FELIX Developer Manual

ATLAS FELIX Group

Version 0.66-3-g994f954

Table of Contents

0 :!table: 0

1. Welcome to the FELIX Developer Manual

1.1 Overview

This document is intended as both an introductory guide and ongoing reference for all developers contributing to the FELIX project for both ATLAS Phase-I and Phase-II upgrades.

1.1.1 Code of Conduct

The FELIX project operates under the CERN code of conduct. Please honour this when working with others.

1.1.2 Important Contacts

Project Leaders: Carlo A. Gottardo

Firmware Coordinator: Frans Schreuder

Software Coordinator: Mark Donszelmann

User Support Coordinator: Sasha Paramonov

1.1.3 Mailing Lists

Developer Mailing List: atlas-tdaq-felix-developers@cern.ch

User Mailing List: atlas-tdaq-felix-users@cern.ch

1.1.4 Developer Quick Links

The FELIX Project Website (with dedicated 'Information for Developers' section):

https://atlas-project-felix.web.cern.ch/atlas-project-felix

FELIX GitLab Group:

https://gitlab.cern.ch/atlas-tdaq-felix

FELIX CERNbox Site:

https://cernbox.cern.ch/index.php/apps/files/?dir=/__myprojects/felix

FELIX project central JIRA instance:

https://its.cern.ch/jira/projects/FLX

FELIX development dashboard (summary of overall status):

https://its.cern.ch/jira/secure/Dashboard.jspa?selectPageId=19315

FELIX Meetings in Indico:

https://indico.cern.ch/category/5501/

Use of SharePoint has been discontinued. Please report any broken links of obsolete material to help improve the overall quality of our documentation.

1.1.5 Releases and User Support Links

The FELIX release distribution site:

https://atlas-project-felix.web.cern.ch/atlas-project-felix/user/dist/

User support requests from users to the FELIX team should be made via the dedicated JIRA project:

https://its.cern.ch/jira/projects/FLXUSERS

FELIX User dashboard (summary of overall status):

https://its.cern.ch/jira/secure/Dashboard.jspa?selectPageId=17501

:!table: 1 :numbering:

2. Checklist for new Developers

2.1 Overview

Welcome to the team!

All new developers should follow the following checklist when joining the project:

- Register for the FELIX developer E-group (needed to view this page on the web!)
- Add the weekly developer meeting to your calendar
  - Mondays at 3pm CERN time
  - All FELIX meetings can be found in indico
- If interested in firmware - add the bi-weekly firmware meeting to your calendar
  - Tuesdays at 4pm CERN time
- Review this guide for an introduction to the project and our working model
- Join Mattermost using this link! (Note: this replaces the old Slack channels).
- Confirm you have read access to the project CERNbox area: `/eos/project/f/felix` or visible via your browser
  - If you need write access, join the CERNbox access control e-group: `atlas-tdaq-felix-cernbox-writers`

We no longer make use of Sharepoint, with all functionality moved to CERNbox. Therefore please don't request access to Sharepoint or direct any users to it.

:!table: 2 :numbering:

3. Development Methodology

3.1 Overview of Components

The work of the FELIX project is designed around the production of firmware and software for the operation of the FELIX card, a PCIe I/O card hosted in a commodity server PC. For more information on the project in general, please consult the specification document. The primary output of the development effort is the **FELIX release**, consisting of a series of firmware images covering different operational modes, along with a self-contained software suite. Dedicated device drivers, facilitating different FELIX features, are also provided. For a list of available releases please consult the user manual.

Full details of the firmware and software will be provided in dedicated sections of this manual. Broadly-speaking, the firmware consists of modules implementing specific I/O card features (such as the link wrapper, central router, or PCIe interface) composed into a specific image using a top-level design. The software consists of the device drivers, plus both low level tools for testing and hardware communication, as well as high level applications for high performance dataflow and command routing. The interface between the firmware and software domains, whereby data and instructions are exchange between host server and I/O card, is implemented for commands and monitoring via programmed I/O using a firmware register map and with DMA transfers for high throughput scenarios.

All code pertaining to FELIX software and firmware, irrespective of the development platform used, is stored and under version control using GitLab. All development issues are tracked and discussed using JIRA. In what remains of this section, a description will be presented of the overall development methodology used in FELIX. Specifically, how we use JIRA and GitLab, what our rules are for creating and integrating changes and how we validate and distribute releases.

3.2 Working with JIRA

Before starting work, you may wish to familiarise yourself with JIRA by consulting the user guide.

All FELIX development items are tracked using the following JIRA instance:

https://its.cern.ch/jira/projects/FLX

For a more focused summary, including upcoming releases and priorities, please consult the project dashboard:

https://its.cern.ch/jira/secure/Dashboard.jspa?selectPageId=19315

A separate JIRA exists as an external interface for user requests:

https://its.cern.ch/jira/projects/FLXUSERS

For a more focused summary, a dedicated dashboard is also available:

https://its.cern.ch/jira/secure/Dashboard.jspa?selectPageId=17501

When contributing to FELIX, please observe the following rules:

1. Create new JIRA issue in FLX describing the change when adding a new feature or fixing a bug
    a. Where an issue was first reported in FLXUSERS, create a corresponding FLX issue to track development
2. When creating an issue remember the following:
    a. Set the affects/fix versions fields corresponding to the relevant release
    b. Set the relevant components affected by the change (the coordinators may later choose to also associate the issue with an Epic)
3. When satisfied that a change is complete, set the issue to **Resolved** but do not close it. The issue will be closed by the relevant coordinator when the change has been sufficiently validated.

3.2.1 JIRA Epics

In order to better prioritise work, FELIX makes use of JIRA Epics in order to better describe overall project priorities beyond the standard release cycle. The list of topics associated with each Epic can be viewed on the FLX dashboard. A JIRA Epic collects together multiple ordinary JIRA issues for tracking.

When prioritising work, please consider Project priority (from the Epic) more important that individual JIRA priority. A list of Epics is given below.

FLX - Urgent

https://its.cern.ch/jira/browse/FLX-812

This is the highest priority JIRA Epic. Issues associated with this Epic should be handled before any other project activities. Within the Epic, issues should be handled according to normal JIRA priority.

FLX - Short Term Development

https://its.cern.ch/jira/browse/FLX-860

This is the second highest priority JIRA Epic. Issues associated with the Epic must be completed in the near future to match the current high priority (but not urgent) goals of the project. Work on these issues should therefore be prioritised ahead of any other issues, with the exception of those marked with FLX - Urgent. Within the Epic, issues should be handled according to normal JIRA priority.

FLX - Medium Term Development

https://its.cern.ch/jira/browse/FLX-815

This is the third highest priority JIRA Epic. Issues associated with this are to be handled after any Urgent or PRR issues are complete. Within the Epic, issues should be handled according to normal JIRA priority.

3.3 Working with GitLab

Before starting work, you may wish to familiarise yourself with GitLab by consulting the user guide.

With the exception of the device driver, all FELIX development contributing to the release is under version control within the following GitLab group:

https://gitlab.cern.ch/atlas-tdaq-felix.

The device driver resides within the TDAQ software GitLab group project called **ROSRCDdrivers**:

https://gitlab.cern.ch/atlas-tdaq-software/ROSRCDdrivers.

As separate group exists for non-release products, such as the project website and this manual:

https://gitlab.cern.ch/atlas-tdaq-felix-dev.

Following the standard git-based development paradigm (*branch early, branch often!*), FELIX requires that all development take place in dedicated branches, taken from an appropriate reference. The primary branch, from which all new firmware and software releases are build, is known as **master**. All commits and merges to this branch are locked unless authorised by either the firmware coordinator (Frans Schreuder) or software coordinator (Mark Donszelmann). Additional protected master-like threads (e.g. for Phase-II firmware) may also be crated as needed, for which the same constraints apply. In order to be accepted for master changes must pass a series of tests (as specified for the domain) to ensure they are working as expected and introduce no regressions. Some specific examples for how to work on particular types of change are given below.

3.3.1 Feature development

All new feature development should, where not otherwise advised, take place against the latest version of **master**. To work on a change, please first create a branch from master with an appropriately informative name (specific details on how will be given in the firmware and software sections). As well as the feature itself, sufficient testing (be it firmware simulation or e.g. software unit tests) should be implemented in the branch to demonstrate functionality and verified to be functioning locally (how to create a merge request). The specific testing requirements for firmware and software will also be given in the dedicated chapters of this manual.

To be accepted, all merge requests must contain a reference to the original JIRA issue pertaining to the change.

Once you are confident that your change has been robustly tested, you should push your branch to GitLab and request a merge into master. The relevant coordinator will then assess your change and decide whether or not to require further tests or provide code review comments. Once this process is complete the change will be merged into master for integration into the nightly testing framework. A feature is not considered completely signed off until demonstrated to be fully working in the nightly.

3.3.2 Bugfixes

When implementing a bugfix on a prior release the overall methodology is the same as for features. However, the branch should be created from the dedicated branch associated with the release component to be fixed. Fixes will then be merged back into this branch and included in the next point release to be issued. Fixes will also then be merged back into **master** where needed. Please indicate in any release bugfix merge request whether you consider that the fix should also go to **master**, or whether it has since been superseded by other changes there.

In the situation that a bugfix needs to be made directly to **master**, this should be handled in an identical way to the development of a new feature above.

As for new features, all bugfix merge requests must contain a reference to the original JIRA issue pertaining to the bug.

3.4 Releases and Validation

Add more detailed description of nightly tests and CI!

3.4.1 Common Testing Facilities

The FELIX project maintains a fully featured validation platform in CERN building 4 (TDAQ testbed), with access to all necessary hardware platforms and software tools. Should your local test setup not be sufficient to fully demonstrate a change (e.g. if you don't have a FELIX I/O card) you can request time on the central testbed to validate your change via the Slack **#testbed** channel. For more details consult the testbed section of this guide.

3.4.2 Release Distributions

Major FELIX releases are built at regular intervals as part of the overall development roadmap of the project. Point releases may be issues at any time should bugs be found. A summary of all current release development can be found in JIRA.

Releases are made available to users via a dedicated page:

https://atlas-project-felix.web.cern.ch/atlas-project-felix/user/dist/

Firmware is released as a self-contained tarball for each top-level build. The primary software release is available both as a tarball and an rpm. All device drivers are available via rpm.

An area for internal downloads for trial releases (before being made public) is available here:

https://atlas-project-felix.web.cern.ch/atlas-project-felix/dev/staging/www/dev/dist/

3.4.3 Installations for Operations

Alongside the standard distribution mechanisms above, dedicated installations of FELIX software and drivers are maintained at P1 and in the B4 testbed. For more information on these installations, please contact Mark Donszelmann for the main release and Markus Joos for the driver.

:!table: 3 :numbering:

4. Firmware Development

4.1 Firmware repository

The FELIX firmware GIT repository is the central place for firmware development.

- The development for phase I has mostly stabilized, and very little new features are added; mostly bugfixes.
  - the master branch is protected.
  - A new feature or bugfix must be related to a JIRA ticket, the name of the branch should start with the JIRA number. If for example the JIRA ticket is FLX-1354, the feature branch name could be FLX-1354_NSW_640Mb_ELink
  - To merge changes into master, a merge request must be created in Gitlab. The merge request must be assigned to the librarian (Frans Schreuder)
- Development for phase II is very active
  - the phase2/master branch is protected.
  - A new feature or bugfix must be related to a JIRA ticket, the name of the branch should start with the phase2/JIRA number. If for example the JIRA ticket is FLX-1354, the feature branch name could be phase2/FLX-1354_NSW_640Mb_ELink
  - To merge changes into phase2/master, a merge request must be created in Gitlab. The merge request must be assigned to the librarian (Frans Schreuder), don't forget to set phase2/master as the target branch.

4.1.1 Merging and Validation of Changes

For a merge request to be completed, a few conditions must be met:

- The Gitlab CI pipeline must be completed without issues. This includes simulation and build of the different flavours
- A test on hardware is required (This is now a hard condition for phase1 branches, at a later stage it will also be required for phase2). This test must include a standard set of automated tests, but a newly added feature or set will be tested separately, depending on the feature.
- The changes will be reviewed by the librarian (Frans Schreuder) visually, and comments / questions can be made.

4.2 Sharing and distribution of bitfiles

Bitfiles that are built by Gitlab CI on a merge request or manual trigger are automatically copied to /eos/project/f/felix/www/dev/dist/firmware/Bitfiles_nightly and are accessible here: https://atlas-project-felix.web.cern.ch/atlas-project-felix/dev/dist/firmware/Bitfiles_nightly/

Developers who build a bitfile for a certain purpose can share this among developers or to users through the cerbox user interface:

1. Request access to the e-group atlas-tdaq-felix-cernbox-writers
2. To share with other developers, upload a bitfile to cernbox/Bitfiles_development
    - The file will be accessible on https://atlas-project-felix.web.cern.ch/atlas-project-felix/dev/dist/firmware/Bitfiles_development/
3. To share with users, upload a bitfile to cernbox/Bitfiles_development_user
    - The file will be accessible on https://atlas-project-felix.web.cern.ch/atlas-project-felix/user/dist/firmware/Bitfiles_development/

4.3 Overview of Firmware Modules

The different HDL files / entities are best described in the FELIX Phase2 firmware specification

4.4 Firmware Top Level

For phase1 based branches, the toplevel VHDL file is dependent on the firmware flavour (GBT / FULL) and the hardware (VC709, BNL712). Note that the BNL712 (FLX712) is version 2.0 of the BNL711, so some names of files and constraints contain bnl711 rather than bnl712.

Flavour	Hardware	Toplevel
GBT	VC709	sources/FelixTop/felix_top.vhd
FULL	VC709	sources/FelixTop/felix_fullmode_top.vhd
GBT	BNL712	sources/FelixTop/felix_top_bnl711.vhd
FULL	BNL712	sources/FelixTop/felix_fullmode_top_bnl711.vhd

For phase2 based branches, all flavours are contained in a single toplevel VHDL file:

Flavour	Hardware	Toplevel
GBT, FULL, PIXEL, STRIP, LPGBT, …	VC709, BNL712, VCU128, VMK180	sources/FelixTop/felix_top.vhd

4.5 Register Map & JINJA

The PCIe DMA core (Wupper), see also the FELIX Phase2 firmware specification consists of a DMA engine, but also contains a register map to control, configure and monitor anything in the FELIX firmware. The registers are stored in a YAML file, for phase I branches the version is rm-4.10, for phase II branches we use rm-5.0.

The YAML file has 3 sorts of registers/bitfields:

- R: A read only register, used to monitor status of the firmware
- W: A read/write register, used to control settings.
- T: Trigger: A self clearing bitfield within a register, will pulse shortly (~5 clock cycles) if any bitfield within a register is written

4.5.1 Control and Monitor Records

Read only registers are placed in register_map_monitor (internal to Wupper), but it is more practical to drive the status signals from within the different blocks in the firmware, rather than a central point. Therefore the record register_map_monitor is divided into different subrecords called "Monitor Sections". The monitor sections are difined in the top section of the YAML file. All R bitfields must be contained in a monitor section.

Read/Write and Trigger registers are placed in register_map_control records. This record is not divided into sub records as it is easy to fan out the complete register map to where it is required without driver conflicts.

4.5.2 Syntax

The syntax of the yaml file is described in the WupperCodeGen manual, but it is easiest to browse existing registers and copy / paste snippets in order to add a register.

Registers can contain complex structures:

- Single (unnamed) bitfields, resulting in a single register with a std_logic_vector
- Multiple named bitfields, resulting in a record containing multiple std_logic_vector bitfields
- A 1D array of registers, resulting in an array of records.
- A 2D array of registers, resulting in an array of an array of records. For Arrays, references (ref:) in YAML must be used, and the referenced tree must have the keyword "number:" for the size of the array.

4.5.3 Generating the files from yaml

To generate the VHDL files (and Latex / HTML documents) from the YAML files, WupperCodeGen is required. WupperCodeGen is depending on: the following system packages:

- python-jinja2
- python-argparse
- python-yaml
- python-markupsafe

It is expected that the different repositories are cloned in the following format:

```
felix
 ├──── firmware
 └──── software
     ├──── wuppercodegen
     └──── ...
```

This tree can be obtained by cloning the felix repository with the following commands:

```
git clone ssh://git@gitlab.cern.ch:7999/atlas-tdaq-felix/felix
cd felix/
./clone_all.sh ssh
cd software/
./clone_all.sh ssh
```

To build the VHDL files and Latex document, the following commands can be used:

```
cd firmware/sources/templates
./build.sh
./build-doc.sh
```

4.6 Getting Started with Vivado

The FELIX Phase I firmware is built with Vivado 2020.1. For CERN Linux computers, an installation on AFS is available, other institutes / systems must manage their own installations. The FELIX Phase I firmware is build using different vivado versions depending on the hardware:

- The FLX709 and FLX712 builds use Vivado 2021.2
- The FLX182 builds use Vivado 2022.2
- The FLX155 builds use Vivado 2023.2

When Versal Premium is well supported in a future Vivado version in 2024, all the hardware platforms for phase II will be upgraded to that Vivado version.

```
export XILINXD_LICENSE_FILE="2112@licenxilinx"
#For Phase I
source /afs/cern.ch/work/f/fschreud/public/Xilinx/Vivado/2020.1/settings64.sh
#For Phase II FLX709 or FLX712
source /eos/project/f/felix/xilinx/Vivado/2021.2/settings64.sh
#For Phase II FLX182
source /eos/project/f/felix/xilinx/Vivado/2022.2/settings64.sh
#For Phase II FLX155
source /eos/project/f/felix/xilinx/Vivado/2024.2/settings64.sh

vivado
```

FELIX is a large project, and therefore needs a solid build server. 64GB of memory or more is recommended to build. Expect build times up to 10 hours for full featured firmware builds.

4.6.1 Introduction to FELIX Firmware build scripts

The FELIX build system is based on .tcl scripts, which can be used for Vivado, Questasim (Modelsim) and Sigasi.

4.6.1.1 Creating the Vivado project and building a bitfile

Launch Vivado, and open the TCL console.

To create and build the Vivado project (Phase I branches), type:

```
#For GBT mode go to the FELIX_top directory, for FULL mode it is FELIX_fullmode_top
cd felix/firmware/scripts/FELIX_top
#To create the project, replace FLX712 with FLX709 when building for the VC709 card
source ./FLX712_GBT_import_vivado.tcl
#To run synthesis, implementation and create a bitstream in felix/firmware/output
source ./do_implementation_BNL712.tcl
```

To create and build the Vivado project (Phase II branches), type:

```
cd felix/firmware/scripts/FELIX_top
#To create the project, replace FLX712 with FLX709 when building for the VC709 card
source ./FLX712_FELIX_import_vivado.tcl
#To run synthesis, implementation and create a bitstream in felix/firmware/output
source ./do_implementation_BNL712_GBT.tcl #GBT mode
source ./do_implementation_BNL712_FULL.tcl #FULL mode
#There are more flavours in that directory that can be built with similar scripts
```

Generated files (.bit, .mcs, .txt, .ltx, .xlsx) will be generated and bundled as .tar.gz in the output/ directory.

4.6.1.2 Debugging with ILA Chipscope probes

An ILA or VIO IP core can simply be added at source level as an IP core, but it can also be added to a syntesized project in the following way:

- In the do_implementation_XXX.tcl script add the following line:

```
set STOP_TO_ADD_ILA 0
```

- Run the script wand wait for synthesis to be completed. In Vivado open the schematic or netlist view to mark nets as "DEBUG". Then click "Set up debug" under Synthesized Design, to create debug probes. Don’t forget to save the synthesized design. Debug probes will be added in constraints/felix_probes.xdc
- in the .tcl console type

```
source ../helper/do_implementation_finish.tcl
```

- The .bit file and debug_probes.ltx file will be generated in output/
- To start with a clean (no debug) project, empty constraints/felix_probes.xdc

4.6.1.3 Filesets

All the files that are used for building or simulating are defined in filesets. The actual project is created by scripts in the scripts/helper directory. A project specific script for Vivado, Questasim or Sigasi contains the following lines:

```
#Initialize empty TCL variables
source ../helper/clear_filesets.tcl

set PROJECT_NAME FLX709_FELIX
set BOARD_TYPE 709
set TOPLEVEL felix_top

#Import blocks for different filesets
source ../filesets/wupper_fileset.tcl
source ../filesets/lpgbt_core_fileset.tcl

#Actually execute all the filesets for Vivado specific
source ../helper/vivado_import_generic.tcl
#For questasim and Sigasi, other tool specific scripts are available in scripts/helper
```

A fileset is a .tcl script that defines TCL arrays using the following syntax:

```
set VHDL_FILES [concat $VHDL_FILES \
  templates/pcie_package.vhd \
  templates/dma_control.vhd]
```

In the snippet above, the variable VHDL files is used. Instead of VHDL_FILES, the following variables can be used:

Variable	Description	Relative to
XCI_FILES	Xilinx IP core file	sources/ip_cores/<architecture>
VHDL_FILES	Synthesizable VHDL	sources/
VERILOG_FILES	Synthesizable Verilog	sources/
SIM_FILES	VHDL files for simulation	simulation/
EXCLUDE_SIM_FILES	Sythesizable VHDL files, not to be included in simulation	sources
WCFG_FILES	(Deprecated) Vivado waveforms	simulation/
BD_FILES	Vivado Block design	sources/ip_cores/<architecture>
XCI_FILES_V7	IP core files for Virtex7	sources/ip_cores/<architecture>
VHDL_FILES_V7	Synthesizable VHDL for Virtex7	sources/
SIM_FILES_V7	Simulation VHDL for Virtex7	simulation/
BD_FILES_V7	Vivado block design files for Vertex7	sources/ip_cores/<architecture>

XCI_FILES_KU	IP core files for Kintex Ultrascale	sources/ip_cores/<architecture>
VHDL_FILES_KU	Synthesizable VHDL for Kintex Ultrascale	sources/
SIM_FILES_KU	Simulation VHDL for Kintex Ultrascale	simulation
BD_FILES_KU	Vivado block design files for Kintex Ultrascale	sources/ip_cores/<architecture>
XCI_FILES_VU9P	IP Core files for Virtex Ultrascale+ VU9P	sources/ip_cores/<architecture>
VHDL_FILES_VU9P	Synthesizable VHDL for Virtex Ultrascale+ VU9P	sources/
SIM_FILES_VU9P	Simulation VHDL for for Virtex Ultrascale+ VU9P	simulation/
BD_FILES_VU9P	Vivado block design files for Virtex Ultrascale+ VU9P	sources/ip_cores/<architecture>
XCI_FILES_VU37P	IP Core files for Virtex Ultrascale+ VU37P	sources/ip_cores/<architecture>
VHDL_FILES_VU37P	Synthesizable VHDL for Virtex Ultrascale+ VU37P	sources/
SIM_FILES_VU37P	Simulation VHDL for for Virtex Ultrascale+ VU37P	simulation/
BD_FILES_VU37P	Vivado block design files for Virtex Ultrascale+ VU37P	sources/ip_cores/<architecture>
XCI_FILES_VERSAL	IP Core files for Versal Prime	sources/ip_cores/<architecture>
VHDL_FILES_VERSAL	Synthesizable VHDL for Versal Prime	sources/
SIM_FILES_VERSAL	Simulation VHDL for for Versal Prime	simulation/
BD_FILES_VERSAL	Vivado block design files for Versal Prime	sources/ip_cores/<architecture>
XDC_FILES_VC709	Constraints for VC709	constraints/
XDC_FILES_HTG710	Constraints for HTG710	constraints/
XDC_FILES_BNL711	Constraints for BNL711	constraints/
XDC_FILES_BNL712	Constraints for BNL712	constraints/
XDC_FILES_VCU128	Constraints for VCU128	constraints/
XDC_FILES_XUPP3R_VU9P	Constraints for XUPP3R_VU9P	constraints/
XDC_FILES_BNL801	Constraints for BNL VU9P Development board	constraints/
XDC_FILES_VMK180	Constraints for VMK180	constraints/

4.6.1.4 Helper scripts

The directory scripts/helper contains a set of scripts that should be sourced by other scripts as mentioned above. The different helper scripts are:

clear_filesets.tcl	Initialize all the filesets variables to "" before filesets can be sourced
vivado_import_generic.tcl	Create the Vivado project
questa_import_generic.tcl	Create the Questasim / Modelsim project
sigasi_import_generic.tcl	Create a Sigasi project
do_implementation_pre.tcl	A set of tasks before synthesis, call at the beginning of the do_implementation* script
do_implementation_post.tcl	Will run synthesis. call at the end of the do_implementation* script.
do_implementation_finish.tcl	Called automatically by do_implementation_post.tcl, or in case of ILA probes manually. Will run implementation, bitstream, reports etc.

4.7 Simulation & UVVM

The FELIX firmware is verified per functional block, not as an entire design. There are several testbenches that can be found in the repository.

As a simulation library, UVVM is used to help with utilities and functional models. VUnit is used to streamline the simulation process.

The VUnit simulation has some dependencies:

- Vivado 2021.2, assumed to be installed in /opt/Xilinx/Vivado/2021.2/
- Questasim 2019.1, assumed to be installed in /opt/questasim-2019.1/
- Instead of questasim, you can also use modelsim or GHDL. Some simulations will fail with GHDL due to the encrypted IP of transceivers
- The tcl filesets are loaded in VUnit using python3-tkinter. The package `python3-tkinter` must be installed on the system
- VUnit can be installed by running

```
pip3 install --user vunit-hdl
```

To run a certain testbench

```
#Source Vivado settings:
~/felix/firmware/simulation/VUnit$ source /opt/Xilinx/Vivado/2021.2/settings64.sh
#Add questasim to the path:
~/felix/firmware/simulation/VUnit$ export PATH=$PATH:/opt/questasim-2019.1/bin
#List available simulations in VUnit
~/felix/firmware/simulation/VUnit$ ./run.py -l
lib.ltittc_vunit_tb.all
```

```
lib.lpgbtlinktohost_vunit_tb.all
lib.tb_lcb_command_decoder_vunit.all
lib.decegroup_8b10b_vunit_tb.all
lib.tb_r3l1_regmap_vunit.all
lib.wuppergen4_vunit_tb.all
lib.tb_r3l1_frame_generator_vunit.all
lib.encodingepath_vunit_tb.all
lib.tb_amac_deglitcher_vunit.all
lib.crtohost_vunit_tb.all
lib.ttc_lti_transmitter_vunit_tb.all
lib.busyvirtualelink_vunit_tb.all
lib.decodingpixel_vunit_tb.all
lib.i2c_vunit_tb.all
lib.tb_lcb_regmap_vunit.all
lib.tb_trickle_trigger_vunit.all
lib.tb_lcb_axi_encoder_vunit.all
lib.hgtd_fastcmd_vunit_tb.all
lib.tb_r3l1_scheduler_encoder_vunit.all
lib.crc20_vunit_tb.all
lib.ttctohostvirtualelink_vunit_tb.all
lib.validate_8b10b_vunit_tb.all
lib.crfromhost_vunit_tb.all
lib.loopback25g_vunit_tb.all
lib.tb_lcb_frame_generator_vunit.all
lib.gbtcrcoding_vunit_tb.all
lib.amac_demo_vunit_tb.all
lib.gbtlinktohost_vunit_tb.all
lib.tb_lcb_scheduler_encoder_vunit.all
lib.tb_r3l1_axi_encoder_vunit.all
lib.fullmodetohost_vunit_tb.all
lib.tb_bypass_frame_vvc_vunit.all
lib.tb_strips_configuration_decoder_vunit.all
lib.tb_r3l1_frame_synchronizer_vunit.all
lib.tb_amac_encoder_vunit.all
lib.decodinggearbox_vunit_tb.all
lib.tb_playback_controller_vunit.all
lib.tb_amac_decoder_vunit.all
lib.tb_l0a_frame_generator_vunit.all
lib.wupper_vunit_tb.all
lib.tb_bypass_scheduler_continuous_write_vunit.all
Listed 41 tests
#Run the simulation of your choice in GUI mode
~/felix/firmware/simulation/VUnit$ ./run.py lib.wupper_vunit_tb.all --gui
```

The UVVM testbenches can be found in `firmware/simulation/UVVMtests/tb` (although some testbenches can also be found at other places).

VUnit uses separate testbenches which are simple wrappers around the UVVM testbenches, they can be found in `firmware/simulation/VUnit/tb`

:!table: 4 :numbering:

5. Software Development

5.1 Overview of Software

5.1.1 Register Map Interface

The registermap is defined in the firmware git repository under sources/templates/registers-x.y.yaml. This file is converted by the tool, 'WupperCodeGen' into files for the firmware build itself, and into files for the software. Changes to the register file result in different firmware to be addressed by different hardware. To keep firmware and software compatible only the addition of registers is allowed for minor version changes. Any change of address, deletion of registers or change of names of registers should result in a major version change of the registermap.

Currently we have version 4.x and version 5.x. These are incompatible and WupperCodeGen will generate different sofware for them. The generation happens in the regmap module. It contains a few submodules:

1. wuppercodegen: to generate the files
2. firmware-4: the branch of the firmware module in which registermap 4 is defined
3. firmware-5: the branch of the firmware module in which registermap 5 is defined

To generate new software for a new registermap, make sure you:

1. checkout the regmap module
2. git init submodule
3. git update submodule
4. make sure firmware-4 and firmware-5 contain the regmap(s) you want to use
5. run the 'build.sh' script, to generate the new software files, which depend on the REGMAP_VERSION environment variable
6. commit the newly generated files

general compilation of felix-distribution, see below, will use the generated files. It will NOT re-convert the regmap(s)

5.1.2 Low Level API and Driver

5.1.3 External Dependencies

The FELIX software has a number of dependencies. Some of them come from LCG and need to be updated when we change LCG version, others come from our own externals directory, in which each dependency has its own external-<dependency> git repository. Both LCG and our own externals distribute binary libraries in different versions. The versions we use are defined in the file:

cmake_tdaq/cmake/modules/FELIX-version.cmake

Our current LCG dependencies are for felixcore, felixbus, felixbase. These include qt, sqlite, tbb, boost, python_libs, zeromq and libsodiom. With the exception of qt, these dependencies will disappear once felixcore is no longer supported.

On top of that you need to use the compiler from LCG to build FELIX. The full setup is in

cmake_tdaq/bin/setup.sh

LCG is available on cvmfs. Cvmfs is available in Point1 but for the libraries FELIX does not depend on it. Our distribution copies in all needed libraries. In Point1 one should use the compiler from cvmfs.

Our current externals for felixcore are: concurrentqueue, czmq, libcurl, readerwriterqueue, simplewebserver, spdlog and zyre. Again, these will be removed with ending of the support of felixcore.

Our externals for felix-star and others are: bootstrap, catch, datatables, docopt, fontawesome, highcharts, jquery, jwrite, libfabric, libnuma, mathjs, moment, patchelf, pybind11, simdjson and yaml-cpp.

For python3 we depend on the LCG setup. We setup our own environment in the python_env project.

5.1.4 Driver

5.1.4.1 Overview and Package Dependencies

The FELIX driver (along with those for the ROS and RCD applications) reside in the `atlas-tdaq-software` gitlab group, within the `ROSRCDdrivers` package:

https://gitlab.cern.ch/atlas-tdaq-software/ROSRCDdrivers

Each driver is associated with a dedicated library package. The library for the FELIX device driver (a.k.a. `flx`) is managed by Henk Boterenbrood, with the code to be found (this time in the atlas-tdaq-felix group) at:

https://gitlab.cern.ch/atlas-tdaq-felix/drivers_rcc

The other library which is used by FELIX is `cmem_rcc`, which handled the allocation of large contiguous memory blocks. The library for this can be found at:

https://gitlab.cern.ch/atlas-tdaq-software/cmem_rcc

Full documentation for this package can be found at:

https://edms5.cern.ch/document/336290/3

Internally `cmem_rcc` uses a few additional TDAQ packages:

`rcc_error`

Code:

https://gitlab.cern.ch/atlas-tdaq-software/rcc_error

Documentation:

https://gitlab.cern.ch/atlas-tdaq-software/rcc_error/-/tree/master/doc/rcc_error.pdf

`DFDebug`

Code:

https://gitlab.cern.ch/atlas-tdaq-software/DFDebug

Documentation:

https://gitlab.cern.ch/atlas-tdaq-software/DFDebug/-/blob/master/doc/DFDebug.pdf

5.1.4.2 Compilation

The simplest way to deploy the driver on an architecture is to install the RPM, which will automatically compile the source code with the correct settings. Instructions on how to do this can be found in the user manual.

Should you instead wish to compile by hand you will need to perform the following actions:

1. Clone the `ROSRCDdrivers` package.
2. Navigate to the `src` subdirectory.
3. Within `src`, find a file called `Makefile_64` and make a copy called `Makefile`.
4. Source the TDAQ release environment required as specified here.
5. Clone the `drivers_rcc` package into the ROSRCDdrivers directory
6. Add `drivers_rcc\flx` to the include path by modifying the `EXTRA_CFLAGS` variable within `Makefile`
7. (Optional) if you need to compile against newer versions of any library than are in the TDAQ release, clone the relevant versions as for drivers_rcc
8. Compile the drivers on a server matching the required kernel version with `gmake CC=/usr/bin/gcc`

Please contact Markus Joos if you require additional assistance.

5.1.5 Low Level Tools (flx and ftools)

(Not an exhaustive list, but perhaps a summary of what each tool layer does and who is responsible for them etc)

5.1.6 High Level Tools

(Link to specification doc as preferred)

5.2 Recommended Development Environments & Tools

The FELIX software is split up into projects, each with its own git repository. The full FELIX software distribution is contained in the felix-distribution project which references all needed projects as git submodules. Felix-distribution also contains scripts to create tar, rpm and help files for fully build distributions.

The FELIX project is currently officially supported for x86_64-centos7-gcc11-opt only.

To be able to build and test the full FELIX software project (latest on a branch) follow these steps (using gcc11 as an example):

1. git clone felix-distribution
2. cd felix-distribution
3. git checkout master (or any other branch)
4. git submodule init
5. git submodule update
6. ./pull_all.sh (to get the latest in each project)
7. source python_env/bin/activate
8. source cmake_tdaq/bin/setup.sh
9. cmake_config
10. cd x86_64-centos7-gcc11-opt
11. make -j 8
12. ctest -j 8

if you switch branch, make sure to run:

1. ./checkout_all.sh <branch-name>
2. ./pull_all.sh to get the latest on these branches
3. rm -fr x86_64-centos7-gcc11-opt (as the output dir will contain all kinds of intermediate results of the previus branch)
4. compile again…

Once cmake has been configured as above, individual projects can be build by stepping into (e.g. for felix-star in gcc11) x86_64-centos7-gcc11-opt/felix-star and running make and ctest as needed.

If you wish to develop and individual project independently, the clone and build procedure is the same as for felix-distribution, as shown below for the example of felix-star with gcc8. Note that each project has its own CI, and as these CIs also use submodules to access their dependencies, you therefore need to make sure that each project is up to date to be able to build on its own.

1. git clone felix-star
2. cd felix-star

3. git submodule init
4. git submodule update
5. source python_env/bin/activate
6. source cmake_tdaq/bin/setup.sh
7. cmake_config
8. cd x86_64-centos7-gcc11-opt
9. make -j 8
10. ctest -j 8

the procedure is exactly the same as for felix-distribution, and is the same as is run by the CI runs. If some submodules are out of date you can update them with the following commands:

1. cd <submodule>
2. git checkout master
3. git pull
4. cd ..
5. Try to build and test in x86_64-centos7-gcc11-opt
6. git commit -a -m "Updated <submodule>"

5.3 Projects/Modules

5.3.1 Current modules

cmake_tdaq	General module to setup for compilation and generate the proper makefiles from cmake. This module contains the versions of all LCG and external modules to be used in cmake_tdaq/cmake/modules/FELIX-versions.cmake.
data_transfer_tools	Python scripts to generate block-files of different types and with different data.
drivers_rcc	The client part of the cmem_rcc and felix driver. This modules extracts its source from the driver rpm and compiles it.
elinkconfig	GUI to setup elinks. Reads or writes a felix device and reads or writes elink configuration files. These files can also be used with the feconf utility from the ftools project.
felig-tools	Tools to control the felig (generator) on a felix device.
felix-bus-fs	Second generation bus which uses the filesystem for communication. In use by felix-star as of version 4.2.
felix-client	Client to felixbus-fs and netio-next implementing felix-interface.

felix-client-thread	Implementation of client which searches for the shared library of felix-client and its dependencies and loads it. A similar module exists in the TDAQ. They can this way just put a felix-client in their LD_LIBRARY_PATH and find that one, independent of the version, as long as the felix-interface has not ahcnged. This module also contains the felix_client_thread_extension which contains methods not yet in the public interface.
felix-def	All cross project definitions in c, c++ and python for FELIX. Includes ports, elinks, and conversions of fids.
felix-interface	The actual interface for felix-client. This module is shared with TDAQ. It contains the actual interface. For any new, but not published, methods for the next version, see felix-client-thread. The latter methods can be used by FELIX itself, just not by the TDAQ until they are published.
felix-mapper	Library to convert logical names into FIDs and vice versa. Used by NSW and DCS.
felix-release-notes	Utility to extract ReleaseNotes from JIRA into html format. Used by the FELIX website and to make the release.
felix-star	Event driven, single threaded, readout application. Replaces felixcore.
felix-star-watchdog	Monitoring application. To be replaced by felix-register, included in the felix-star module.
felix-tag	Module to inject the same version number in all of the programs that depend on this module.
felix-unit-test	Unit tests support in Python and C++ to support running tests using supervisord.
flxcard	API and low-level tools for the FELIX card.
flxcard_py	Python interface to flxcard.
ftools	High level tools for the FELIX card.
hdlc_coder	Module to handle HDLC encoding and decoding (parts that are not in the firmware).
netio-next	Event driven RDMA communication library. Replaces netio.
python_env	Python environment for FELIX. Assumes a proper python3 version is picked up from LCG. All extra libraries are installed here.
regmap	Register map generated code. Uses wuppercodegen and registermaps in the firmware repository to create c header files.

tdaq_tools	Additional tools to include in the felix-distribution if there is not TDAQ installed.
wuppercodegen	Tool to generate VHDL, C code, HTML and LaTeX documentation from a single Yaml registermap file.

5.3.2 External Modules

All the external libraries and tools to be used by FELIX. No branches here, but different versions on 'master'. Binaries and Libraries are checked into git for these modules.

external/bootstrap	Styling library for web pages. In use by statistics module in felixcore and felix-star.
external/catch	Testing framework for c++.
external/datatables	Table widget for web pages. In use by statistics module in felixcore and felix-star.
external/docopt	C++ version of docopt command line parsing.
external/highcharts	Graphing widget for web pages. In use by statistics module in felixcore and felix-star.
external/jquery	Framework for webapps. In use by statistics module in felixcore and felix-star.
external/json	Json reader and writer in c++.
external/jwrite	Fast json writer in c.
external/libfabric	RDMA support for mellanox cards. Used by netio and netio-next.
external/libnuma	Numa support for applications.
external/mathjs	Mathematics library in JavaScript.
external/patchelf	Utility to patch binaries and libraries. Used to change the RPATH.
external/pybind11	Python binding for c++ libraries.
external/simdjson	Very fast json reader in c.
external/yaml-cpp	Yaml parser in c++.

5.3.3 Legacy Modules, to be removed for regmap 5.0

felixbase	Base libarry for felixcore and felixbus.
felixbus	Communication Bus to handle lookups of elinks/fids and provide ip and port information. Based on ZeroMQ and Zyre, being replaced by felix-bus-fs.

felixbus-client	Client application for the felixbus.
felixcore	Multi-threaded readout application for FELIX cards. This application is to be replaced by the single threaded event driver version felix-star.
felixpy	Python interface to felixcore and some of the ftools.
netio	Communication library, being replaced by the event-driven netio-next library.
packetformat	Module to decode the FELIX block format(s). In use by felixcore. Felix-star has its own internal functions to decode the blocks.

5.3.4 Legacy External Modules, to be removed for regmap 5.0

external/concurrentqueue	Queue system in use by felixcore.
external/czmq	C interface to ZeroMQ.
external/simplewebserver	Simple web server in c++ in use by felixcore for statistics.
external/spdlog	Logging system in use by felixcore.
external/zyre	Broadcast system on top of ZeroMQ and czmq, in use by felixbus.

5.3.5 Support

clowder	Scripts/cronjobs which poll the testbed machines for hardware information (cards, programmers, etc) and publish this information in the page: https://atlas-project-felix.web.cern.ch/atlas-project-felix/dev/cern-testbed.html
cvmfs-felix	Our own, reduced, cvmfs distribution, for developers to use if no access is available to cvmfs.
cvmfs-felix-test	Test module to see if cvmfs-felix works correctly.
felix-ci-status	Single web pages showing all CI jobs for FELIX for different branches and tags.
felix-image	Docker image of the FELIX installation. Do we still support this ?
gitlab_ci_runner	Docker images with the same software as is there for netboot, as well as images for AsciiDocter and Antora.
gitlab-ci-scripts	Bunch of example scripts to automatize some things in gitlab.
mgmt-mellanox-ofed	Scripts to build the mellanox driver on a specific platform and linux version.

pitrigger	RaspberryPi trigger for the TTCVi. PiTrigger is meant to overcome the limitation of the TTCvi of not having a busy gating input. PiTrigger runs on a RaspberryPi (tested on model 3B+) and outputs random pulses at a selectable rate on GPIO pin (default 22). A second GPIO pin (default 27) is the input of the busy signal. When the busy signal is low the output is gated. The RaspberryPi output has to be converted to NIM logic before being fed to the TTCvi. PiTrigger also implements a routine to measure the maximum Felix performance. In this condition the trigger rate is incremented in stages of selectable duration and frequency rate steps. The run stops when busy is received for at least half the duration of a stage.

5.3.6 Documentation

felix-antora	(WIP) Manual on how to use antora in FELIX.
felix-doc	(WIP) Combined antora documentation container of all FELIX documentation.
felix-developer-manual	This Developers Manual. All documented in AsciiDoc format.
felix-software-doc	Software specification document. All documented in AsciiDoc format.
felix-ui-default	(WIP) Style library to handle conversion with antora.
felix-user-manual	The FELIX user manual. Published to staging when committed to. Published to public when tagged. All documented in AsciiDoc format.
felix-www	The FELIX website. Published to staging when committed to. Published to public when tagged. Most sources in Markdown format. Converted using jekyll into html. Styled using bootstrap.
tdaq_for_flx	Documentation of the felix drivers.

5.4 CI setup

5.5 Implementing Tests

What test handlers do we have available and how to use them.

5.6 Merging and Validation of Changes

What will you require in terms of testing to be happy to merge?

5.7 Release the FELIX software

The release of the felix software is all defined in the felix-distribution project. This project contains all submodules needed to do the release. Different branches, such as 'master' and '5.0' are rebuilt to produce the 'nightly' as defined in the schedule in gitlab. Currently all submodules in felix-distribution on both branches 'master' and '5.0' are all of the 'master' branch. All modules (including felix-distribution) use the environment variable REGMAP_VERSION (either set to 0x0400 or 0x0500) to compile for a particular version. Reason to have a '5.0' branch in felix-distribution is that gitlab-ci does not show this variable, so it would be hard to distinguish between a 'master' ci and a '5.0' ci.

Any commit to 'felix-distribution' will have gitlab build the 'latest' distribution and publish it.

If you tag the 'felix-distribution' gitlab will create a real release (or release candidate).

This runs through the following sequence:

1. check out felix-distribution and get all submodules
2. compile the code
3. test the code (limited tests at this time)
4. generate the ReleaseNotes.html, based on the version number in the environment variable FELIX_VERSION
5. create a tar file
6. create an rpm file
7. create the help tar file
8. untar the tar file and validate it
9. install the rpm file and validate it
10. copy the tar, rpm and help file over to its destination

The destinations are below:

Latest	https://atlas-project-felix.web.cern.ch/atlas-project-felix/user/dist/software/latest/
Nightly	https://atlas-project-felix.web.cern.ch/atlas-project-felix/user/dist/software/nightly/
Release 4.x and 5.x	https://atlas-project-felix.web.cern.ch/atlas-project-felix/user/dist/software/apps/4.x/

TBR: There is also a docker image generated, do we still need this ?

5.7.1 Make a release:

1. Check out felix-distribution and select your branch (master or 4.2.x)
2. `git submodule init`

3. `git submodule update`
4. NOTE: all submodules are of branch 'master', but not the latest, to get the latest do:
5. `./pull_all.sh`
6. To build:
7. source cmake_tdaq/bin/setup.sh
8. check output for the correct version FELIX_VERSION and the correct regmap REGMAP_VERSION
9. Close all issues in JIRA (ReleaseNotes will be generated)
10. Commit and push to check if the branch properly runs through the pipeline in gitlab, producing the "Latest", see https://gitlab.cern.ch/atlas-tdaq-felix/felix-distribution/pipelines
11. Tag the release with the next tag: felix-XX-YY-ZZ, and push the tag
12. Wait for it to build in gitlab to be successful
13. **If not, retag it with a different version number, do NOT re-use tags.**
14. It should appear on our distribution site: `/eos/project/f/felix/www/user/dist/software/apps`
15. Distribute to sites as mentioned below
16. Add a news item to the website, see https://gitlab.cern.ch/atlas-tdaq-felix/felix-www
17. Mark release as released in JIRA
18. Announce on `atlas-felix-tdaq-users`

5.7.2 Copy the distribution

The distribution is to be copied to the following places:

Website

[Automatic] `/eos/project/f/felix/www/user/dist/software/apps/4.x`

TestBed

[Untar] `/tbed/tdaq/felix`

NOTE

Check if flx-info does not give a regmap warning when run on testbed with a rm4 firmware

cvmfs

[Semi-Automatic]

- `/cvmfs/atlas-online-nightlies.cern.ch/felix/nightlies` [automatic]
- `/cvmfs/atlas-online-nightlies.cern.ch/felix/releases`, [automatic]

or by hand, needs privs by Reiner Hauser given to: Mark, Carlo and flx

```
ssh cvatlasonlinenightlies@cvmfs-atlas-online-nightlies.cern.ch felix-04-02-10-rm4
```

```
x86_64-el9-gcc13-opt
```

FLX

[Untar] `/afs/cern.ch/user/f/flx/public/felix/software`, used by DCS

Own area

[Untar] `~/public/felix`, used by copy procedures below

TestBedAtlas

[Install] `/sw/atlas/felix`, only Will, Mark and Carlo can do

```
ssh pc-tbed-cfs-02
cd FELIX/release/felix-tbed
./sync_tbed_dir.sh felix-04-02-10-rm4-stand-alone
```

Point1

[Install] `/sw/atlas/felix`, only Will, Mark and Carlo can do

```
ssh lxplus
scp <tarfile> atlasgw:/shared/data/

ssh atlasgw
pc-atlas-pub
cd /det/tdaq/felix
tar zxvf /shared/data/<tarfile>
./sync_install.sh felix-04-02-10-rm4-stand-alone
```

:!table: 5 :numbering:

6. CERN BLDG. 4 (TDAQ) Testbed Guide

The TDAQ testbed, in the basement of CERN bldg. 4, is the central testing and common development site for the FELIX project. Each institute may also maintain its own setup, but the largest and most comprehensive setup is maintained at CERN. Thus, if your institute does not have a particular piece of hardware, it may be possible to gain access to it in bldg. 4.

6.1 Summary of Available Testing Setups

- TTC crate with both modified TTCvi (substituting for LTP), ALTI and RODBUSY module
  - Able to be connected to drive any full chain setup
- Full chain GBT-mode setup with two FLX-712 cards connected to other FLX-712 cards configured as FELIG data sources.
  - FELIX server connected to switched network at 25 GbE.
- Full chain FULL-mode setup with single FLX-712 card connected to another FLX-712 configured as FMEmu data source.
  - FELIX server connected to switched network at 100 GbE.
- Servers hosting FLX709 (a.k.a. MiniFelix)
- SW ROD production pre-series server with 100 GbE input connected to switched network
- VLDB & NSW L1DDC, able to be deployed flexibly for GBTx and SCA testing

6.2 Gaining Access to Testbed Resources

In order to access the testbed, please request an account by following the instructions on the following page:

https://twiki.cern.ch/twiki/bin/view/Atlas/Lab4Testbed

For a summary of currently active testbed systems, please check the following page:

https://atlas-project-felix.web.cern.ch/atlas-project-felix/dev/cern-testbed.html

To gain physical access to the testbed please make a request for room **0004-S-001** via:

https://adams.cern.ch/

For any general requests or questions regarding the testbed, for instance to book a slot to use a resource, please ask in the the Mattermost channel

https://atlas-project-felix.web.cern.ch/atlas-project-felix/dev/cern-testbed.html

Please do not start working on a testbed system without asking permission first!

0 :!table: 6 :numbering:

7. Documentation System

The FELIX documentation system uses a set of converters to publish both an integrated website as well as individual PDF documents.

7.1 Sources and Formats

The (current) FELIX software and some firmware documentation consists of a number of manuals, which are all in separate gitlab projects:

- The Software Specification: https://gitlab.cern.ch/atlas-tdaq-felix/felix-software-doc
- The User Manual: https://gitlab.cern.ch/atlas-tdaq-felix-dev/felix-user-manual
- The Developer Manual: https://gitlab.cern.ch/atlas-tdaq-felix-dev/felix-developer-manual

Each of these manuals are written in AsciiDoc format (https://asciidoc.org) and their files are organized to be handled by the Antora (https://antora.org) tool.

Chapters are written file by file. Images of different formats can be included. HTML and PDF are generated.

7.2 Conversion

The actual conversion of AsciiDoc is done by AsciiDoctor (https://asciidoctor.org) and produces html. A similar AsciiDoctor-pdf tool (https://docs.asciidoctor.org/pdf-converter/latest/) converts each manual to a single PDF file.

Antora orchestrates the actual conversion. To convert locally to HTML you need an installation of NPM, Antora and LUNR:

```
npm install --global @antora/cli @antora/site-generator @antora/site-generator-default
git-describe
npm install @antora/lunr-extension tar-stream fs-extra
```

To convert locally to PDF you need an installation of Ruby and ascidoctor-pdf:

```
gem install asciidoctor asciidoctor-pdf
```

You can also use FELIX's images which includes these tools:

- NPM, Antora, LUNR and asciidoctor: gitlab-registry.cern.ch/atlas-tdaq-felix/gitlab_ci_runner:antora3-lunr-ext
- Ruby and asciidoctor-pdf: gitlab-registry.cern.ch/atlas-tdaq-felix/gitlab_ci_runner:ruby-asciidoctor

You can then run:

```
./make-html.sh
./make-pdf.sh
```

which produces the website for the current checked out files of the manual in

- build/site/index.html

including an attached PDF file, unless you did not run make-pdf.sh

If files are committed and pushed to the repository the CI will run the conversions for you and the output of for instance "felix-developer-manual" branch "master" can be found on:

- https://atlas-project-felix.web.cern.ch/atlas-project-felix/user/manuals/felix-developer-manual/master

7.3 Branches and Tags

There is no concept of a staged manual. If you make changes to the repository and push them they get converted and published. Branches such as 5.x and 4.2.x have therefore the latest documentation. If conversion fails the previous website and document is still available.

If you want to check out changes, create a branch and push it. It will them be available under:

- https://atlas-project-felix.web.cern.ch/atlas-project-felix/user/manuals/<manual_name>/<branch>/

If you need to publish a fixed version of the manual, run:

```
./make-tag.sh
```

which sets up the antora files for the tag (version), commits and tags it, resets the antora files back to the branch (e.g. 1.x) and commits it again. All of this gets pushed.

```
This tag is then published under:
```

- https://atlas-project-felix.web.cern.ch/atlas-project-felix/user/manuals/<manual_name>/<tag>/

The URLs used here are normally not used by the FELIX users as they would prefer access to the integrated documentation, see below.

7.4 Integration

Antora can integrate several manuals and versions of them into a single website. In the bottom left corner of each of the pages you can switch between manuals and their versions. Antora creates this site by accessing each of the gitlab repositories of each of the manuals. It therefore has access to all versions of each manual. Each manual has its own PDF (see below), however there is no integrated

PDF of all manuals.

The integration project itself is on gitlab under:

- https://gitlab.cern.ch/atlas-tdaq-felix-dev/felix-doc

You can convert this locally (you need antora) by running:

```
./make-html.sh
```

which produces the integrated website under:

- build/site/index.html

If files are committed and pushed to the repository the CI will run the conversion for you and the integrated output can be found on:

- https://atlas-project-felix.web.cern.ch/atlas-project-felix/user/felix-doc

which is the official FELIX website for the manuals.

The CI felix-doc project is triggered every time there is a commit to itself or to any of its manuals.

7.5 PDF

Antora does not know about PDF, therefore the asciidoctor-pdf converter is run for each manual and version individually at the time of commit/push to the repository. The PDF files are stored for each manual and version combination in:

- https://atlas-project-felix.web.cern.ch/atlas-project-felix/user/dist/software/pdfs/

When antora builds the fully integrated website it downloads the earlier generated PDFs and attaches them to the integrated website.

7.6 Extensions

To enable Antora to handle its conversion of individual manuals and integration of them we use a number of extensions. These (if used) have to be declared in the antora-playbook.xml file in the same order as listed below.

7.6.1 Search extension

To enable webwide search for individual manuals as well as for the integrated website you need to include:

- @antora/lunr-extension

This is a generic extension of antora and does not need to be a subproject.

7.6.2 Check Config extension

To make sure a subproject has the proper branch/tag and is also referred to as such in the integration project we check the configuration with this extension. Any misconfigurations are given as errors/warnings.

This ANTORA extension needs to be a subproject of the manual (and of felix-doc). Further info under:

- https://gitlab.cern.ch/antora/check-config-extension

7.6.3 Set Version extension

This runs either "git describe" on the repo to get an accurate version number, or in the case of felix-doc dowloads the "same" version number from a file from the PDFs directory:

- https://atlas-project-felix.web.cern.ch/atlas-project-felix/user/dist/software/pdfs/

The file is version.adoc and can thus be included in the generated website and PDF.

This ANTORA extension needs to be a subproject of the manual (and of felix-doc). Further info under:

- https://gitlab.cern.ch/antora/set-version-extension

7.6.4 Custom Format extension (Numbering sections)

By default antora handles AsciiDoc files without any section/subsection numbering. The antora numbering restarts for each AsciiDoc file conversion, and thus is wrong. PDF does not have this problem as it regards the AsciiDoc manual as one file.

This extension numbers the sections and subsections file by file, where each file just needs the chapter number set.

This ANTORA extension needs to be a subproject of the manual (and of felix-doc). Further info under:

- https://gitlab.cern.ch/antora/custom-format-extension

7.6.5 Helpfile extension

To include help output from FELIX commands in the documentation we access some files which contains those help text files and which are created as part of the felix-distribution.

The FELIX command needs to be included in build-tar.sh of felix-distribution. The command also needs to be included in the build-help.sh. It will then end up in a file like this

- felix-master-rm5-x86_64-centos7-gcc11-opt-help.tar.gz

on the FELIX distribution site:

- https://atlas-project-felix.web.cern.ch/atlas-project-felix/user/dist/software/latest/

The Helpfile extension downloads such a file, extracts all the help files from it and adds them as virtual files for Antora. In the AsciiDoc sources of the manual one can then include the help files one by one.

This ANTORA extension needs to be a subproject of the manual (and of felix-doc). Further info under:

- https://gitlab.cern.ch/antora/helpfile-extension

7.6.6 Export Content extension

To convert to pdf real AsciiDoc files are needed. The above extensions have generated a bunch of virtual files (help, version, ...) in the virtual files system of Antora.

This extension exports all files (and renames a few) to a directory to be able to run AsciiDoctor-pdf on.

This ANTORA extension needs to be a subproject of the manual (and of felix-doc). Further info under:

- https://gitlab.cern.ch/antora/export-content-extension

7.6.7 PDF Download extension

As the full Antora integration module (felix-doc) can only assemble the html for the website, the previously uploaded PDF files need to be downloaded again when the integrated site is created.

This extensions downloads any PDFs needed from:

- https://atlas-project-felix.web.cern.ch/atlas-project-felix/user/dist/software/pdfs/

This ANTORA extension needs to be a subproject of felix-doc. Further info under:

- https://gitlab.cern.ch/antora/pdf-download-extension