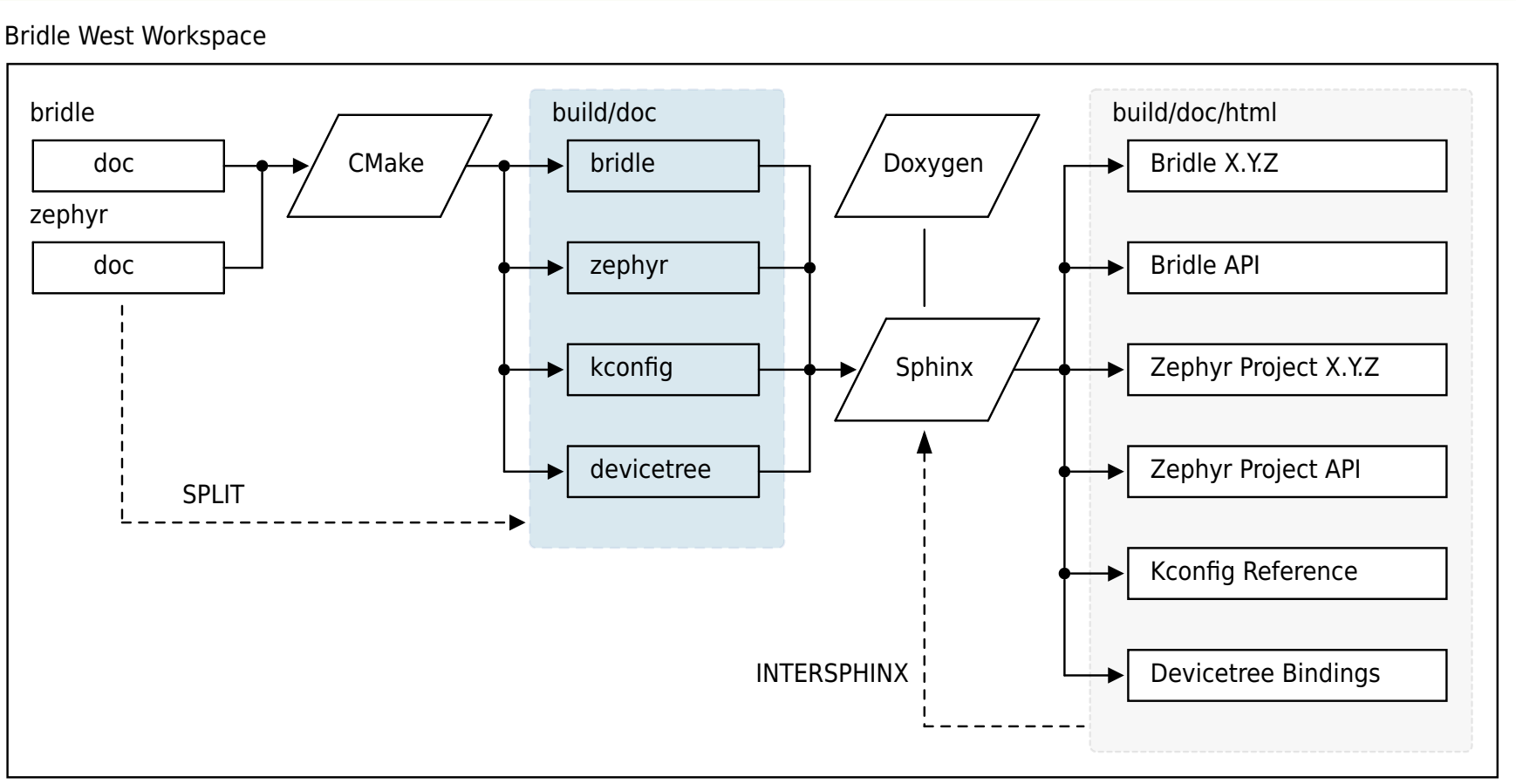
The documentation consists of several inter-linked documentation sets, one for each repository. You can switch between these documentation sets by using the selector in the *bottom-left corner of each page*.

The entry point is the Bridle documentation that you are currently reading. The local Zephyr documentation is a slightly extended version of the official Zephyr Project documentation, containing some additions specific to TiaC Systems.



BUILDING BRIDLE DOCUMENTATION

https://bridle.tiac-systems.net/doc/3.4/bridle/doc_build.html



DIVING DEEPER INTO BRIDLE

DIVING DEEPER INTO BRIDLE

(RECALL)

Architectural needs

Bridle gives you a blueprint for

System architecture

- *System Architecture Integration*
- *Configuration Management (SW versions, Toolchain versions)*

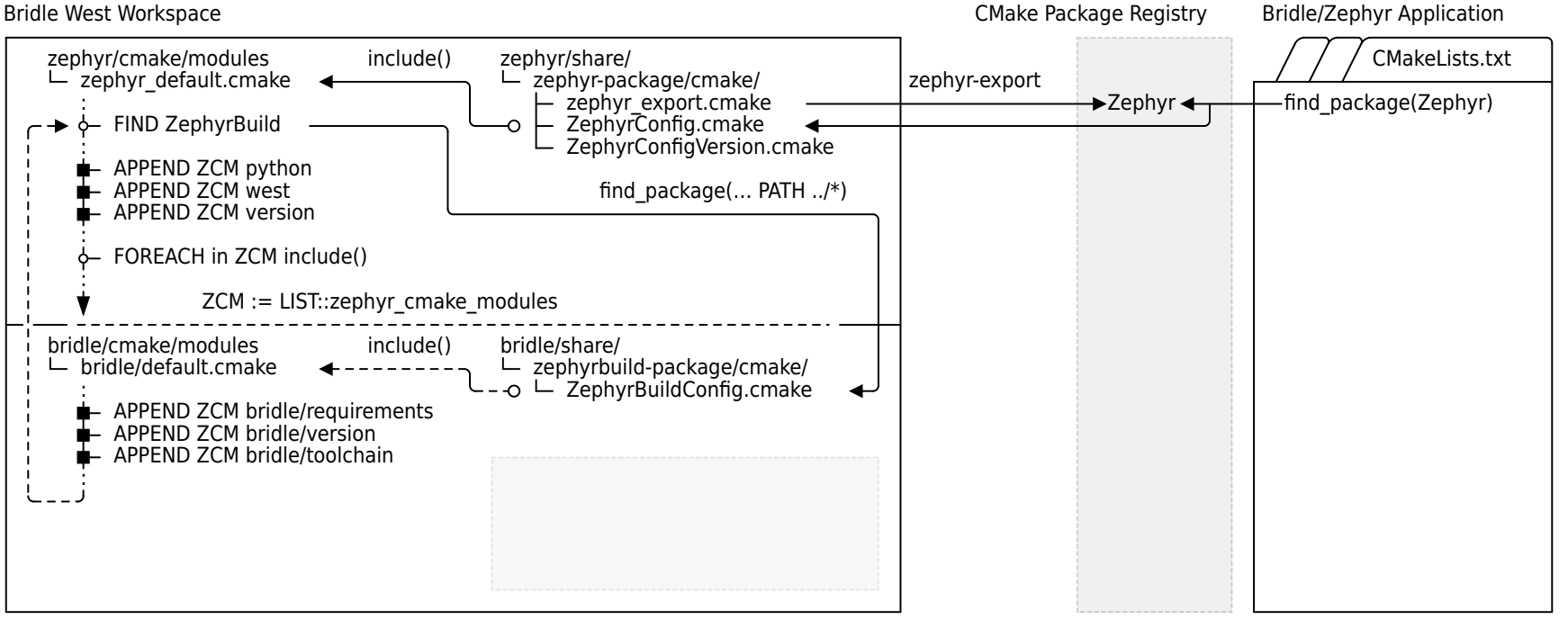
Source and Meta Code

- *extends the framework (West / CMake / Kconfig)*
- *expands the code base (ARCH / SOC / DTS / boards / drivers / tests / apps)*

BRIDLE HAS ITS OWN CMAKE PACKAGE

https://cmake.org/cmake/help/book/mastering-cmake/chapter/Finding%20Packages.html

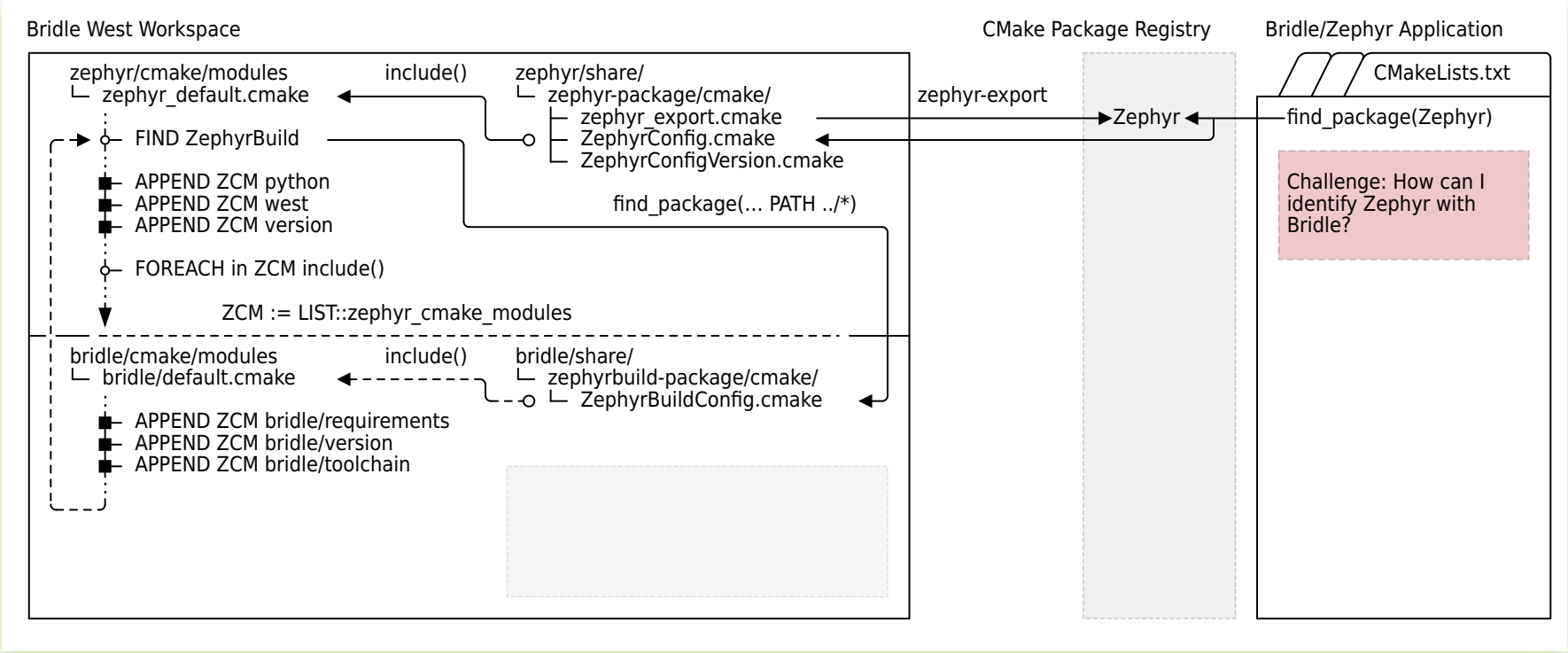
Zephyr v3.4 CMake Package: https://bridle.tiac-systems.net/doc/3.4/zephyr/build/zephyr_cmake_package.html



BRIDLE HAS ITS OWN CMAKE PACKAGE

https://cmake.org/cmake/help/book/mastering-cmake/chapter/Finding%20Packages.html

Zephyr v3.4 CMake Package: https://bridle.tiac-systems.net/doc/3.4/zephyr/build/zephyr_cmake_package.html

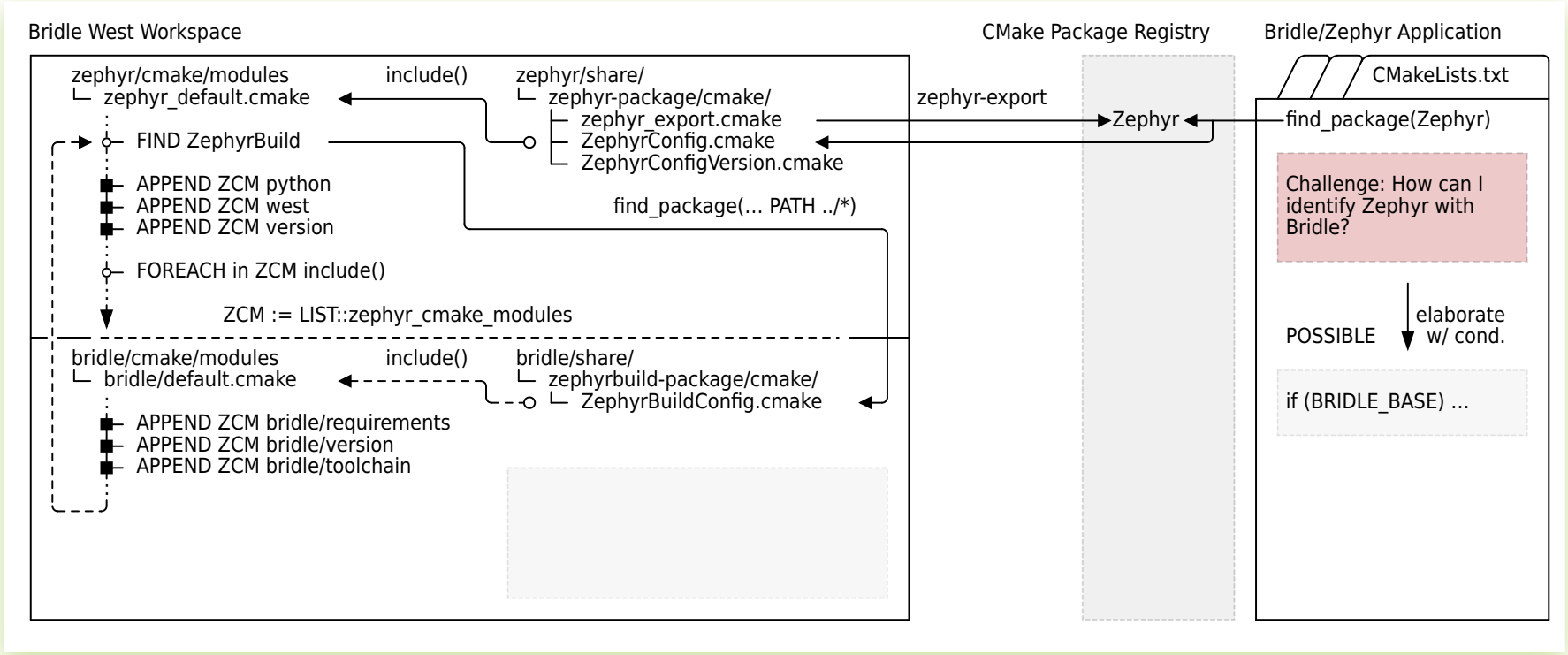


Challenge: How can we identify the **correct workspace**, `based on Bridle`?

BRIDLE HAS ITS OWN CMAKE PACKAGE

https://cmake.org/cmake/help/book/mastering-cmake/chapter/Finding%20Packages.html

Zephyr v3.4 CMake Package: https://bridle.tiac-systems.net/doc/3.4/zephyr/build/zephyr_cmake_package.html

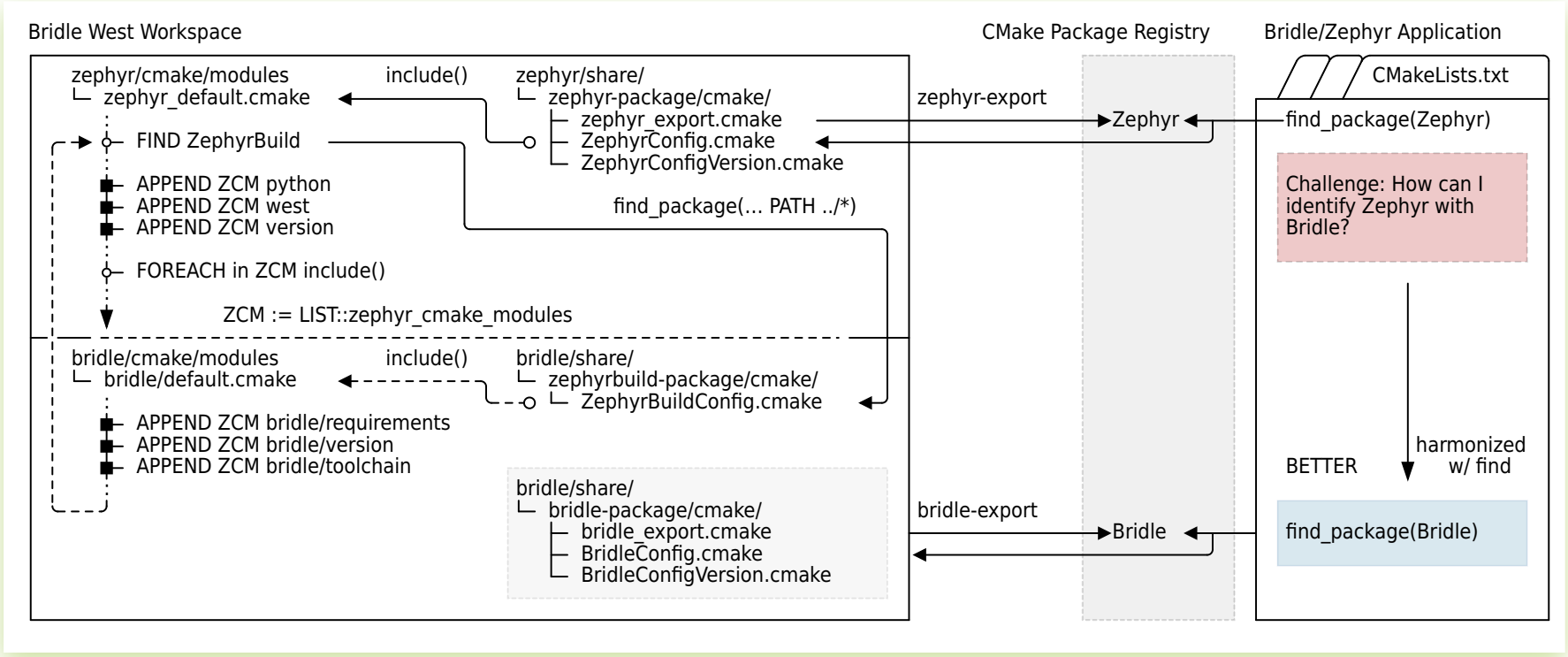


Challenge: How can we identify the **correct workspace**, `based on Bridle`?

BRIDLE HAS ITS OWN CMAKE PACKAGE

https://cmake.org/cmake/help/book/mastering-cmake/chapter/Finding%20Packages.html

Zephyr v3.4 CMake Package: https://bridle.tiac-systems.net/doc/3.4/zephyr/build/zephyr_cmake_package.html

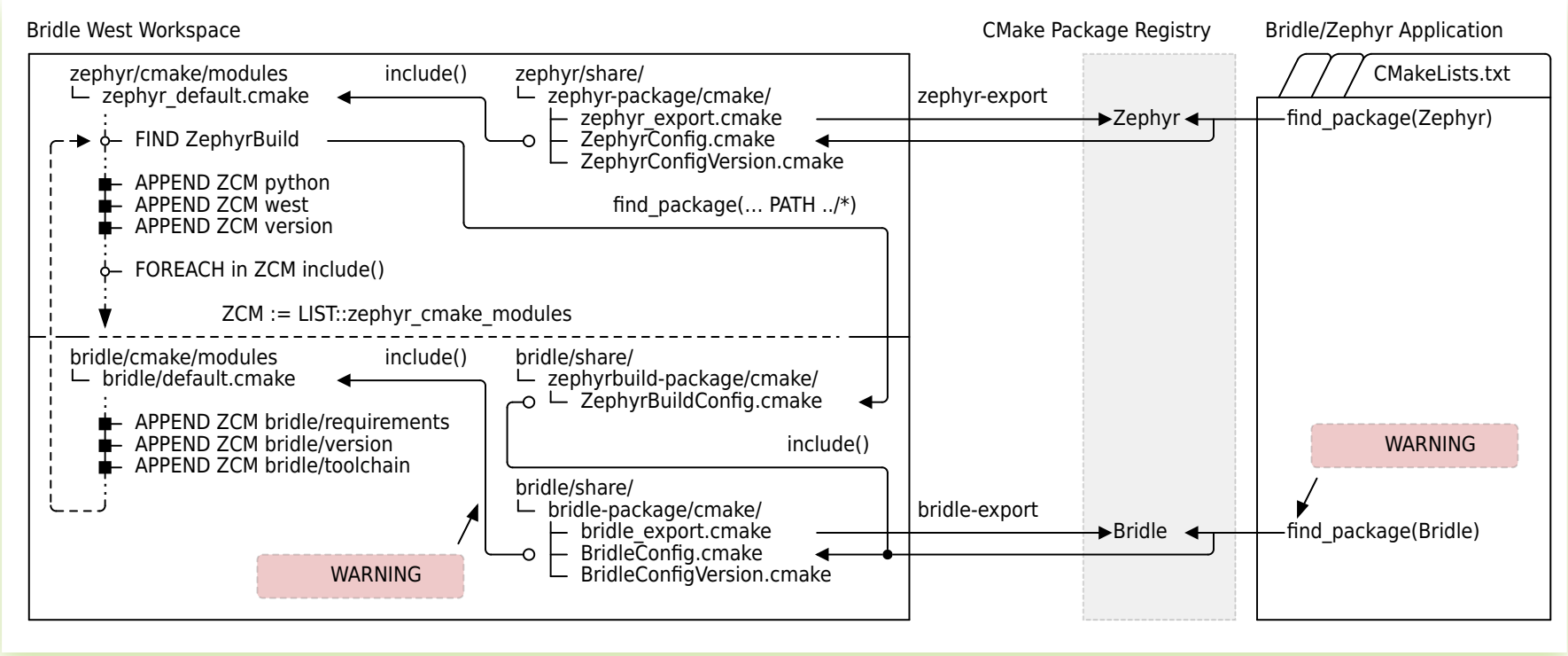


Solution: Provide our own CMake Package and use it.

BRIDLE HAS ITS OWN CMAKE PACKAGE

https://cmake.org/cmake/help/book/mastering-cmake/chapter/Finding%20Packages.html

Bridle v3.4 CMake Package: https://bridle.tiac-systems.net/doc/3.4/bridle/west_bridle_cmds.html

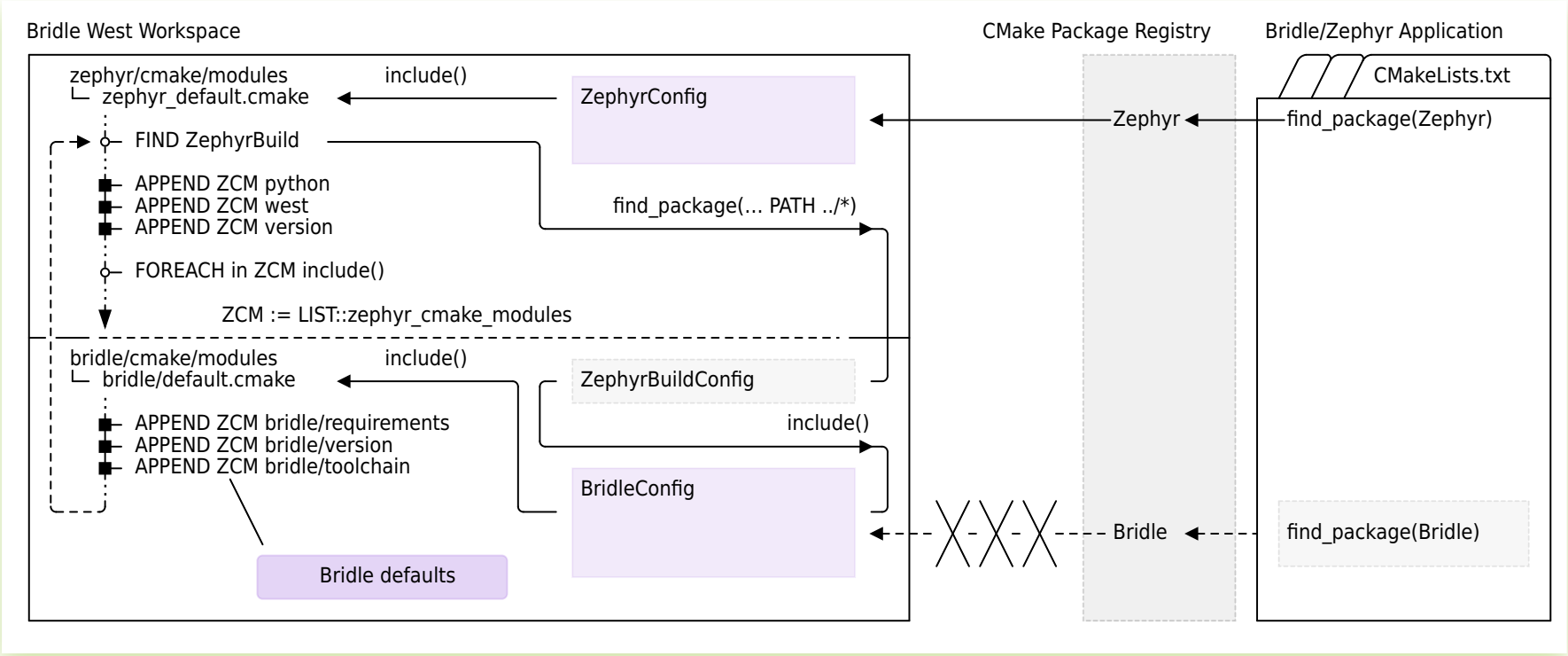


WARNING: Freestanding mode for Bridle not usable. Bridle back to Zephyr default is not yet implemented, see **[BUG] issue 104**.

CMAKE MODULES INJECTED INTO ZEPHYR BY BRIDLE

https://cmake.org/cmake/help/book/mastering-cmake/chapter/Modules.html

Zephyr v3.4 CMake Package: https://bridle.tiac-systems.net/doc/3.4/zephyr/build/zephyr_cmake_package.html



CMAKE MODULES INJECTED INTO ZEPHYR BY BRIDLE

bridle/cmake/modules
└ bridle/default.cmake
  ┆ APPEND ZCM bridle/requirements

Bridle v3.4 `requirements.cmake`: https://github.com/tiacsys/bridle/blob/v3.4.0/cmake/modules/bridle/requirements.cmake

Bridle's toolchain requirements converter (TXT → CMake):

- determine and set the **required Zephyr SDK** version: `set(BRIDLE_TOOLCHAIN_ZEPHYR_SDK_REQUIRED_VERSION …)`
- determine and set the **allowed Zephyr SDK** version: `set(BRIDLE_TOOLCHAIN_ZEPHYR_SDK_VERSION …)`
- determine and set the **required Doxygen** version: `set(BRIDLE_DOXYGEN_REQUIRED_VERSION …)`
- determine and set the **required Sphinx** version: `set(ZEPHYR_SPHINX_REQUIRED_VERSION …)`

`${BRIDLE_BASE}`/scripts/	Zephyr SDK	Doxygen	`${ZEPHYR_BASE}`/scripts/	Sphinx
tools-versions-minimum.txt	zephyr-sdk=0.15.1	doxygen=1.9.1	requirements-doc.txt	sphinx~=5.0,!=5.2.0.post0
tools-versions-linux.txt	zephyr-sdk=0.16.1	doxygen=1.9.2-1+ppa~tsn1~focal		
tools-versions-macos.txt	zephyr-sdk=0.16.1			
tools-versions-win10.txt	zephyr-sdk=0.16.1			

bridle/cmake/modules
└ bridle/default.cmake
  ┼ APPEND ZCM bridle/version

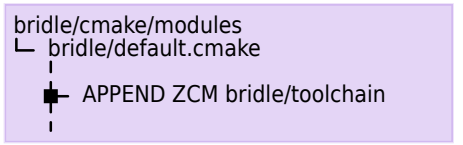
CMAKE MODULES INJECTED INTO ZEPHYR BY BRIDLE

Bridle v3.4 `version.cmake`: https://github.com/tiacsys/bridle/blob/v3.4.0/cmake/modules/bridle/version.cmake

Bridle's version generator for the CMake package and *Bridle Services* API:

- determine this **Bridle package version** from `${BRIDLE_BASE}/VERSION`
- loaded multiple times by `BridleConfigVersion.cmake`
- used by `${BRIDLE_BASE}/cmake/gen_version_h.cmake` to **auto-generate** `generated/version_bridle.h`
- setup CMake variables, e.g.:

`set(PROJECT_VERSION`	`0.2.5.99)`	`set(BRIDLE_VERSION_STRING`	`"0.2.5-extraver")`
`set(BRIDLE_VERSION_MAJOR`	`0)`	`set(BRIDLEVERSION`	`0x20563)`
`set(BRIDLE_VERSION_MINOR`	`2)`	`set(BRIDLE_VERSION_NUMBER`	`0x205)`
`set(BRIDLE_PATCHLEVEL`	`5)`	`set(BRIDLE_VERSION_CODE`	`517)`



CMAKE MODULES INJECTED INTO ZEPHYR BY BRIDLE

Bridle v3.4 `toolchain.cmake`: https://github.com/tiacsys/bridle/blob/v3.4.0/cmake/modules/bridle/toolchain.cmake

Bridle's toolchain version guard for the Zephyr SDK:

- verify that required variables have been defined, see `cmake/modules/bridle/requirements.cmake` above
- verify that selected toolchain matches the required version: `find_package(Zephyr-sdk ${BRIDLE_TOOLCHAIN_ZEPHYR_SDK_REQUIRED_VERSION} …)`
- currently, only the Zephyr SDK is supported

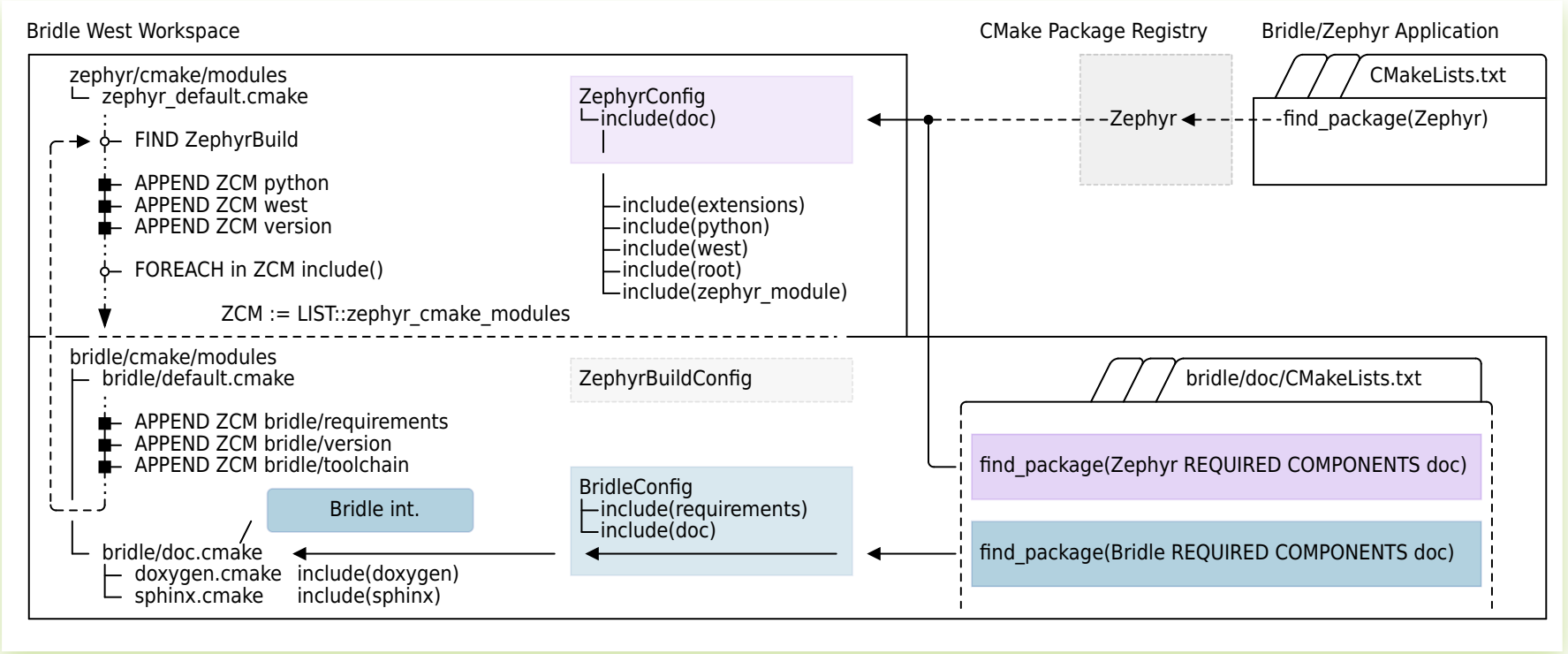
Note:

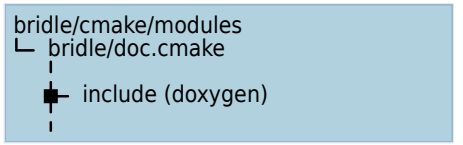
Zephyr has `FindZephyr-sdk.cmake`: `find_package(Zephyr-sdk …)`

CMAKE MODULES USED BY BRIDLE INTERNALLY

https://cmake.org/cmake/help/book/mastering-cmake/chapter/Modules.html

Bridle v3.4 CMake Package: https://bridle.tiac-systems.net/doc/3.4/bridle/west_bridle_cmds.html





CMAKE MODULES USED BY BRIDLE INTERNALLY

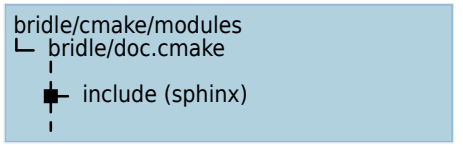
Bridle v3.4 `doxygen.cmake`: https://github.com/tiacsys/bridle/blob/v3.4.0/cmake/modules/doxygen.cmake

Looking for Doxygen:

- verify that required variables have been defined, see `cmake/modules/bridle/requirements.cmake` above
- looking for required Doxygen version: `find_package(Doxygen ${BRIDLE_DOXYGEN_REQUIRED_VERSION} … )`
- looking for required Doxygen components: `find_package( … REQUIRED dot mscgen)`
- setup CMake variables to programs found:
  - `DOXYGEN_EXECUTABLE`
  - `DOXYGEN_DOT_EXECUTABLE`
  - `DOXYGEN_MSCGEN_EXECUTABLE`

Note:

CMake 3.9 has `FindDoxygen.cmake`: `find_package(Doxygen …)`



CMAKE MODULES USED BY BRIDLE INTERNALLY

Bridle v3.4 `sphinx.cmake`: https://github.com/tiacsys/bridle/blob/v3.4.0/cmake/modules/sphinx.cmake

Looking for Sphinx:

- verify that required variables have been defined, see `cmake/modules/bridle/requirements.cmake` above
- looking for required Sphinx version: `find_package(Sphinx ${ZEPHYR_SPHINX_REQUIRED_VERSION} … )`
- looking for required Sphinx components: `find_package( … REQUIRED build)`
- setup CMake variables to programs found:
  - `SPHINX_BUILD_EXECUTABLE`
  - `SPHINXBUILD` (deprecated but still used)

Note:

Bridle has `FindSphinx.cmake`: `find_package(Sphinx …)`

BRIDLE EXTENDS WEST COMMANDS

https://bridle.tiac-systems.net/doc/3.4/zephyr/develop/west/extensions.html

Bridle v3.4 West Commands: https://github.com/tiacsys/bridle/blob/v3.4-branch/scripts/west-commands.yml

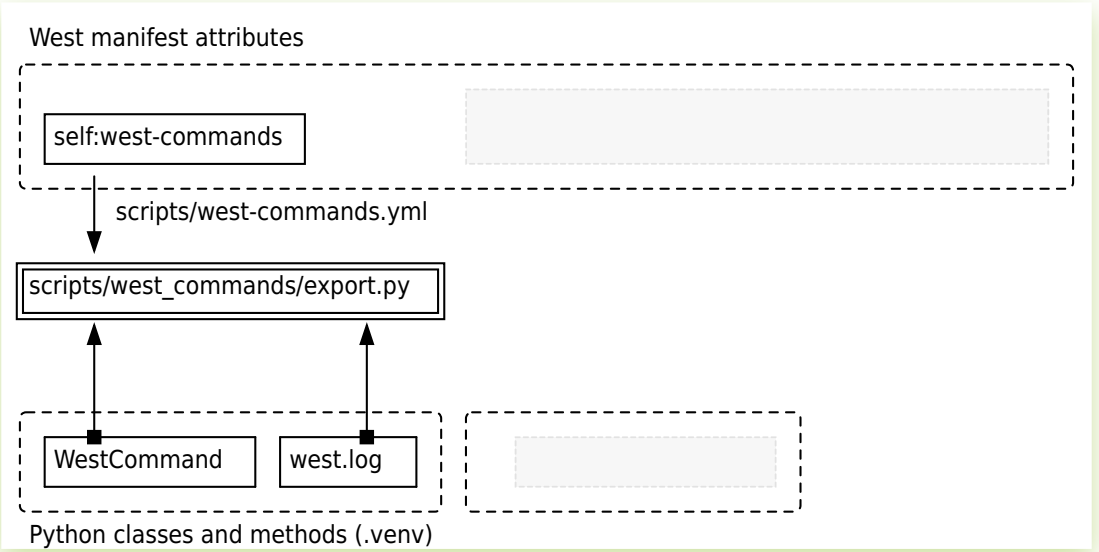
West Manifest (remember)

```
manifest:
  self:
    path: bridle
    west-commands: scripts/west-commands.yml
```

West Commands in Bridle

```
west-commands:
  - file: scripts/west_commands/export.py
    commands:
      - name: bridle-export
        class: BridleExport
        help: export Bridle installation as a CMak
```

West Command Extension in Bridle



Goal: Export Bridle's CMake package in a manner exactly to Zephyr.

WEST COMMAND BRIDLE-EXPORT

https://bridle.tiac-systems.net/doc/3.4/zephyr/develop/west/extensions.html

Bridle v3.4 Export Command: https://github.com/tiacsys/bridle/blob/v3.4-branch/scripts/west_commands/export.py

West Manifest (remember)

```
manifest:
  self:
    path: bridle
    west-commands: scripts/west-commands.yml
```

West Commands in Bridle

```
west-commands:
  - file: scripts/west_commands/export.py
    commands:
      - name: bridle-export
        class: BridleExport
        help: export Bridle installation as a CMak
```

Imp. of `bridle-export` ——→

West Command Implementation for Bridle CMake package export (fragments)

```
from west import log
from west.commands import WestCommand

class BridleExport(WestCommand):

    def __init__(self):

    def do_add_parser(self, parser_adder):

    def do_run(self, args, unknown_args):
        share = os.path.join(Path(__file__).parents[2], 'share')
        self.run_cmake_export(os.path.join(share, 'bridle-package', 'cmake'))

    def run_cmake_export(self, path):
        lines = run_cmake(['-P', os.path.join(path, 'bridle_export.cmake')],
                          capture_output = True)
        msg = [line for line in lines if not line.startswith('-- ')]
        log.inf('\n'.join(msg))
```

Need my own `run_cmake()` implementation! – Why not reuse the original full fancy Zephyr implementation?

WEST COMMAND BRIDLE-EXPORT

https://bridle.tiac-systems.net/doc/3.4/zephyr/develop/west/extensions.html

Bridle v3.4 West Commands: https://github.com/tiacsys/bridle/blob/v3.4-branch/scripts/west-commands.yml

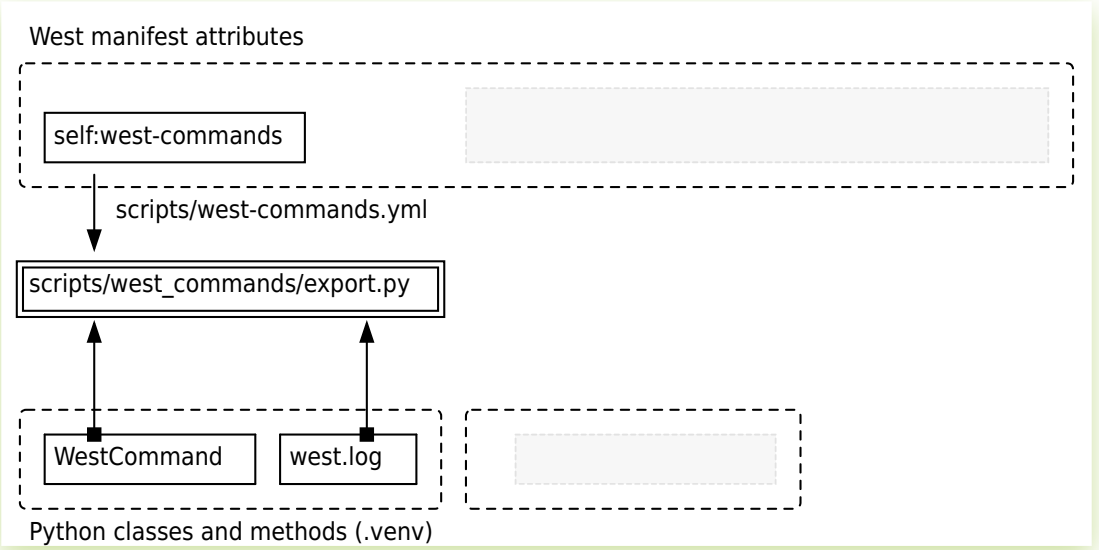
West Manifest (remember)

```
manifest:
  self:
    path: bridle
    west-commands: scripts/west-commands.yml
```

West Commands in Bridle

```
west-commands:
  - file: scripts/west_commands/export.py
    commands:
      - name: bridle-export
        class: BridleExport
        help: export Bridle installation as a CMak
```

West Command Extension in Bridle



Zephyr's `run_cmake()` implementation lives in a specific sub-folder within the West project `zephyr`.

WEST COMMAND BRIDLE-EXPORT

https://bridle.tiac-systems.net/doc/3.4/zephyr/develop/west/extensions.html

Bridle v3.4 West Commands: https://github.com/tiacsys/bridle/blob/v3.4-branch/scripts/west-commands.yml

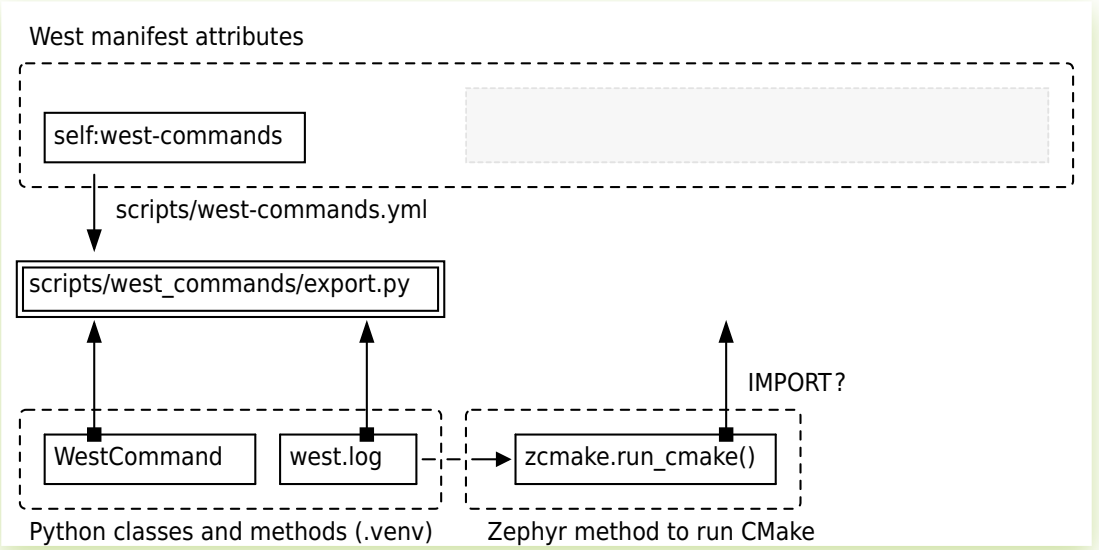
West Manifest (remember)

```
manifest:
  self:
    path: bridle
    west-commands: scripts/west-commands.yml
```

West Commands in Bridle

```
west-commands:
  - file: scripts/west_commands/export.py
    commands:
      - name: bridle-export
        class: BridleExport
        help: export Bridle installation as a CMak
```

West Command Extension in Bridle



Challenge: How can we do “`from zcmake import run_cmake`” without correctly set `sys.path`?

WEST COMMAND BRIDLE-EXPORT

https://bridle.tiac-systems.net/doc/3.4/zephyr/develop/west/extensions.html

Bridle v3.4 West Commands: https://github.com/tiacsys/bridle/blob/v3.4-branch/scripts/west-commands.yml

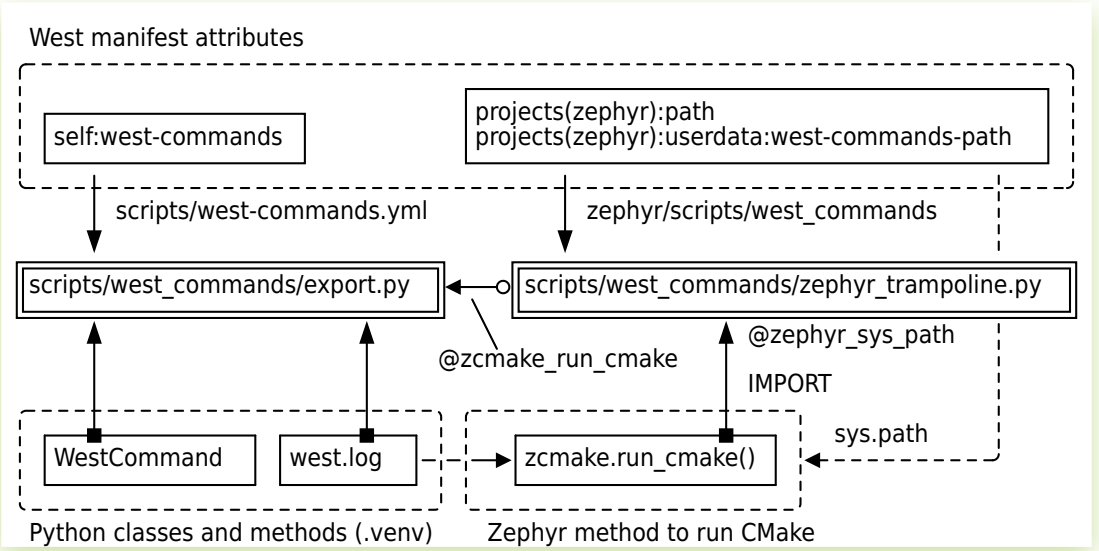
West Manifest (remember)

```
manifest:
  self:
    path: bridle
    west-commands: scripts/west-commands.yml
  projects:
    - name: zephyr
      path: zephyr
      userdata:
        west-commands-path: scripts/west_commands
```

West Commands in Bridle

```
west-commands:
  - file: scripts/west_commands/export.py
    commands:
      - name: bridle-export
        class: BridleExport
        help: export Bridle installation as a CMak
```

West Command Extension in Bridle



Solution: a dedicated *"trampoline"* module pointed to Zephyr by decorators.

WEST COMMAND BRIDLE-EXPORT

Bridle v3.4 Trampoline to Zephyr: https://github.com/tiacsys/bridle/blob/v3.4-branch/scripts/west_commands/zephyr_trampoline.py

West Manifest (remember)

```
manifest:
  self:
    path: bridle
    west-commands: scripts/west-commands.yml
  projects:
    - name: zephyr
      path: zephyr
      userdata:
        west-commands-path: scripts/west_commands
```

West Commands in Bridle

```
west-commands:
  - file: scripts/west_commands/export.py
    commands:
      - name: bridle-export
        class: BridleExport
        help: export Bridle installation as a CMak
```

Extend Python `sys.path` on demand only once, provide as decorator ——→

Bridle's Trampoline to Zephyr (fragments) – 1/2

```
from west import log

def _get_zephyr(topdir, manifest):
    ident = 'zephyr'
    try:
        projects = manifest.get_projects([ident], allow_paths = False)
    if projects:
        return projects[0]

def _get_west_cmd_path(project):
    try:
        abspath = project.abspath
        cmdpath = project.userdata['west-commands-path']
    if abspath and cmdpath:
        return os.path.join(abspath, cmdpath)
```

```
from functools import wraps

def zephyr_sys_path(function):
    @wraps(function)
    def wrapper(self, *args, **kwargs):
        if 'sys_path_with_zephyr' not in function.__globals__:
            try:
                zephyr = _get_zephyr(self.topdir, self.manifest)
            try:
                west_cmd_path = _get_west_cmd_path(zephyr)
            sys.path.append(west_cmd_path)
            function.__globals__['sys_path_with_zephyr'] = sys.path
        retval = function(self, *args, **kwargs)
        return retval
    return wrapper
```

WEST COMMAND BRIDLE-EXPORT

Bridle v3.4 Trampoline to Zephyr: https://github.com/tiacsys/bridle/blob/v3.4-branch/scripts/west_commands/zephyr_trampoline.py

West Manifest (remember)

```
manifest:
  self:
    path: bridle
    west-commands: scripts/west-commands.yml
  projects:
    - name: zephyr
      path: zephyr
      userdata:
        west-commands-path: scripts/west_commands
```

West Commands in Bridle

```
west-commands:
  - file: scripts/west_commands/export.py
    commands:
      - name: bridle-export
        class: BridleExport
        help: export Bridle installation as a CMak
```

Create `run_cmake()` on demand by reusing Zephyr's `run_cmake()` in module `zcmake`, provide as decorator ——→

Bridle's Trampoline to Zephyr (fragments) – 2/2

```
from functools import wraps
from west import log

def _get_zephyr(topdir, manifest):

def _get_west_cmd_path(project):

def zephyr_sys_path(function):
    @wraps(function)
    def wrapper(self, *args, **kwargs):
        retval = function(self, *args, **kwargs)
        return retval
    return wrapper
```

```
from functools import wraps

def zcmake_run_cmake(function):

    @zephyr_sys_path
    @wraps(function)
    def wrapper(self, *args, **kwargs):

        if 'run_cmake' not in function.__globals__:
            from zcmake import run_cmake
            function.__globals__['run_cmake'] = run_cmake

        retval = function(self, *args, **kwargs)
        return retval

    return wrapper
```

WEST COMMAND BRIDLE-EXPORT

Bridle v3.4 Export Command: https://github.com/tiacsys/bridle/blob/v3.4-branch/scripts/west_commands/export.py

West Manifest (remember)

```
manifest:
  self:
    path: bridle
    west-commands: scripts/west-commands.yml
  projects:
    - name: zephyr
      path: zephyr
      userdata:
        west-commands-path: scripts/west_commands
```

West Commands in Bridle

```
west-commands:
  - file: scripts/west_commands/export.py
    commands:
      - name: bridle-export
        class: BridleExport
        help: export Bridle installation as a CMak
```

Imp. of `bridle-export` ——→

West Command Implementation for Bridle CMake package export (fragments)

```
from west import log
from west.commands import WestCommand

from zephyr_trampoline import zcmake_run_cmake

class BridleExport(WestCommand):

    def __init__(self):

    def do_add_parser(self, parser_adder):

    def do_run(self, args, unknown_args):
        share = os.path.join(Path(__file__).parents[2], 'share')
        self.run_cmake_export(os.path.join(share, 'bridle-package', 'cmake'))

    @zcmake_run_cmake
    def run_cmake_export(self, path):
        lines = run_cmake(['-P', os.path.join(path, 'bridle_export.cmake')],
                          capture_output = True)
        msg = [line for line in lines if not line.startswith('-- ')]
        log.inf('\n'.join(msg))
```

Finally: This West command in Bridle want to rely on core functionality from Zephyr, thus it uses the `@zcmake_run_cmake` decorator provided by Bridle's trampoline to Zephyr.

ZEPHYR EXAMPLE APPLICATION VS. BRIDLE

https://github.com/zephyrproject-rtos/example-application ↔ https://github.com/tiacsys/bridle

Content	*Example*	*Bridle*	*Content*	*Example*	*Bridle*	*Content*	*Example*	*Bridle*
`.github`	build only	**HIL/Tests, QA & DOC**	`drivers`	example_sensor	**multiple**	`scripts` & `.yamllint`	West	**West** & **linting**
`applications`	single	prepared	`dts`	binding only	bindings & **stm32f777xx**	`share`	—	**CMake package**
`arch`	—	prepared	`include`	custom_lib	**versioning**	`soc`	—	**stm32f777xx**
`boards`	custom_plank	**multiple**	`lib`	custom_lib	**versioning**	`subsys`	—	**Shell commands**
`cmake`	—	**CMake extensions**	`modules`	—	prepared	`tests`	custom_lib	**multiple**
`doc`	—	**doc-sets**	`samples`	—	multiple	`VERSION` & `version.h.in`	—	**versioning**

GETTING STARTED IN PRACTICE

https://bridle.tiac-systems.net/doc/3.4/bridle/getting_started.html

Getting started

To quickly get started with Bridle, use the Getting Started Assistant to set up your development environment. Alternatively, check the Installing Bridle manually section for instructions on setting up your environment manually.

We recommend using [t.b.d.] for building your applications. See Building and programming a sample application for the download links and instructions. In case you run into problems, see Troubleshooting ... t.b.d. (TBD) IDE .

Once you are set up, check out the Samples. If this is the first time you work with embedded devices, it is probably a good idea to program an unchanged sample to your board first and test if it works as expected. After that, pick a sample that is related to the application you want to create and start developing!

- Supported operating systems
- Requirements
  - Supported operating systems
  - Required tools

INSTALLING BRIDLE MANUALLY

https://bridle.tiac-systems.net/doc/3.4/bridle/gs_installing.html#cloning-the-repositories

1. Installing the required tools – refer also the Zephyr documentation: Install Linux Host Dependencies
2. Installing the toolchain – refer also the Zephyr documentation: Zephyr SDK
3. Getting the Bridle code
    1. Python Virtual Environment
    2. West workspace for Bridle
    3. Export **CMake packages** (Zephyr and Bridle)

CMake messages from Bridle

```
-- west build: generating a build system
Loading Zephyr default modules (Zephyr base).
Loading Bridle default modules (Bridle workspace).
-- Bridle version: 3.4.0 (…/workspace/bridle)
-- Bridle is trying to locate Zephyr SDK 0.16.1.
-- Using Zephyr SDK 0.16.1 for building Bridle. (/opt/zephyr-sdk-0.16.1)
… … …
-- Zephyr version: 3.4.0 (…/workspace/zephyr), build: v3.4.0-2-gd6c9e937a33e
… … …
-- Bridle version: 3.4.0 (…/workspace/bridle), build: v3.4.0-1-g4ae9f2aa7ba0
… … …
```

West Workspace Setup from Bridle

```
west init -m https://github.com/tiacsys/bridle --mr v3.4-branch
west update
```

manifest https://github.com/tiacsys/bridle/blob/v3.4-branch/west.yml

Python Virtual Environment Setup

```
pip3 install --upgrade --requirement zephyr\scripts\requirements.txt
pip3 install --upgrade --requirement bridle\scripts\requirements.txt
```

requirements https://github.com/tiacsys/bridle/blob/v3.4-branch/scripts/requirements.txt

CMake Package Export

```
west zephyr-export
west bridle-export
```

“Hello World!” sample on QEMU

```
west build -p always -b qemu_cortex_m3 zephyr/samples/hello_world
west build -t run
```

"HELLO WORLD!" SAMPLE ON QEMU

```
-- west build: building application
[1/140] Preparing syscall dependency handling

[2/140] Generating include/generated/version.h
-- Zephyr version: 3.4.0 (/var/tmp/ascrec/workspace/zephyr), build: v3.4.0-2-gd6c9e937a33e
[3/140] Generating ../../zephyr/include/generated/version_bridle.h
-- Bridle version: 3.4.0 (/var/tmp/ascrec/workspace/bridle), build: bridle-v3.4.0-1-g4ae9f2aa7ba0
[130/140] Linking C executable zephyr/zephyr_pre0.elf

[134/140] Linking C executable zephyr/zephyr_pre1.elf

[140/140] Linking C executable zephyr/zephyr.elf
Memory region         Used Size  Region Size  %age Used
           FLASH:        8746 B       256 KB      3.34%
             RAM:        3960 B        64 KB      6.04%
       IDT_LIST:          0 GB         2 KB      0.00%
(.venv) user@host:workspace$ west build -t run &
-- west build: running target run
[0/1] To exit from QEMU enter: 'CTRL+a, x'[QEMU] CPU: cortex-m3
qemu-system-arm: warning: nic stellaris_enet.0 has no peer
Timer with period zero, disabling
*** Booting Zephyr OS build v3.4.0-2-gd6c9e937a33e ***
Hello World! qemu_cortex_m3

-----------------------------------------------------------------------------------------------------------
1️⃣ 2️⃣
```

"Hello World!" sample with Bridle

ZEPHYR DOWNSTREAM ATTRIBUTES

Bridle is a Zephyr downstream project, because it:

- provides THE **West Manifest** for source code organization
- is itself a **Zephyr Module** to extend the Zephyr framework
- provides at least entries for **CMake** and **Kconfig**
- extends **West** if required

only this is

The Zephyr Example Application.

→ *https://github.com/zephyrproject-rtos/example-application*

but

The Bridle Downstream Project is more than that. → *https://github.com/tiacsys/bridle*

FURTHER HIGHLIGHTS AND OUTLOOK

Bridle

/ Supported Boards

Supported Boards

The Bridle project developers are continually adding board-specific support as documented below.

To add support documentation for a new board, please use the template available under doc/templates/board.tmpl

- x86 Boards
- ARM Boards
  - Arduino/Genuino Zero
  - Seeed Studio XIAO SAMD21 (Seeeduino XIAO)
  - Seeeduino Lotus Cortex-M0+
  - TiaC Magpie STM32F777NIHx
- ARC Boards
- NIOS II Boards

Bridle

/ Supported Boards / ARM Boards
/ TiaC Magpie STM32F777NIHx

TiaC Magpie STM32F777NIHx

Bridle

/ Supported Boards / Shields
/ Seeed Studio Grove Interconnect Shields

Seeed Studio Grove Interconnect

Bridle

/ Applications and samples

Applications and samples

The Zephyr Project contains a variety of application samples and demos. Documentation for those is available in the Samples and Demos section.

Bridle provides additional examples that specifically target TiaC Systems devices and show how to implement typical use cases with our libraries and drivers. Samples showcase a single feature or library, while applications include a variety of libraries to implement a specific use case.

- Applications
- Samples
  - Button
  - Hello Shell
- Tests

Bridle

/ API Reference

API Reference

- Zephyr API Overview and API Terminology:
  - reschedule
  - sleep
  - no-wait
  - isr-ok
  - pre-kernel-ok
  - async
  - supervisor

- API Overview
- Bridle Services

Previous Next

BRIDLE OUTLOOK

1. short term
    - *fix*: **[BUG] issue 104** – CMake freestanding mode for Bridle not usable.
    - *revice*: build system for documentation – avoid multiple Doxygen runs.
    - *feasibility*: **Zephyr issue 58882** – trace existing test cases back to design specification, e.g. with **Sphinx-Needs**.
    - *improve*: documentation – render PDF per documentation set and version.
    - *improve*: Bridle's *Development Model* – release policy and workflow.
2. mid term
    - *add*: more Seeed Studio **Grove Shields**, e.g. Buzzer, OLED, Sensors
    - *add*: more popular connector systems similar to Grove, e.g. **PMOD**
    - *improve (try out)*: **PWM (ADC/DAC) channel** mapping over **DeviceTree Nexus Nodes**
    - *finalize*: next generation on premise **HIL Infrastructure**
3. long term
    - *feasibility*: Motor / Actuator API (stepper, BLDC, linear, VFD-AC ? ...)
    - *feasibility*: further conceptual domains beside or on top of *Boards* and *Shields*, e.g. *Modules / Units* and *Rigs*

THANK YOU FOR YOUR ATTENTION!

Your presenters were:

Tobias Kästner
tobias.kaestner@ul.com

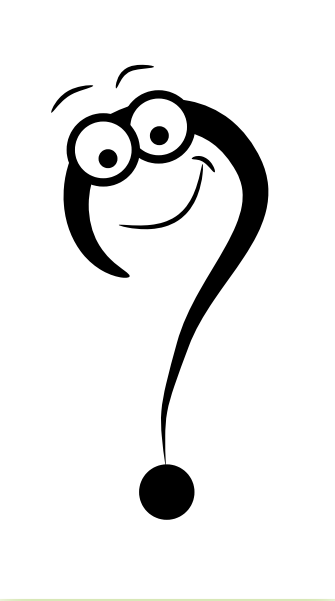
Stephan Linz
stephan.linz@navimatix.de

Organization Navimatix GmbH
Landing https://navimatix.de
Contact info@navimatix.de
Address Jena, Germany

online presentation https://bridle.tiac-systems.net/zds23-bridlekite
printable presentation https://bridle.tiac-systems.net/zds23-bridlekite/slides.pdf

SCAN ME

QUESTIONS



- Bridle
  - Git: https://github.com/tiacsys
  - Git: https://github.com/tiacsys/bridle
  - Doc: https://bridle.tiac-systems.net/
  - Web: https://www.tiac-systems.net/
- Zephyr
  - Git: https://github.com/zephyrproject-rtos/zephyr
  - Doc: https://docs.zephyrproject.org/
  - Web: https://www.zephyrproject.org/
- Zephyr (Downstream) Example Application
  - Git: https://github.com/zephyrproject-rtos/example-application

SCAN ME