dv-sdk

Release 1_7_0

iniVation AG

Jul 16, 2025

DV-SDK

CHAPTER

ONE

DV-SDK

1.1 Installation

In order to develop modules for DV[1], the `dv-sdk` library is required. For this purpose, a working C++ 20 toolchain must be installed on the system. This library is currently only available on *Mac* and *Linux* systems. This means using one of the following:

- Apple clang version 14 or newer (on Mac)
- gcc version 10.0 or newer
- clang version 13 or newer

1.1.1 MacOS

Prerequisites

The recommended toolchain is the standard toolchain provided by Apple with XCode. To install this, run:

```
xcode-select --install
```

We also require *cmake* to build applications. The easiest way to install *cmake* is via Homebrew[2]. Run:

```
brew install cmake
```

Library Installation

The `dv-sdk` library is included in the *installation of dv-runtime*.

If you have a mac with Apple Silicon processor, the development runtime will run natively on it.

1.1.2 Linux

Make sure you have a C++ build environment, including *cmake*, installed on your system.

Ubuntu Linux

Ubuntu Linux

We provide a PPA repository[3] for Focal (20.04 LTS), Jammy (22.04 LTS) and Noble (24.04 LTS) on the x86_64, arm64 and armhf architectures.

[1] https://docs.inivation.com/software/dv/index.html
[2] https://brew.sh
[3] https://launchpad.net/~inivation-ppa/+archive/ubuntu/inivation

```
sudo add-apt-repository ppa:ubuntu-toolchain-r/test
sudo add-apt-repository ppa:inivation-ppa/inivation
sudo apt-get update
sudo apt-get install dv-runtime-dev
```

Fedora Linux

We provide a COPR repository[4] for Fedora 34, 35, 36 and rawhide on the x86_64, arm64 and armhf architectures.

```
sudo dnf copr enable inivation/inivation
sudo dnf install dv-runtime-devel
```

Arch Linux

The required files to use `dv-sdk` are provided via the *dv-runtime installation*.

Gentoo Linux

The required files to use `dv-sdk` are provided via the *dv-runtime installation*.

1.2 DV Modules

1.2.1 Create your Own DV Module

In this documentation we show you how to:

1. Create your *First Module*.
2. Understand and use the general *Module API*.
3. Understand and use the *I/O API*.
4. Understand and use the *Configuration API*.
5. Understand and use the *Logging API*.

Write a First Module

Prerequisites

- A working installation of DV on your computer
  - **macOS** Make sure to also install the `dv-runtime` package via brew
  - **Windows** Development on Windows is not yet supported
- Knowledge of C++ programming
- A working C++ toolchain for your system, including *cmake*
- Your C++ editor or IDE of choice

[4] https://copr.fedorainfracloud.org/coprs/inivation/inivation/

Start a new project

A demo module that counts events can be found on Gitlab. *(https://gitlab.com/inivation/dv/dv-example-cpp)*.

Clone the start module repository with git

```
git clone https://gitlab.com/inivation/dv/dv-example-cpp.git
```

Rename project

After downloading, let's rename the project first. First, rename the project folder

```
mv dv-example-cpp my-first-module
```

after that, rename the project name in the `CMakeLists.txt` file by changing the line `PROJECT(dv-example-cpp C CXX)` with your favourite text editor. For example, you could change it to `PROJECT(my-first-module C CXX)`.

Command Line Build

To build the demo module from terminal, use the following commands:

```
cd my-first-module
cmake .
make -j2 -s
sudo make install
```

If everything goes well, you should end up with a file `my-first-module.(so|dylib)` in the `modules` directory.

IDE

QT Creator

Click *File -> Open File or Project...* and select the projects *CMakeLists.txt*. Configure the project by selecting the builds you want to create and specify their location or use the default.

CLion

Click *File -> Open* and select the project root directory. CLion should automatically detect the project as a CMake project and try to run `cmake` in the project folder.

Eclipse

Eclipse does not support *cmake* natively yet. It does however support *Makefile* projects. Since *cmake* essentially generates a *Makefile*, we can run cmake in the command line and open the resulting project in Eclipse.

Run cmake in the project directory

```
cmake .
```

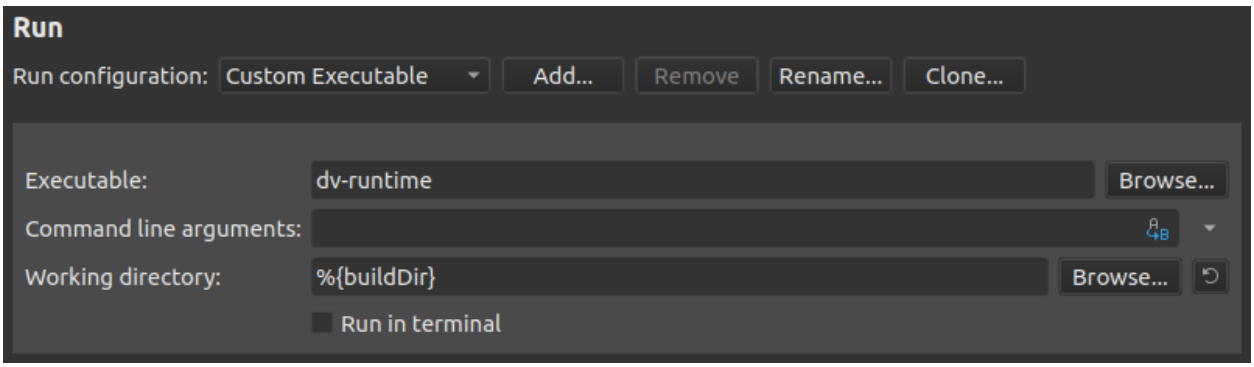
Now, you can import it into Eclipse with *File -> Import... -> Existing Code as Makefile Project*.

Other

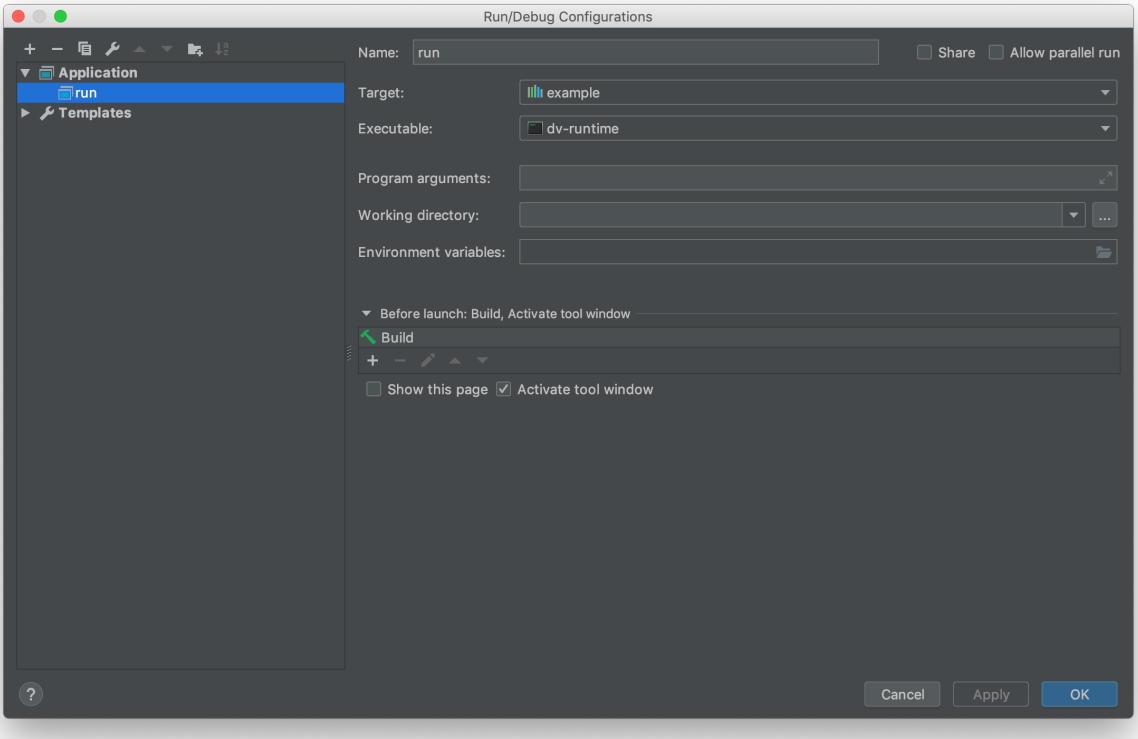
The project is a simple *CMake* project. If your IDE supports CMake, it should be capable of opening the project out of the box.

Run configuration

Set your IDEs run configuration to launch `dv-runtime`. `dv-runtime` is (most likely) in `/usr/bin` (Linux) or `/usr/local/bin` (macOS).



Run configuration for QT Creator (Projects -> Build & Run -> Run).



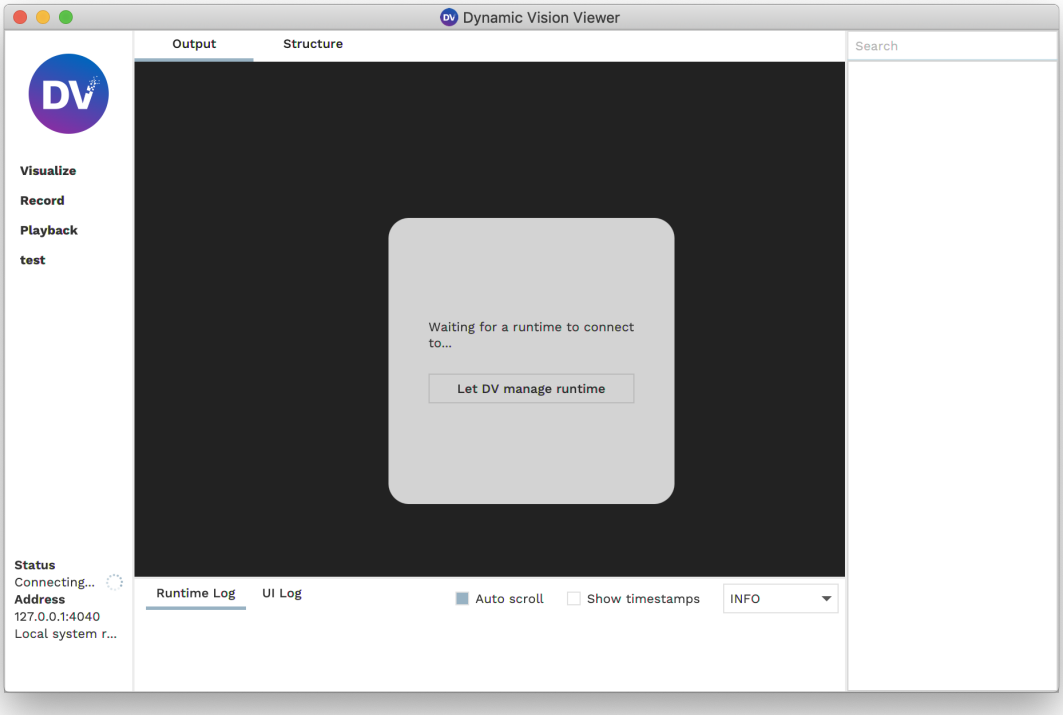
Run configuration for CLion.

To have DV load modules automatically when running the project within an IDE, add `DV_MODULES_PATH=$PATH_TO_MODULE_PROJECT` to **Environment variables**. ($PATH_TO_MODULE_PROJECT is a path to the project root directory of your new module.)

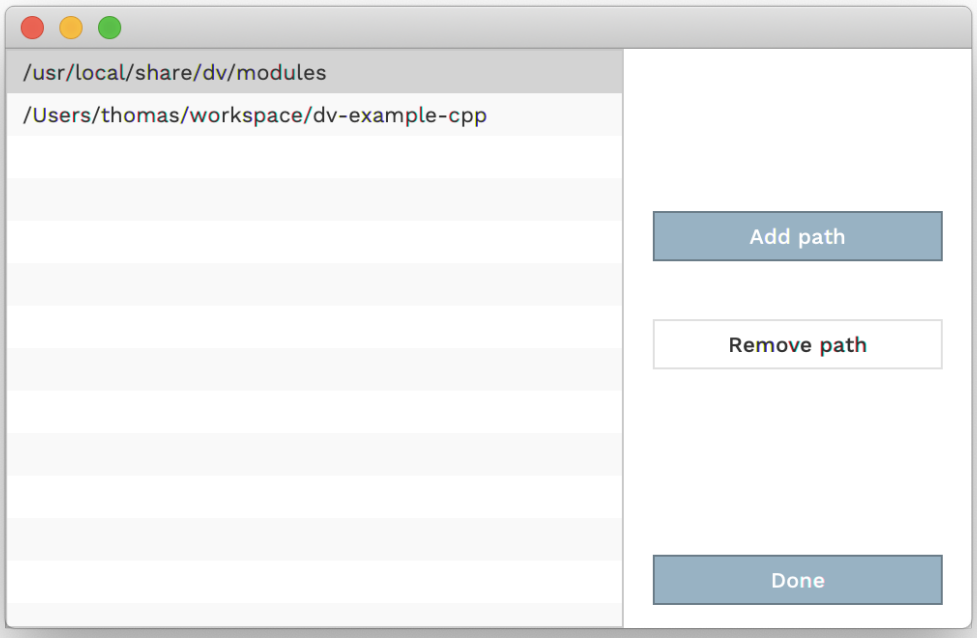
Set up DV

To iteratively test a module, you will have to set up DV the following way

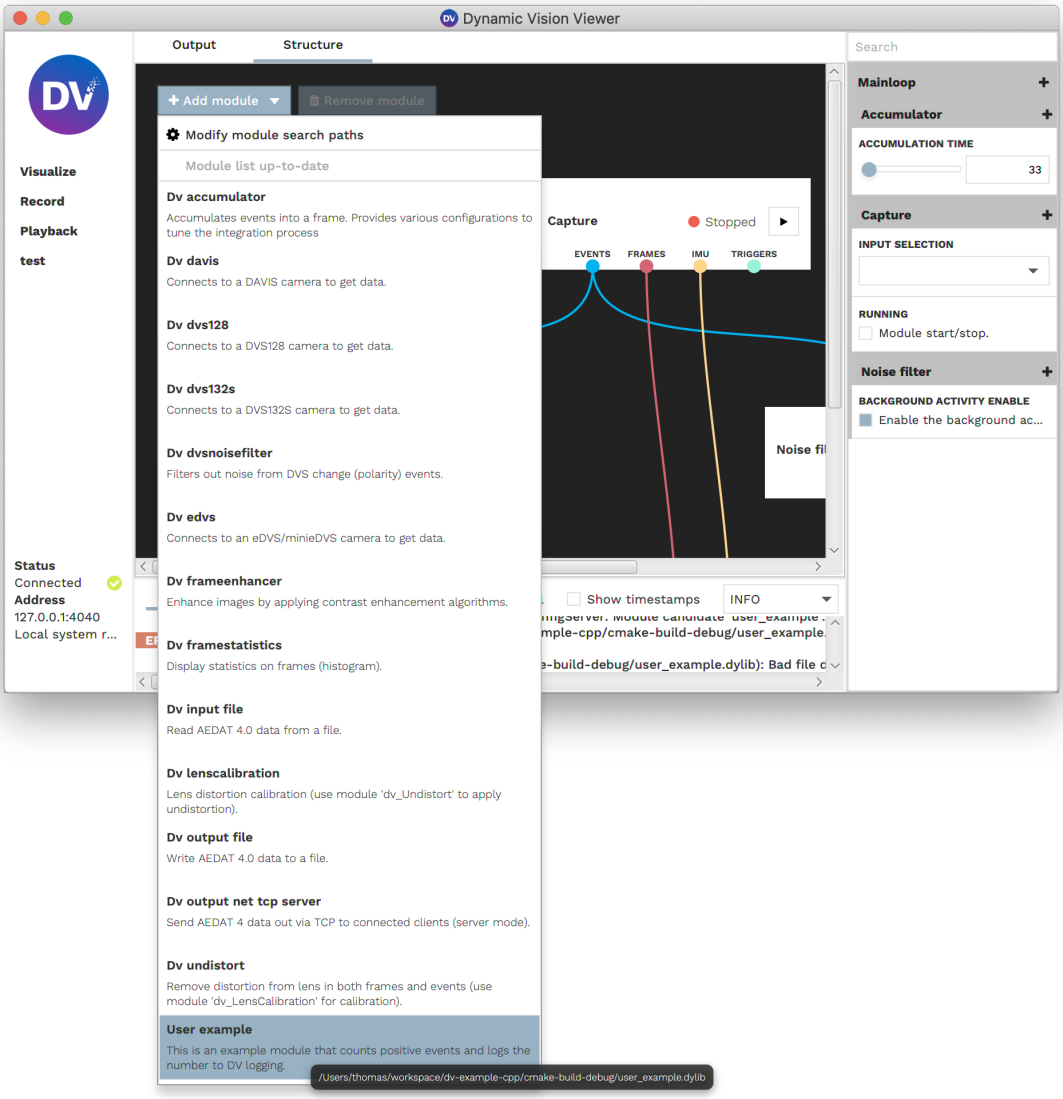
1. Launch DV
2. Disable *Connect to -> Manage local runtime instances*. This tells DV to not start its own runtime, and just wait for someone else to start a runtime in its behalf.
3. Select *Shutdown* if DV asks to shut down the current instance.
4. DV is now waiting for a runtime to be started externally, and will connect as soon as one is started



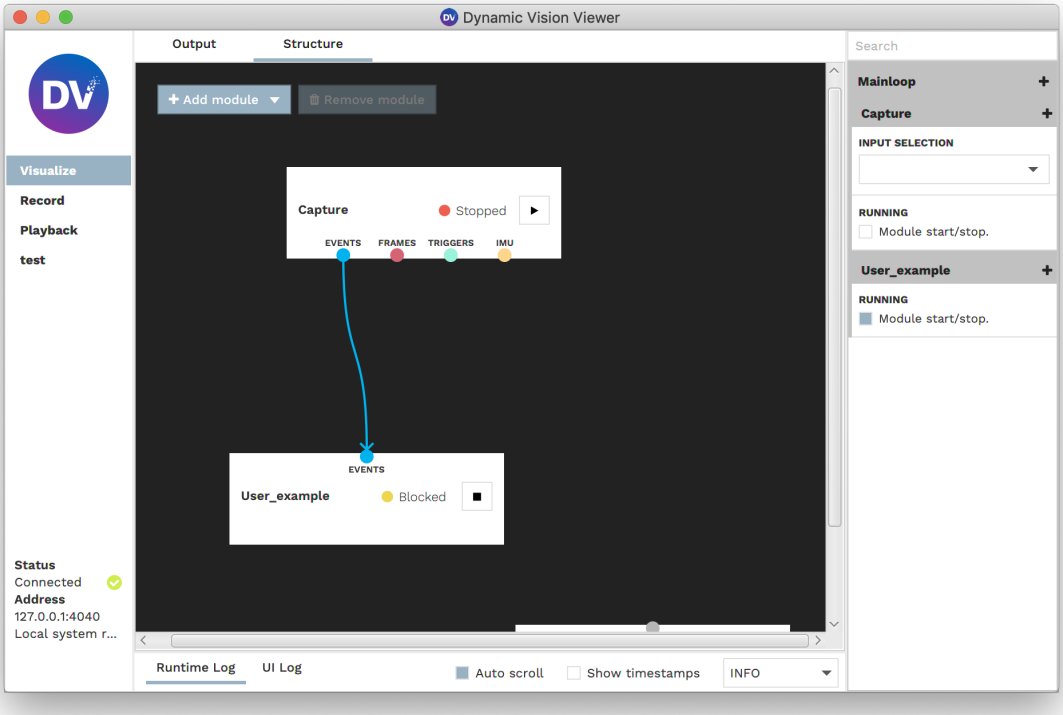
5. Start your runtime by clicking the play button in your IDE. Make sure to have set up your IDE run configuration as described above.
6. As soon as DV connects to the runtime, select the *Structure* tab, click on *Add module* and select *Modify module search path*.
7. Click on *Add path* and add the path where you compiled your modules project directory. Click on *Done*.



8. Click on `Add module` again. If your module compiled succesfully, it should show up in the list of available modules to add. Click on your module name.



9. Connect the input and outputs of your module. Drag the connections from outputs into your modules inputs, and drag connections from your modules outputs to other inputs.



10. Click the *Play* Button on your module to start it.
11. If your module is working, you should see your expected outputs.

Adapt the module

Getting event data from an input

Event data gets passed between modules in packets. The size of event packets is decided by the interval the preceeding module emits the packets. The `run` function gets executed whenever there is new data available, or, when nothing happens, periodically.

Every module has inputs. Inputs are defined in the `initInputs` function. The demo module we just compiled before, has an *events* input called "events". To access the most recently arrived data packet on this input and iterate over the events, override the `run` function as such:

```
void run() override {
    auto inData = inputs.getEventInput("events").events();

    for (const auto &event : inData) {
        // do something with the events
    }
}
```

Getting meta data from an input

Input do not only provide raw data, they also provide (static) meta information, such as width and height dimensions. For example, to get the width and height of the event input, write:

```
int width = inputs.getEventInput("events").sizeX();
int height = inputs.getEventInput("events").sizeY();
```

There is a simple counting algorithm already implemented in the example project

Adding outputs

Modules can not only take data, they can also output data to subsequent modules. Module outputs are defined in the static `initOutputs` function. Since the example module we worked on so far, does not have an `initOutputs` function, we add it like so

```
static void initOutputs(dv::OutputDefinitionList &out) {
    out.addEventOutput("events");
}
```

The `out.addEventOutput("events");` line adds an output of the event type, with the name "events" to the module. Contrary to the input case, we're not entierly done yet. Any output you define in your module has to be assigned its required meta data. For example, an event output needs to have an assigned width and height to it. It is common that you would want to setup the output with the same dimensions as an input to the module. Since the information to what is actually connected to an input is only available at run time, we do not perform the setup call in the `initOutputs` function, but rather in the classes constructor.

Since our class does not have a constuctor yet, we create one like so:

```
ExampleModule() {
    outputs.getEventOutput("events").setup(inputs.getEventInput("events"));
}
```

The line in the constructor body sets up the event output named "events" to the same parameters as the event input named "events". To set the output up with different parameters, you can call a different setup function like

```
outputs.getEventOutput("events").setup(346, 260, "Data from example module");
```

Adding config options

Config options are configurable parameters that are displayed in the gui for easy access. Config options are defined in the `initConfigOptions` function. In our example app, the `initConfigOptions` function already exists. The value of a config option can be accessed in the run function by calling `config.getInt` (or similar for other datatypes).

```
static void initConfigOptions(dv::RuntimeConfig &config) {
        config.add(
                "printInterval", dv::ConfigOption::intOption(
                                                        "Interval in number of␣
↪events between consecutive printing of the event number.", 10000));
}
```

A note on performance: Looking up the value of a config option requires an access to a hashmap with a string key. This is an efficient operation, but in case of event-by-event processing, it still adds considerable overhead. To solve this issue, one can override a function `void configUpdate()` which gets called whenever a config value is changed by the user.

Use this function to look up config values and copy them into local variables to improve performance in the `run` function. Example:

```
void configUpdate() override {
    printInterval = config.getInt("printInterval");
}
```

Example: Create a refractorary period filter

The complete code for this example can be accessed at gitlab.com/inivation/dv/dv-tutorial-code[5]

Time to make our example module do something useful. A refractory period filter limits the maximum firing rate of a pixel. Whenever an event at a pixel passes through the filter, all subsequent events are discarded until the refractory period is over.

Add private data members

Let's change the private data members of our class to store the data we need to function

```
private:
    // user selectable refractory period in microseconds
    long refractoryPeriod;
    // a matrix storing the last firing times for every pixel
    dv::TimeMat lastFiringTimes;
```

The first integer `refractoryPeriod` stores a configuration value, where the user can set the refractory period to a user defined value. The second member `lastFiringTimes` is of the `dv::TimeMat` type. `dv::TimeMat` is a simple matrix type that stores a 2D array of 64bit integers. We usually use the OpenCV `Mat` types for handling 2D data. OpenCV does not provide a 64bit integer type, which is why we provide the `dv::TimeMat` type. Make sure to include

```
#include <dv-sdk/processing.hpp>
```

Set inputs and outputs

Our refractory period filter needs exactly one event input and one event output. We define them as follows:

```
static void initInputs(dv::InputDefinitionList &in) {
    in.addEventInput("events");
}

static void initOutputs(dv::OutputDefinitionList &out) {
    out.addEventOutput("events");
}
```

Set description

Let's set the description of the refractory period filter

```
static const char *initDescription() {
    return "This module filters events by applying a refractory period to the event␣
↪timestamps.";
}
```

[5] https://gitlab.com/inivation/dv/dv-tutorial-code

Set configuration options

Our refractory period filter should have exactly one configuration option, the option to set the refractory period. The type of the option is long and we set a default of 10ms with sensible range from 1ms to 1000ms.

```
static void initConfigOptions(dv::RuntimeConfig &config) {
        config.add(
                "refractoryPeriod", dv::ConfigOption::longOption("Refractory period␣
↪to apply to events (in ms)", 10, 1, 1000));

        config.setPriorityOptions({"refractoryPeriod"});
}
```

The `setPriorityOptions` call only makes sure that the config option is exposed to the right side bar of the gui by default. If you do not specify this, you access the option in the gui by clicking on the black plus icon.

Set constructor

The job of the constructor is all about initializing private data members as well as outputs. We initialize the `lastFiringTimes` array as well as the output to the same dimensions as the *events* input.

```
ExampleModule() : refractoryPeriod(0), lastFiringTimes(inputs.getEventInput("events").
↪size()) {
        outputs.getEventOutput("events").setup(inputs.getEventInput("events"));
}
```

Define the config update

The `configUpdate` function gets called at the start (before run) as well as whenever the config is changed. In our case, only when the user changes the *refractoryPeriod* configuration. In the function, we look up the new value and store it in the private data member. We could look up the value in the run function as well, but since config changes are quite rare, it makes sense to only do it when there is a change.

```
void configUpdate() override {
    refractoryPeriod = config.getLong("refractoryPeriod") * 1000;
}
```

We multiply the value by 1000, to convert the value from miliseconds to microseconds.

Implement run function

The `run` function is where the actual processing happens. Our refractory period filter is pretty simple:

```
void run() override {
    auto input = inputs.getEventInput("events");
    auto output = outputs.getEventOutput("events");

    for (const auto &event : input.events()) {
        if ((event.timestamp() - lastFiringTimes.at(event.y(), event.x())) >␣
↪refractoryPeriod) {
            lastFiringTimes.at(event.y(), event.x()) = event.timestamp();
            output << event;
        }
    }
```

(continues on next page)

(continued from previous page)

```
    output << dv::commit;
}
```

If the time between the current event and the last firing time at that position is larger than the refractory period, update the last firing time for that pixel and append the event to the output.

After we appended all events the output, calling `output << dv::commit;` sends the events out in a packet to the next module.

Congratulations, you built your first useful module!

The complete code for this example can be accessed at gitlab.com/inivation/dv/dv-tutorial-code[6]

Resources

- Minimal example module: *gitlab.com/inivation/dv/dv-example-cpp*[7]
- Module with header / code separation: *gitlab.com/inivation/dv/dv-tutorial-code/-/tree/master/color-paint-example*[8]

Module API

A DV module is processing unit that takes some streams as inputs, does operations on them, and outputs streams as outputs. Typically, a module takes on polarity event data, does computer vision algorithms on the data and outputs the results. Input and outputs can have arbitrary types.

Module Basics

Every DV module inherits from the `dv::ModuleBase` class. In order to register a module class (that inherits from `dv::ModuleBase`) as a DV module, the following macro has to be called at the end of the file where the class is defined:

```
registerModuleClass(RefractoryPeriodFilter)
```

Fields

Inheriting from the this class gives access to the following fields

Field	Type	Purpose
`config`	dv::RuntimeConfig	Access config values from config options.
`log`	dv::Logger	A configured logging instance to log values to DV
`inputs`	dv::RuntimeInputs	Access to the inputs of the module to get data
`outputs`	dv::RuntimeOutputs	Access to the outputs of the module to send data

Static Methods

Every module has to provide some static methods to work properly with DV. The static methods must have the exact footprint as described in the list below. **You have to provide these methods.**

---

[6] https://gitlab.com/inivation/dv/dv-tutorial-code
[7] https://gitlab.com/inivation/dv/dv-example-cpp
[8] https://gitlab.com/inivation/dv/dv-tutorial-code/-/tree/master/color-paint-example

```
static const char *initDescription()
```

Return a description of what the module does **This function is required**

```
static void initInputs(dv::InputDefinitionList &in)
```

Define the inputs to the module

```
static void initOutputs(dv::OutputDefinitionList &out)
```

Define the outputs to the module

```
static void initConfigOptions(dv::RuntimeConfig &config)
```

Define configuration options for the module

Overridable Methods

`dv::BaseModule` exhibits two overridable functions to implement the functionality of the module.

```
void run()
```

Gets called periodically or as new data arrives. Should do the processing. **This function is required**

```
void configUpdate()
```

Gets called whenever the user changes the configuration. Used to read new configuration values and store them in instance data members.

I/O API

Inputs

Every module can have zero or more inputs, as well as outputs. Inputs take data from other modules, outputs push data to subsequent modules. Inputs and outputs preferrably have one of the predefined types, but can in general take on any custom flatbuffer defined type.

The predefined types are

- Event
- Frame
- IMU
- Trigger
- BoundingBox

Input definition

Inputs can **only** be defined in the `initInputs` static function. The number of inputs as well as their types and names must be fixed for every module, and can not be changed at runtime. However, to take on a variable number of input signals, inputs can me marked as optional. In that case, the code has to check at runtime if an input is connected.

Inputs get defined in the `static void initInputs(dv::InputDefinitionList &in)` function. The argument is a modifiabable container that offers functions to add inputs of different types:

```
in.addEventInput(const std::string &name, bool optional = false)
in.addFrameInput(const std::string &name, bool optional = false)
in.addIMUInput(const std::string &name, bool optional = false)
in.addTriggerInput(const std::string &name, bool optional = false)
in.addBoundingBoxInput(const std::string &name, bool optional = false)
```

To add an input for a custom flatbuffer type, use the following function. The type identifier is the four character type identifier string.

```
in.addInput(const std::string &name, const std::string &typeIdentifier, bool optional␣
↪= false)
```

Inputs at runtime

Inputs can be accessed in constructor, `run` and `configUpdate` functions. Inheriting from `dv::ModuleBase` provides an object called `inputs`.

To retrieve an input with a known name and type, call the respective function

```
const auto input = inputs.getEventInput(const std::string &name);
const auto input = inputs.getFrameInput(const std::string &name);
const auto input = inputs.getIMUInput(const std::string &name);
const auto input = inputs.getTriggerInput(const std::string &name);
const auto input = inputs.getBoundingBoxInput(const std::string &name);
```

To retrieve an input for a custom flatbuffer type `<T>`, call

```
const auto input = inputs.getInput<T>(const std::string &name);
```

Input APIs

Not all input types share the same information. Therefore, the APIs for the different input types differ slightly.

Events

Frames

IMU

Triggers

Bounding Box

Data

```
const auto inEvents = input.events();
```

Returns an (random access) iterable data structure of the latest events to arrive at the input. One can convenently loop over the arrays with `for (const auto &event : input.events()) {}`.

The returned value can also be implicitly converted to a `const dv::EventStore`. It can directly be passed to any function that takes a `const dv::EventStore&`

Meta data

```
int width = input.sizeX();
int height = input.sizeY();
cv::Size size = input.size();
```

Return the width / height of the input data. The maximum x-coordinate of an event at this input is `input.sizeX() - 1`. This value is depends on what module / camera is attached to the input at runtime. E.g. In case of a low-resolution *DVS128* it would be 128, in case of a high-resolution *DVS346* it would be 346. The same holds for the y-coordinate.

Frame Data

```
const auto frame = input.frame()
```

Individual Frame Meta Data

The following methods for accessing the frame contents are supported:

- `const auto &pixels = frame.pixels();` A reference to a vector type for raw pixel access
- `cv::Mat *matPtr = frame.getMatPointer();` A pointer to an *OpenCV* `Mat` representing this frame

Each frame has additional metadata, that can differ for each frame

- `long timestamp = frame.timestamp();` The timestamp of the frame
- `long timestamp = frame.timestampStartOfFrame();` The timestamp of the start of frame
- `long timestamp = frame.timestampEndOfFrame();` The timestamp of the end of frame
- `long timestamp = frame.timestampStartOfExposure();` The timestamp ofthe start of the exposure
- `long timestamp = frame.timestampEndOfExposure();` The timestamp ofthe end of the exposure
- `dv::FrameFormat format = frame.format();` The format of the frame. Options are *GRAY*, *BGR*, *BGRA*
- `int width = frame.sizeX();` The width of this specific frame. The width can be smaller or equal to the input width, but never larger.
- `int height = frame.sizeY();` The height of this specific frame. The height can be smaller or equal to the input height, but never larger.

Input Meta Data

```
int width = input.sizeX();
int height = input.sizeY();
cv::Size size = input.size();
```

Returns the maximum dimensions of the input. Individual frames on the input can be smaller than this, but never bigger.

Data

```
input.data()
```

Retrieved the newest data on the this input. The returned data type is (random access) iterable. A convient way to iterate through the newest data is to call `for (const auto &sample : input.data()) {}`

Data

```
input.data()
```

Retrieved the newest data on the this input. The returned data type is (random access) iterable. A convient way to iterate through the newest data is to call `for (const auto &sample : input.data()) {}`

Data

```
input.data()
```

Retrieved the newest data on the this input. The returned data type is (random access) iterable. A convient way to iterate through the newest data is to call `for (const auto &boundingBox : input.data()) {}`

Input Meta Data

```
int width = input.sizeX();
int height = input.sizeY();
cv::Size size = input.size();
```

Returns the maximum dimensions of the input. A bounding box on this input is always in respect to these dimensions.

Common for all inputs

The following functions are available for all input types, including custom flatbuffer types.

```
input.isConnected()
```

For optional inputs, this function returns true iff the input is connected to an output in the current runtime.

```
input.getOriginDescription()
```

Returns a string describing the origin of the data. In most cases, this string will give some information about the original creator of the data, like serial number of the camera. However, this is not guaranteed, as every module the data passes through is allowed to alter the string it passes down in any way.

```
input.data()
```

Advanced use Retrieved the newest data on the this input. The returned data type is equivalent to a shared pointer type. Dereferencing the return value gives the flatbuffer type `<T>` of the latest data.

```
input.infoNode()
```

Advanced use Returns the underlying config trees info node about the output connected to this input. This info node provides information about dimensions etc about the input. Convenience functions such as the event inputs `sizeX` etc. are based on this information. An input of a custom flatbuffer type can have arbitrary meta information about the connection, which an be obtained from the info node. For example, to get an integer with key *sizeX* from the info node, call `input.infoNode().getInt("sizeX")`

Outputs

Output definition

Outputs can only be defined in the static `initOutputs` function. The number, names and types of outputs have to be constant during runtime.

Outputs are defined in the `static void initOutputs(dv::OutputDefinitionList &out)` function. The argument is a modifiabable container that offers functions to add outputs of different types:

```
out.addEventOutput(const std::string &name)
out.addFrameOutput(const std::string &name)
out.addIMUOutput(const std::string &name)
out.addTriggerOutput(const std::string &name)
out.addBoundingBoxOutput(const std::string &name)
```

To add an output of a custom flatbuffer type, use the generic function with your custom types four-character type identifier.

```
out.addOutput(const std::string &name, const std::string &typeIdentifier)
```

Output setup

Upon initialization of your module, **outputs have to be set up in the constructor**. Setting up outputs means assigning them required meta information such as dimensions and source identifiers. In most cases, one wants to set up an output in terms of an input. For example, a filter module takes events of a certain dimension and wants to emit events with the same dimension. This is why setup takes place at initialization time, rather than static.

Event, Frame and Bounding Box outputs

```
output.setup(int sizeX, int sizeY, const std::string &originDescription)
```

Sets up the output with width `sizeX` and height `sizeY` and the supplied origin description. It is advisable to copy the origin description from an input, as the purpose of this field is to keep track of the original creator of the data.

```
output.setup(const RuntimeInput &input)
```

Sets up the output with the same parameters as a compatible input. Event, Frame and Bounding Box inputs are compatible with each other. For example, setting up a frame output with the same parameters as an Event input would look like `outputs.getFrameOutput("frames").setup(inputs.getEventInput("events"));`

Other outputs

```
output.setup(const std::string &originDescription)
```

Sets up the output with the given origin description. It is advisable to copy the origin description from an input, as the purpose of this field is to keep track of the original creator of the data.

Advanced use This is sufficient for the provided IMU and Trigger types. For a custom flatbuffer type, you may require additional information to be present in the output info config node. To set up these custom fields, put them in the output info node obtained by `output.infoNode()`

Send data to outputs

Data can be sent to outputs at any time in the run function. One can send as many data packets as needed (or none). Any sending of data has to happen on the thread where the run function runs on.

Sending data is easy. Depending on the datatype, we expose different convenience functions to use.

Events, IMU, Trigger

To send data to an output, simply stream the data to the desired output. In case of events, IMU, and trigger data, the piped data elements are first appended to an out packet. They only get sent out when piping in `dv::commit`.

```
outputs.getEventOutput("events") << event1 << event2 << dv::commit;
```

Alternatively, one can specifically obtain the the output container, append to it and commit it.

```
auto outEvents = outputs.getEventOutput("events").events();
outEvents.push_back(event1);
outEvents.push_back(event2);
outEvents.commit();
```

After committing, the output container gets reassigned and can be used again immediately after.

For IMU and Trigger outputs, one uses the `data()` function instead of the `events()` function. (The `data()` function is also available for events and functually identical).

```
auto outIMU = outputs.getIMUOutput("imu").data();
outIMU.push_back(imu1);
outIMU.push_back(imu2);
outIMU.commit();
```

The output containers for events, IMU and trigger also have stream operator support, as

```
auto outTrigger = outputs.getTriggerOutput("trigger").data();
outTrigger << trigger1 << trigger2;
outTrigger.commit();
```

All methods are equivalent in performance.

Frames

OpenCV Frames

To send an *OpenCV* frame to an output, simply use the stream operator on the output.

```
outputs.getFrameOutput("frames") << myFrame << dv::commit;
```

OpenCV frames do not have a notion of timestamps. By default, all frames sent out this way would have a timestamp of 0. To make sure the frames on the output have timestamps, one can stream in the timestamp first.

```
outputs.getFrameOutput("frames") << timestamp << myFrame << dv::commit;
```

Alternatively, one can get the frame at the output, and use that to commit the *OpenCV* matrix manually. This is used, when one wants to set additional parameters such as exposure times.

```
auto outFrame = outputs.getFrameOutput("frames").frame();
outFrame.setTimestamp(timestamp);
outFrame.setTimestampStartOfExposure(timeStartOfExposure);
outFrame.setTimestampEndOfExposure(timeEndOfExposure);
outFrame.setMat(openCVFrame);
outFrame.commit()
```

Non-OpenCV Frames

```
// Get current output frame
auto outFrame = outputs.getFrameOutput("frames").frame();
// Setting the format and size makes the pixel buffer allocate the right amount of␣
↪storage
outFrame.setFormat(dv::FrameFormat::BGR);
outFrame.setSize(640, 480);
// Set the timestamp
outFrame.setTimestamp(timestamp);
// .pixels gives a writable, vector compatible buffer with enough storage.
// You can write the image in there with any method
std::copy(myBuffer, outFrame.pixels());
// As soon as all the data is ready, calling commit sends the data out
outFrame.commit();
```

Non *OpenCV* frames are bit harder to send out. The idea is to generate an empty array on the output, by setting the format as well as the size. Then write the image data in the buffer obtained by `pixels()`. As soon as all the data is ready, call `commit()` to send the frame out.

Note Set the size and format before writing any data to the pixel array. After calling commit, one has to set the size and format again for the next frame, which gives a new array.

Configuration API

Configuration options are settings that are exposed via a network interface from DV runtime. The way configuration options are implemented, the user can change the options dynamically at runtime, either via the DV gui, or with the command line utility.

Definition of configuration options

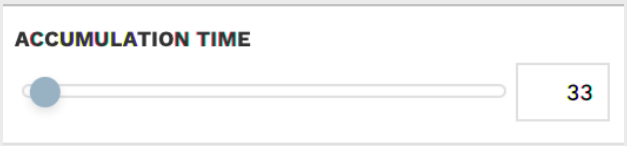
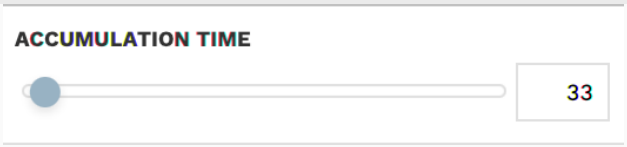
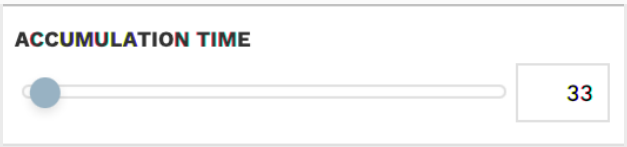
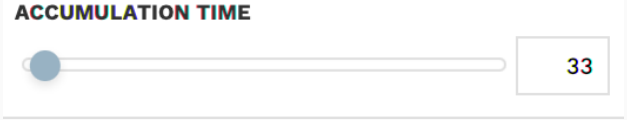
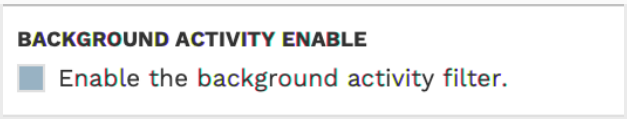
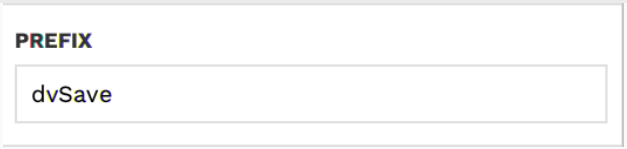
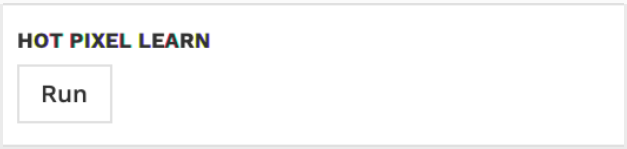
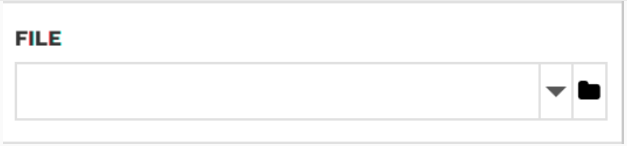
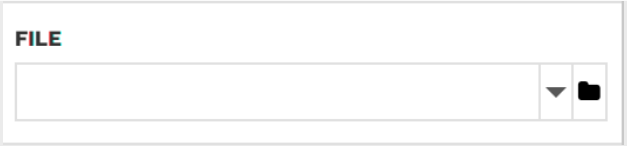
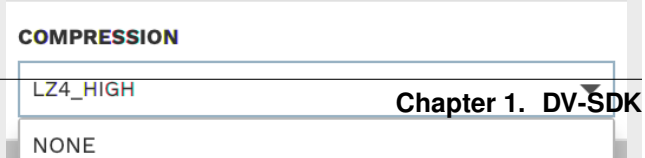
Static vs Runtime definition

Configuration options should be added in the static function `static void initConfigOptions(dv::RuntimeConfig &config)` in your module. This static function gets executed upon adding the module to the project, but *before* it has started. The function is used to populate the configuration before starting the module.

One can add configuration options at runtime, by accessing the modules `config.add` function at runtime. This makes sense, for example, in the constructor.

Selection of config option types

The following configuration option types are available

Name	Value type	Description	GUI representation
int	32bit integer	Accepts a range	
long	64bit integer	Accepts a range	
float	32bit float	Accepts a range	
double	64bit float	Accepts a range	
bool	boolean	Displays a checkbok	
string	string	Displays an edit field	
button	boolean	Displays a button	
fileSave	string	Shows a file save dialog	
fileOpen	string	Shows a file open dialog	
directory	string	Shows a directory chooser dialog	
			

Adding a numerical option (int, long, float, double)

A numerical option can be added as follows

```
config.add(<name>, dv::ConfigOption::intOption(<description>, <defaultValue>,
↪<minValue>, <maxValue>))
config.add(<name>, dv::ConfigOption::longOption(<description>, <defaultValue>,
↪<minValue>, <maxValue>))
config.add(<name>, dv::ConfigOption::floatOption(<description>, <defaultValue>,
↪<minValue>, <maxValue>))
config.add(<name>, dv::ConfigOption::doubleOption(<description>, <defaultValue>,
↪<minValue>, <maxValue>))
```

- `name` The name of the configuration option. Should be a camel cased variable compliant name
- `description` A description of the purpose of the config option
- `defaultValue` The value this config option should get initialized with
- `minValue` The minimum allowed value for this config option. This is optional. If not set, the range defaults to the order of magnitude of the `defaultValue`
- `maxValue` The maximum allowed value for this config option. This is optional. If not set, the range defaults to the order of magnitude of the `defaultValue`

Adding a string option

```
config.add(<name>, dv::ConfigOption::stringOption(<description>, <defaultValue>))
```

- `name` The name of the configuration option. Should be a camel cased variable compliant name
- `defaultValue` The default string to be used in this option

Adding a file option

A file option without a default value can be added as follows:

```
config.add(<name>, dv::ConfigOption::fileOpenOption(<description>, <allowedExtensions>
↪))
config.add(<name>, dv::ConfigOption::fileSaveOption(<description>, <allowedExtensions>
↪))
```

A file option with a default value can be added as follows:

```
config.add(<name>, dv::ConfigOption::fileOpenOption(<description>, <defaultValue>,
↪<allowedExtensions>))
config.add(<name>, dv::ConfigOption::fileSaveOption(<description>, <defaultValue>,
↪<allowedExtensions>))
```

A directory option can be added with:

```
config.add(<name>, dv::ConfigOption::directoryOption(<description>, <defaultValue>))
```

- `name` The name of the configuration option. Should be a camel cased variable compliant name
- `defaultValue` The default file path for the option. This is optional. If not set, the default is an empty string.
- `description` A description of the purpose of the config option

- `allowedExtensions` A comma separated list of the allowed extensions. Extensions should just be the ending, without dot. E.g. `jpg,jpeg,png`

Adding a button option

```
config.add(<name>, dv::ConfigOption::buttonOption(<description>, <buttonLabel>))
```

- `name` The name of the configuration option. Should be a camel cased variable compliant name
- `description` A description of the purpose of the config option
- `buttonLabel` The string that should be displayed on the button

A button option has a boolean value type. By default, the value is false. As soon as a user clicks the button, the button gets disabled and the value set to true to indicate the request. To re-arm the button, reset the value back to false.

Adding a list option

```
config.add(<name>, dv::ConfigOption::listOption(<description>, <defaultChoice>, {
↪<choice1>, <choice2>, ...}, <allowMultipleSelection>))
```

- `name` The name of the configuration option. Should be a camel cased variable compliant name
- `description` A description of the purpose of the config option
- `defaultChoice` The default choice for this option. This can either be a string or the index of the choice in the list of choices. In case of a string, the string has to be present in the list of choices as well.
- `{<choice1>, <choice2>, ...}` A vector of strings, denoting all possible selections for this option

Reading config options

Config values can be read easily with

```
bool value = config.getBool(<name>)
int value = config.getInt(<name>)
long value = config.getLong(<name>)
float value = config.getFloat(<name>)
double value = config.getDouble(<name>)
std::string value = config.getString(<name>)
```

The get functions are not compile time type checked. Make sure to use the correct getter function for your data type to prevent a runtime error. The correct value type for the config options can be obtained from the table above.

Validity of read values

Config option values can be read safely everywhere in the code. Outside changes to the option values are applied in between calls to the *run* and *configUpdate* functions. During the function calls, the value of config options can be seen as constant. (Unless explicitly changed in your code).

Performance considerations

Looking up a config value requies a string key lookup in a hash map. This is a very efficient operation, however, when used in a hot loop in the code, it can add overhead.

To mitigate this problem, the module api provides the opportunity to override a `void configUpdate()` function. This function, compared to the run function, only runs when a config value actually has changed. You can then copy the value of the config variable to an instance member. E.g.

```
void configUpdate() override {
    this->myConfigOption = config.getInt("myConfigOption");
}
```

During the `run` function call, one can then just use the instance member integer, rather than querying the value of the config option over and over again.

Setting config options

Config options can be set everywhere in the code. The option assumes the new value immediately after setting it. To set a config option, use the correct corresponding function

```
config.setBool(<name>, <value>)
config.setInt(<name>, <value>)
config.setLong(<name>, <value>)
config.setFloat(<name>, <value>)
config.setDouble(<name>, <value>)
config.setString(<name>, <value>)
```

Again, it is important to use the correct function for the options value type. Note that by setting config options from code, you are overwriting changes the user has made to the options.

Re-enabling buttons

Buttons have a boolean value type. As long as their config option value is `false`, the button appears armed and can be clicked. Clicking the button sets the value to `true`, to indicate a click request. After performing the required action, set the value back to `false` to re-enable the button.

Example:

```
void configUpdate() override {
    // check if user has clicked the button
    if(config.getBool("myButton") {

        // do complicated action..

        // re-enable button
        config.setBool("myButton", false);
    }
}
```

Logging API

DV offers a logging facility. Log messages get written to the DV log file (typically in `$HOME/.dv-logger.txt`), as well as published to any attached clients, such as the GUI. Logs get automatically tagged with the module name as well as the timestamp.

Accessing the log object

Every DV module class extends from `dv::ModuleBase`. This provides the module with an object called `log`, that can be accessed from everywhere in the class, including constructor and destructor.

Log levels

Every log level has a member in the log object. To log to the four different levels available, use

```
log.debug << "This is a debug message" << dv::logEnd;
log.info << "This is an info message" << dv::logEnd;
log.warning << "This is a warning message" << dv::logEnd;
log.error << "This is an error message" << dv::logEnd;
```

All logging methods described below, can be applied to all log levels.

Logging syntax

Simple logging

The simplest logging syntax can be achieved with

```
log.info("This is an info string");
```

DV logging accepts any type that `std::cout` accepts.

```
log.info(12);
```

Stream logging

Log messages can be streamed into the logging object. Every object that is streamed in appends to the current log message. To send the message out and start a new message, stream in `dv::logEnd`.

```
log.info << "The answer to answer is " << 42 << dv::logEnd;
```

Messages are not complete until `dv::logEnd` is streamed in

```
log.info << 1 << " + " << 1 << " = ";
log.info <<  (1 + 1) << dv::logEnd;
```

Formatted logging

DV supports formatting log messages with C++20's std::format syntax. Use the function `format` for this purpose

```
double mean;
double stddev;
int nSamples;

log.info.format("mean: {:.2f}, stddev: {:.2f}, samples: {}.", mean, stddev, nSamples);
```

Consult the fmt library[9] documentation for the full format syntax.

[9] https://fmt.dev/latest/syntax.html

1.2.2 Create a Custom Datatype

The data transferred between DV modules is implemented as FlatBuffers. This tutorial is to show how to create a new datatype based on *FlatBuffers*[10] which can then be used to transmit data between different modules.

Prerequisites

First, the DV-runtime needs to be *installed* on your machine. Additionally, the following tools need to be available:

- cmake
- clang-format
- make
- bash

How-To

Getting the Schema Compiler

The precompiled binaries for the schema compiler `flatc` are located in the `dv-runtime` repository. The precompiled binaries are located in `dv-runtime/flatbuffers`. For more information on the schema compiler `flatc`, check out the official documentation: *https://google.github.io/flatbuffers/flatbuffers_guide_using_schema_compiler.html*

Writing the Schema

The full syntax for the schema file is documented in the following guides:

- https://google.github.io/flatbuffers/flatbuffers_guide_writing_schema.html
- https://google.github.io/flatbuffers/flatbuffers_guide_writing_schema.html

An example for a schema file is shown below:

```
file_identifier "PLOT";

struct Vector3D {
        x: int32;
        y: int32;
        z: int32;
}

table PlotPoint {
        /// Timestamp (µs).
        timestamp: int64;
        /// Plot X coordinate.
        coord: Vector3D (native_inline);
}

table PlotPacket {
        elements: [PlotPoint] (native_inline);
}

root_type PlotPacket;
```

[10] https://google.github.io/flatbuffers/index.html

This schema creates a FlatBuffers `vector` which can be used as an input or output throughout the DV toolchain using the *dv-sdk*. The `vector` is of type `PlotPacket` and contains elements of type `PlotPoint`. Each `PlotPoint` contains a `timestamp` and a `Vector3D`.

Remarks Regarding the Schema

- In order for the schema to be compatible with the DV toolchain it is necessary for it to contain a variable called `timestamp` of type `int64`.
- The attribute `native_inline` is used whenever the class created by `flatc` should contain an object of the struct for which the attribute is dedicated. If the attribute is not present, the field will be implemented as a pointer.
- `tables` may be extended in future versions of the FlatBuffer that is created, `structs` may not.
- The `file_identifier` needs to be a `string` of four characters

Compiling the Schema

In order to compile the schema for use with the *dv-sdk* the following command should be used:

```
<path_to_flatc> --cpp \
                --scoped-enums \
                --gen-object-api \
                --gen-compare \
                --gen-name-strings \
                --cpp-ptr-type "std::unique_ptr" \
                --cpp-str-type "std::string" \
                --cpp-str-flex-ctor \
                --cpp-vec-type "std::vector" \
                --reflect-types \
                --reflect-names <schema_file>.fbs

mv `basename <schema_file>`_generated.h <schema_file>_base.hpp
clang-format -i <schema_file>_base.hpp
```

For simplicity, this has been implemented in a bash script in the dv-runtime repository: `flatbuffers/dv-sdk-flatc.sh` (see https://gitlab.com/inivation/dv/dv-runtime/-/tree/master/flatbuffers)

You can also just clone the dv-runtime repository instead and execute the file directly.

Using the FlatBuffer Object

Now that the `<schema_file>_base.hpp` object is created, it can be used with the *dv-sdk*.

The following examples show how the FlatBuffer object can be used in a Module.

Registering your Custom Type with the Module

Add the following static function to your module class.

```
static void initTypes(std::vector<dv::Types::Type> &types) {
        types.push_back(dv::Types::makeTypeDefinition<PlotPacket, PlotPoint>("A point␣
↪to be plotted"));
}
```

Adding the FlatBuffer Object as an Output of a Module

```
static void initOutputs(dv::OutputDefinitionList &out) {
  out.addOutput("stats", PlotPacket::TableType::identifier);
}
```

The value of `PlotPacket::TableType::identifier` is the four character `file_identifier` (in this case `"PLOT"`).

Adding an Object to the FlatBuffer Output Vector

```
auto stats  = outputs.getVectorOutput<PlotPacket, PlotPoint>("stats").data();

Vector3D vec(1, 2, 3);
PlotPoint point(<timestamp>, vec);

stats.commit();
```

Adding the FlatBuffer Object as an Input of a Module

```
static void initInputs(dv::InputDefinitionList &in) {
  in.addEventInput("stats", PlotPacket::TableType::identifier);
}
```

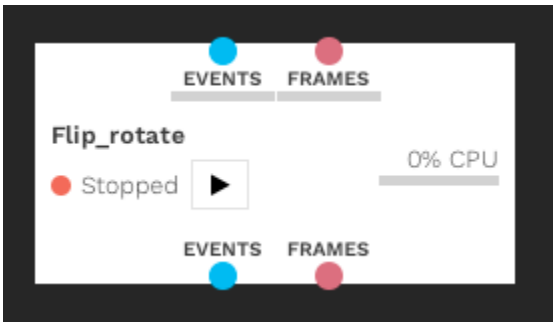
The value of `PlotPacket::TableType::identifier` is the four character `file_identifier` (in this case `"PLOT"`).

Getting an Object from the FlatBuffer Input Vector

```
void run() {
  auto stats  = outputs.getVectorInput<PlotPacket, PlotPoint>("stats").data();

  for(const auto &point: stats) {
    // do whatever
  }
}
```

A DV module is a piece of code that applies some sort of operation to data, usually having multiple data inputs and outputs.



Some modules only have outputs, such as the camera or file playback modules, as they only produce new data for the framework to consume. Some modules only have inputs, such as the network or file recorder modules, as they send data

outside the framework. When you want to build an application for DV, you most likely want to build a module that takes on some camera data, and outputs some measurements or visualization outputs.

Every module is essentially a CMake project, that most modern IDEs understand.

In this documentation we show you how to:

- *Create a DV Module*
- *Create a Custom Datatype* to use as input/output for modules.

1.2.3 Module Files Location

Modules are shared libraries. They are files with the extension *.so* (Linux), *.dylib* (macOS), or *.dll* (Windows). Modules library files are in:

- `/usr/share/dv/modules` (Linux)
- `/usr/local/share/dv/modules` or `/opt/homebrew/share/dv/modules` (macOS)
- `C:\Program Files\DV\runtime\dv_modules` (Windows)

For a library to be discoverable by DV, it has to be in that directory.

`dv-sdk` is the library we provide to extend the provided functionalities of the DV[11] software or integrate other algorithms into it. It is used to create new modules that would implement new processing features or integrate existing algorithms.

In this documentation we explain how to:

- *Install the dv-sdk library*
- *Understand and Create DV Modules*

[11] https://docs.inivation.com/software/dv/index.html

CHAPTER

TWO

DV-RUNTIME

2.1 Installation

Installation of `dv-runtime` is only provided on *Mac* and *Linux* systems.

2.1.1 MacOS

The easiest way to install `dv-runtime` is via Homebrew[12]. We provide a Homebrew tap[13] for macOS. Install it with:

```
brew tap inivation/inivation
brew install dv-runtime
```

2.1.2 Linux

Ubuntu Linux

We provide a PPA repository[14] for Focal (20.04 LTS), Jammy (22.04 LTS) and Noble (24.04 LTS) on the x86_64, arm64 and armhf architectures.

```
sudo add-apt-repository ppa:ubuntu-toolchain-r/test
sudo add-apt-repository ppa:inivation-ppa/inivation
sudo apt-get update
sudo apt-get install dv-runtime
```

Fedora Linux

We provide a COPR repository[15] for Fedora 34, 35, 36 and rawhide on the x86_64, arm64 and armhf architectures.

```
sudo dnf copr enable inivation/inivation
sudo dnf install dv-runtime
```

Arch Linux

You can find `dv-runtime` as a package in the AUR repository[16].

[12] https://brew.sh/
[13] https://gitlab.com/inivation/homebrew-inivation/
[14] https://launchpad.net/~inivation-ppa/+archive/ubuntu/inivation
[15] https://copr.fedorainfracloud.org/coprs/inivation/inivation/
[16] https://aur.archlinux.org/packages/dv-runtime

Gentoo Linux

You can find `dv-runtime` as a valid Gentoo ebuild repository available here[17] over Git. The package to install is 'dev-util/dv-runtime'.

Embedded Devices

On most embedded devices, you can follow the *Ubuntu Linux instructions*.

To start a `dv-runtime` instance, you might have to set the following environment variable: `UNW_ARM_UNWIND_METHOD=4`.

Nvidia Jetson

The Nvidia Jetson embedded boards are supported via *our Ubuntu packages*. The Linux for Tegra (L4T)[18] distribution is based on Ubuntu, so the same installation instructions apply. One thing to note: Nvidia provides an incompatible version of OpenCV 4.x, so you'll first have to uninstall their version that's pre-installed and then install the one from our PPA repository.

Raspberry Pi

Raspberry Pi is supported on the Ubuntu operating system. Ubuntu is an officially supported OS for Raspberry Pi[19]. Please use *our Ubuntu packages*.

2.2 Usage

2.2.1 dv-runtime

`dv-runtime` is an executable that launches one instance of our runtime system.

Executable options:

Option	Argument	Purpose
`-h` or `--help`	None	Print help text (this table)
`-v` or `--version`	None	Print runtime version
`-i` or `--ipaddress`	IP address (default 127.0.0.1)	Use this IP address for config-server to listen on
`-p` or `--port`	Port number (default 4040)	Use this port number for config-server to listen on
`-l` or `--log`	Path to log file (default `~/.dv-runtime.log`)	Use this file to log messages from the runtime
`--systemservice`	`0` or `1` (default 0)	Enable system service mode
`--console`	`0` or `1` (default 1)	Log messages to the console
`-c` or `--config`	Path to xml file	Use this XML as initial configuration

Note1: Multiple instances of the runtime can be started on the same machine, but they need to be on different IP/Port combinations.

Note2: By default, `dv-runtime` will only access *modules* that are located in the *default folders*. To access modules located in different paths, one can use the following environment variable:

[17] https://gitlab.com/inivation/gentoo-inivation/
[18] https://developer.nvidia.com/embedded/jetson-linux
[19] https://ubuntu.com/download/raspberry-pi

```
DV_MODULES_PATH=/path/to/folder/ # containing your module(s) (module.so)
```

2.2.2 dv-control

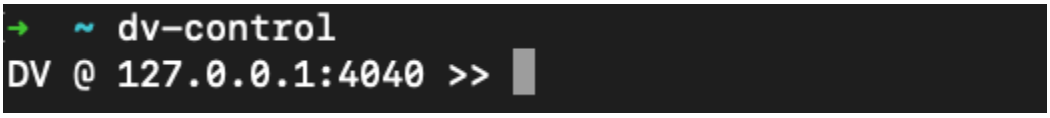
`dv-control` is an executable that starts a Command Line Interface used to control and modify live running instances of the runtime.

Executable options:

Option	Argument	Purpose
`-h` or `--help`	None	Print help text (this table)
`-i` or `--ipad-dress`	IP address (default 127.0.0.1)	Use this IP address to connect to a runtime on this address
`-p` or `--port`	Port number (default 4040)	Use this port number to connect to a runtime on this port
`--tls`	Path to a file in the PEM format (default "")	Enable TLS for connection using the specified CA file (no argument: default CA for verification)
`--tlscert`	Path to a file in the PEM format (default "")	Uses the specified TLS key file for client authentication
`-s` or `--script`	CLI command (more details *here*)	Runs the provided command as if typed in the CLI, then exits

Command Line Interface

Once started, `dv-control` becomes a Command Line Interface.



The different available commands and their usages are the following:

Command	Arguments	Action
`node_exists`	`node`	Returns `true` if the provided `node` exists, else `false`.
`attr_exists`	`node key`	Returns `true` if, at the provided `node`, the provided `key` exists, else `false`.
`get_children`	`node`	Returns the names of all children from the provided `node`.
`get_type`	`node key`	Returns the type of the provided `key` from the provided `node`.
`help/get_descripti`	`node key`	Returns the description of the provided `key` from the provided `node`.
`get`	`node key`	Returns the value of the provided `key` from the provided `node`.
`put/set`	`node key value`	Sets the `value` of the provided `key` from the provided `node`.
`add_module`	`name library`	Adds a module from the provided `library` with the provided `name` to the structure.
`remove_module`	`name`	Removes the module with the provided `name` from the structure.
`dump_tree`	*None*	Send a full dump of the current configuration tree.
`get_client_id`	*None*	Get the ID of this client.
`quit/exit`	*None*	Exits the CLI.

Note1:

- `node` arguments are in the form of paths. Ex: `/mainloop/accumulator`.

- `key` arguments are generally attributes (written in *camelCase*) from nodes. Ex: `decayParam`.
- `library` arguments are a name of a module library. Ex: `dv_accumulator`.

Note2: The CLI has autocomplete, which means most of the commands, nodes and keys can be auto-completed using the `tab` key.

Note3: To connect the outputs of a module to the inputs of another one, one should use the following:

```
set /mainloop/receiver_module_name/inputs/input_name from sender_module_name[output_
↪name]
```

2.2.3 Config Server Communication

The configuration server is accessible through a TCP connection, with optional TLS v1.2 encryption and authentication support. Clients connect to the server and can then send commands and receive data back.

All communication to and from the configuration server is done using Flatbuffers[20], a schema-based serialization protocol originally developed by Google. The following schema is used:

```
namespace dv;

enum ConfigAction : int8 {
        CFG_ERROR          = 0,
        NODE_EXISTS        = 1,
        ATTR_EXISTS        = 2,
        GET_CHILDREN       = 3,
        GET_ATTRIBUTES     = 4,
        GET_TYPE           = 5,
        GET_RANGES         = 6,
        GET_FLAGS          = 7,
        GET_DESCRIPTION    = 8,
        GET                = 9,
        PUT                = 10,
        ADD_MODULE         = 11,
        REMOVE_MODULE      = 12,
        ADD_PUSH_CLIENT    = 13,
        REMOVE_PUSH_CLIENT = 14,
        PUSH_MESSAGE_NODE  = 15,
        PUSH_MESSAGE_ATTR  = 16,
        DUMP_TREE          = 17,
        DUMP_TREE_NODE     = 18,
        DUMP_TREE_ATTR     = 19,
        GET_CLIENT_ID      = 20
}

enum ConfigType : int8 {
        UNKNOWN = -1,
        BOOL    = 0,
        INT     = 1,
        LONG    = 2,
        FLOAT   = 3,
        DOUBLE  = 4,
        STRING  = 5
```

(continues on next page)

[20] https://google.github.io/flatbuffers/index.html

(continued from previous page)

```
}

enum ConfigNodeEvents : int8 {
        NODE_ADDED   = 0,
        NODE_REMOVED = 1
}

enum ConfigAttributeEvents : int8 {
        ATTRIBUTE_ADDED           = 0,
        ATTRIBUTE_MODIFIED        = 1,
        ATTRIBUTE_REMOVED         = 2,
        ATTRIBUTE_MODIFIED_CREATE = 3
}

table ConfigActionData {
        action: ConfigAction = CFG_ERROR;
        nodeEvents: ConfigNodeEvents; // Node related.
        attrEvents: ConfigAttributeEvents; // Attribute related.
        id: uint64;
        node: string;
        key: string;
        type: ConfigType = UNKNOWN;
        value: string;
        // On attribute create() only.
        ranges: string;
        flags: int32;
        description: string;
}

root_type ConfigActionData;
```

The buffer content is defined in ConfigActionData. The 'action' member is the only one that's required to be always set. The ConfigType, ConfigNodeEvents and ConfigAttributeEvents enums reflect the values found in the main dv::Config namespace, adapted for the Flatbuffer schema here. The ConfigAction enum defines all possible action that the configuration server can execute or respond with. The following table goes into detail of what data the client needs to send or what data he can expect back, based on the ConfigAction sent. On success, the response message will have the same action value, with possibly other additional data members set (see 'Server to Client Response'); on failure, CFG_ERROR is returned with a string explaining the problem.

ConfigAction	Client to Server Message	Server to Client Response
CFG_ERROR	N/A (only from server due to errors).	Sent whenever an error occours. Error message is contained in member 'value'.
NODE_EXISTS	Check if a given 'node' exists.	Member 'value' either "true" or "false".
ATTR_EXISTS	Check if a given 'key' (with optional 'type') exists for the given 'node'.	Member 'value' either "true" or "false".
GET_CHILDREN	Get the names of all children of a 'node'.	Member 'value' contains all child node names, separated by '|'.
GET_ATTRIBUTE	Get the names of all attribute keys of a 'node'.	Member 'value' contains all attribute key names, separated by '|'.
GET (_TYPE,_ SCRIPTION)	Get all information for the 'key' (with optional 'type') of 'node'.	All members are filled with information on key.
PUT	Update the value for the 'key' (with optional 'type') of 'node', using the string member 'value'.	No additional return value.
ADD_MODULE	Create a new module with name of 'node' and library of 'key'.	No additional return value.
REMOVE_MULE	Remove the module with name of 'node'.	No additional return value.
ADD_PUS	Add this client to the list of clients that shall receive push notifications on configuration changes. No input values.	Client ID formatted as string in 'value'.
REMOVE_PU	Remove this client from the list of clients that shall receive push notifications on configuration changes. No input values.	No additional return value.
PUSH_MESSAGE_NO	N/A (only from server after ADD_PUSH_CLIEN	Detected change on a node. 'id' contains the ID of the change originator (0=runtime, 1-N=connected client), 'nodeEvents' the kind of change, 'node' the path of the node that changed.
PUSH_MESSAGE_AT	N/A (only from server after ADD_PUSH_CLIEN	Detected change on an attribute. 'id' contains the ID of the change originator (0=runtime, 1-N=connected client), 'attrEvents' the kind of change, 'node' the path of the node where the attribute resides, 'key' the name of the attribute key, 'type' the type of the attribute, 'value' the new value. On attribute addition (attrEvents=ATTRIBUTE_ADDED), the following members are also set: 'flags' to the current flags value, 'ranges' to the min/max ranges and 'description' to the current description (help text).

Our runtime system can be used in multiple ways:

- Via our DV GUI Software. All explanations are available on our main documentation page[21].
- Via terminal. This is the focus of this documentation.
  - *dv-runtime* to launch instances of the runtime system
  - *dv-control* to control and modify live the running instances.
- Via other custom applications, using our provided *config server communication* (ADVANCED).

`dv-runtime` is our provided Runtime System[22]. It is the environment that will host the modules and take care of the threads, the memory allocation and data exchange between the modules. It is made to be used via DV[23], but it can also be used on its own.

In this documentation we explain how to:

- *Install dv-runtime*
- *Use dv-runtime*

2.3 API

Full API documentation, automatically generated from doxygen comments.

template<*dv*::*Config*::*AttributeType*>

struct **_ConfigAttributes**

#include </builds/inivation/dv/dv-runtime/include/dv-sdk/config.hpp> Maps the selected config type to a struct with additional config params that have to be set for the selected config type. Defaults to an empty struct.

template<>

struct **_ConfigAttributes**<*dv*::*Config*::*AttributeType*::*BOOL*>

Public Functions

inline **_ConfigAttributes**(*dv*::*_BooleanAttributeType* attributeType_, const std::string &buttonLabel_ = std::string())

Public Members

dv::*_BooleanAttributeType* **attributeType**

const std::string **buttonLabel**

template<>

struct **_ConfigAttributes**<*dv*::*Config*::*AttributeType*::*DOUBLE*>

[21] https://docs.inivation.com/software/dv/gui/index.html
[22] https://en.wikipedia.org/wiki/Runtime_system
[23] https://docs.inivation.com/software/dv/index.html

Public Functions

inline **_ConfigAttributes**(double minValue, double maxValue)

Public Members

dv::*Config*::*AttributeRanges*<*dv*::*Config*::*AttributeType*::*DOUBLE*> **range**

std::string **unit**

template<>

struct **_ConfigAttributes**<*dv*::*Config*::*AttributeType*::*FLOAT*>

Public Functions

inline **_ConfigAttributes**(float minValue, float maxValue)

Public Members

dv::*Config*::*AttributeRanges*<*dv*::*Config*::*AttributeType*::*FLOAT*> **range**

std::string **unit**

template<>

struct **_ConfigAttributes**<*dv*::*Config*::*AttributeType*::*INT*>

Public Functions

inline **_ConfigAttributes**(int32_t minValue, int32_t maxValue)

Public Members

dv::*Config*::*AttributeRanges*<*dv*::*Config*::*AttributeType*::*INT*> **range**

std::string **unit**

template<>

struct **_ConfigAttributes**<*dv*::*Config*::*AttributeType*::*LONG*>

Public Functions

inline **_ConfigAttributes**(int64_t minValue, int64_t maxValue)

Public Members

dv::*Config*::*AttributeRanges*<*dv*::*Config*::*AttributeType*::*LONG*> **range**

std::string **unit**

template<>

struct **_ConfigAttributes**<*dv*::*Config*::*AttributeType*::*STRING*>

Public Functions

inline **_ConfigAttributes**(int32_t minLength, int32_t maxLength, *_StringAttributeType* t)

Public Members

dv::*Config*::*AttributeRanges*<*dv*::*Config*::*AttributeType*::*STRING*> **length**

_StringAttributeType **type**

std::vector<std::string> **listOptions**

bool **listAllowMultipleSelections**

FileDialogMode **fileMode**

std::string **fileAllowedExtensions**

template<*dv*::*Config*::*AttributeType* **T**>

class **_ConfigOption**

#include </builds/inivation/dv/dv-runtime/include/dv-sdk/config.hpp> Templated implementation class of a *ConfigOption*. Stores extra attributes according to the selected config type.

Public Functions

inline **_ConfigOption**(const std::string &description_, *_AttrType* initValue_, const *_ConfigAttributes*<*T*> &attributes_, *dv*::*Config*::*AttributeFlags* flags_, bool updateReadOnly_ = false)

Public Members

const std::string **description**

const *_AttrType* **initValue**

const *_ConfigAttributes*<*T*> **attributes**

const *dv*::*Config*::*AttributeFlags* **flags**

const bool **updateReadOnly**

_AttrType **currentValue**

Private Types

using **_AttrType** = typename *dv*::*Config*::*AttributeTypeGenerator*<*T*>::type

template<typename **T**>

class **_InputDataWrapperCommon**

#include </builds/inivation/dv/dv-runtime/include/dv-sdk/data/wrappers.hpp> Common parts of the implementation of an input wrapper. All specific input data wrappers for data types as well as the generic input data wrapper inherit from this.

Template Parameters
T – The type of the input data

Subclassed by *dv::_InputVectorDataWrapperCommon< dv::BoundingBoxPacket, dv::BoundingBox >*, *dv::_InputVectorDataWrapperCommon< dv::EventPacket, dv::Event >*, *dv::_InputVectorDataWrapperCommon< dv::IMUPacket, dv::IMU >*, *dv::_InputVectorDataWrapperCommon< dv::TriggerPacket, dv::Trigger >*, *dv::InputDataWrapper< T >*, *dv::_InputVectorDataWrapperCommon< T, U >*

Public Functions

inline explicit **operator bool**() const noexcept

inline std::shared_ptr<const *T*> **getBasePointer**() const noexcept

inline **operator std::shared_ptr<const *T*>**() const noexcept

Protected Functions

inline **_InputDataWrapperCommon**(std::shared_ptr<const *T*> p)

Protected Attributes

std::shared_ptr<const *T*> **ptr**

template<typename **T**, typename **U**>

class **_InputVectorDataWrapperCommon** : public *dv*::*_InputDataWrapperCommon*<*T*>, public *dv*::*vectorConstProxy*<*U*>

Subclassed by *dv::InputVectorDataWrapper< T, U >*

Protected Functions

inline **_InputVectorDataWrapperCommon**(std::shared_ptr<const *T*> p)

template<typename **T**>

class **_OutputDataWrapperCommon**

#include </builds/inivation/dv/dv-runtime/include/dv-sdk/data/wrappers.hpp> Common parts of the implementation of an output wrapper. All specific output data wrappers for data types as well as the generic output data wrapper inherit from this.

Template Parameters
T – The type of the output data

Subclassed by *dv::_OutputVectorDataWrapperCommon< dv::BoundingBoxPacket, dv::BoundingBox >*, *dv::_OutputVectorDataWrapperCommon< dv::EventPacket, dv::Event >*, *dv::_OutputVectorDataWrapperCommon< dv::IMUPacket, dv::IMU >*, *dv::_OutputVectorDataWrapperCommon< dv::TriggerPacket, dv::Trigger >*, *dv::OutputDataWrapper< T >*, *dv::_OutputVectorDataWrapperCommon< T, U >*

Public Functions

inline *_OutputDataWrapperCommon* &**operator=**(const *_InputDataWrapperCommon*<*T*> &rhs)

inline explicit **operator bool**() const noexcept

inline *T* ***getBasePointer**() noexcept

inline const *T* ***getBasePointer**() const noexcept

inline void **commit**()

inline *_OutputDataWrapperCommon* &**operator<<**(*commitType*)

Protected Functions

inline **_OutputDataWrapperCommon**(*T* *p, *dvModuleData* m, const std::string &n)

Protected Attributes

T ***ptr**

dvModuleData **moduleData**

std::string **name**

template<typename **T**, typename **U**>

class **_OutputVectorDataWrapperCommon** : public *dv*::*_OutputDataWrapperCommon*<*T*>, public *dv*::*vectorProxy*<*U*>

Subclassed by *dv::OutputVectorDataWrapper< T, U >*

Public Functions

inline void **commit**()

inline *_OutputVectorDataWrapperCommon* &**operator<<**(*commitType*)

inline *_OutputVectorDataWrapperCommon* &**operator<<**(const *U* &rhs)

template<typename **K**>
inline *_OutputVectorDataWrapperCommon* &**operator<<**(const *K* &container)

Appends all events the iterable container to the output buffer

Parameters
container – The store with the events to be copied into the output buffer

Returns
A reference to the output buffer

Protected Functions

inline **_OutputVectorDataWrapperCommon**(*T* *p, *dvModuleData* m, const std::string &n)

class **_RateLimiter**

Public Functions

inline **_RateLimiter**(int32_t messageRate, int32_t perMilliseconds)

inline bool **pass**()

Private Members

float **rate**

float **allowanceLimit**

float **allowance**

std::chrono::time_point<std::chrono::steady_clock> **last_check**

template<typename **T**>

class **_RuntimeInputCommon**

#include </builds/inivation/dv/dv-runtime/include/dv-sdk/data/wrappers.hpp> Base class for a runtime input definition. There are template-specialized subclasses of this, providing convenience function interfaces for the most common, known types. There is also a generic, templated subclass `RuntimeInput` which does not add any more convenience functions over this common subclass, and can be used for the generic case

Subclassed by *dv::_RuntimeVectorInputCommon< dv::BoundingBoxPacket, dv::BoundingBox >*, *dv::_RuntimeVectorInputCommon< dv::EventPacket, dv::Event >*, *dv::RuntimeInput< T >*, *dv::_RuntimeVectorInputCommon< T, U >*

Public Functions

inline const *InputDataWrapper*<*T*> **data**() const

Get data from an input

Returns
An input wrapper of the desired type, allowing data access

inline bool **isConnected**() const

Returns true, if this optional input is actually connected to an output of another module

Returns
true, if this input is connected

inline const *dv*::*Config*::*Node* **infoNode**() const

Returns an info node about the specified input. Can be used to determine dimensions of an input/output

Returns
A node that contains the specified inputs information, such as “sizeX” or “sizeY”

inline const std::string **getOriginDescription**() const

Returns the description of the origin of the data

Returns
the description of the origin of the data

Protected Functions

inline std::shared_ptr<const *T*> **getUnwrapped**() const

Fetches available data at the input and returns a shared_ptr to it. Also casts the shared_ptr to this particular input type.

Returns
A shared_ptr of the input data type to the latest received data

inline **_RuntimeInputCommon**(const std::string &name, *dvModuleData* moduleData)

This constructor is called by the child classes in their initialization

Parameters

- **name** – The name of this input
- **moduleData** – Pointer to the dv moduleData struct

Private Members

std::string **name_**

dvModuleData **moduleData_**

template<typename **T**>

class **_RuntimeOutputCommon**

#include </builds/inivation/dv/dv-runtime/include/dv-sdk/data/wrappers.hpp> Base class for a runtime output. This class acts as the base for various template-specialized sub classes which provide convenience functions for outputting data in their respective data types. There is a templated generic subclass `RuntimeOutput<T>` that can be used for the generic case

Template Parameters
T – The flatbuffers type of the output data

Subclassed by *dv::_RuntimeVectorOutputCommon< dv::BoundingBoxPacket, dv::BoundingBox >*, *dv::_RuntimeVectorOutputCommon< dv::EventPacket, dv::Event >*, *dv::RuntimeOutput< T >*, *dv::_RuntimeVectorOutputCommon< T, U >*

Public Functions

inline void **setup**(const std::string &originDescription)

Sets up the output. Has to be called in the constructor of the module.

Parameters
originDescription – A description of the original creator of the data

template<typename **U**>
inline void **setup**(const *RuntimeInput*<*U*> &input)

Sets this output up with the same parameters as the supplied input.

Parameters
input – An input to copy the information from

template<typename **U**, typename **TT**>
inline void **setup**(const *RuntimeVectorInput*<*U*, *TT*> &input)

Sets this event output up with the same parameters as the supplied vector input.

Parameters
input – A vector input to copy the information from

inline *OutputDataWrapper*<*T*> **data**()

Returns a writeable output wrapper for the given type of this output. Allocates new output memory if necessary. The output can be committed by calling commit on the returned object.

Returns
A wrapper to allocated output memory to write to

inline void **operator<<**(*commitType*)

Convenience shortcut to commit the data on this output directly

inline *dv*::*Config*::*Node* **infoNode**()

Returns an info node about the specified output, can be used to set output information.

Returns
A node that can contain output information, such as "sizeX" or "sizeY"

inline const *dv*::*Config*::*Node* **infoNode**() const

Returns an info node about the specified output, can be used to set output information.

Returns
A node that can contain output information, such as "sizeX" or "sizeY"

inline const std::string **getOriginDescription**() const

Returns the description of the origin of the data

Returns
the description of the origin of the data

Protected Functions

inline *T* ***allocateUnwrapped**()

Allocates a new instance of the datatype of this output and returns a raw pointer to the allocated memory. If there was memory allocated before (This function has been called before) but the output never has been commited, a raw pointer to the previously allocated memory gets returned.

Returns
A raw pointer to the allocated memory

inline void **createSourceAttribute**(const std::string &originDescription)

Creates the output information attribute in the config tree. The source attribute is a string containing information about the original generator of the data

Parameters
originDescription – a string containing information about the original generator of the data

inline void **createSizeAttributes**(int sizeX, int sizeY)

Adds size information attributes to the output info node

Parameters

- **sizeX** – The width dimension of the output
- **sizeY** – The height dimension of the output

inline **_RuntimeOutputCommon**(const std::string &name, *dvModuleData* moduleData)

This constructor is called by the subclasses constructors

Parameters

- **name** – The configuration name of the module this output belongs to
- **moduleData** – A pointer to the dv moduleData struct

Protected Attributes

std::string **name_**

dvModuleData **moduleData_**

template<typename **T**, typename **U**>

class **_RuntimeVectorInputCommon** : public *dv*::*_RuntimeInputCommon*<*T*>

Subclassed by *dv::RuntimeVectorInput< T, U >*

Public Functions

inline const *InputVectorDataWrapper*<*T*, *U*> **data**() const

Get data from a vector input

Returns
An input wrapper of the desired type, allowing data access

Protected Functions

inline **_RuntimeVectorInputCommon**(const std::string &name, *dvModuleData* moduleData)

template<typename **T**, typename **U**>

class **_RuntimeVectorOutputCommon** : public *dv*::*_RuntimeOutputCommon*<*T*>

Subclassed by *dv::RuntimeVectorOutput< T, U >*

Public Functions

inline *OutputVectorDataWrapper*<*T*, *U*> **data**()

Returns a writeable output wrapper for the given type of this output. Allocates new output memory if necessary. The output can be committed by calling commit on the returned object.

Returns
A wrapper to allocated output memory to write to

template<typename **K**>
inline *_RuntimeVectorOutputCommon* &**operator<<**(const *K* &rhs)

Convenience shortcut to commit the data on this output directly This gets resolved to one of the various << functions of the data wrapper.

inline void **operator<<**(*commitType*)

Convenience shortcut to commit the data on this output directly

Protected Functions

inline **_RuntimeVectorOutputCommon**(const std::string &name, *dvModuleData* moduleData)

template<*AttributeType* **T**>

struct **AttributeRangeGenerator**

Public Types

using **rangeType** = typename *AttributeTypeGenerator*<*T*>::type

template<>

struct **AttributeRangeGenerator**<*AttributeType*::*BOOL*>

Public Types

using **rangeType** = int32_t

template<>

struct **AttributeRangeGenerator**<*AttributeType*::*STRING*>

Public Types

using **rangeType** = int32_t

template<*AttributeType* **T**>

struct **AttributeRanges**

Public Types

using **rangeType** = typename *AttributeRangeGenerator*<*T*>::rangeType

Public Functions

inline **AttributeRanges**(*rangeType* minVal, *rangeType* maxVal)

inline **AttributeRanges**(const *dvConfigAttributeRanges* ranges)

inline *dvConfigAttributeRanges* **getCStruct**() const

Public Members

rangeType **min**

rangeType **max**

template<>

struct **AttributeRanges**<*AttributeType*::*BOOL*>

Public Types

using **rangeType** = typename *AttributeRangeGenerator*<*AttributeType*::*BOOL*>::*rangeType*

Public Functions

inline **AttributeRanges**()

inline **AttributeRanges**(const *dvConfigAttributeRanges* ranges)

inline *dvConfigAttributeRanges* **getCStruct**() const

Public Members

rangeType **min**

rangeType **max**

template<typename **T**>

struct **AttributeTypeConverter**

template<>

struct **AttributeTypeConverter**<bool>

Public Static Attributes

static constexpr *AttributeType* **type** = *AttributeType*::*BOOL*

template<>

struct **AttributeTypeConverter**<double>

Public Static Attributes

static constexpr *AttributeType* **type** = *AttributeType*::*DOUBLE*

template<>

struct **AttributeTypeConverter**<float>

Public Static Attributes

static constexpr *AttributeType* **type** = *AttributeType*::*FLOAT*

template<>

struct **AttributeTypeConverter**<int32_t>

Public Static Attributes

static constexpr *AttributeType* **type** = *AttributeType*::*INT*

template<>

struct **AttributeTypeConverter**<int64_t>

Public Static Attributes

static constexpr *AttributeType* **type** = *AttributeType*::*LONG*

template<>

struct **AttributeTypeConverter**<std::string>

Public Static Attributes

static constexpr *AttributeType* **type** = *AttributeType*::*STRING*

template<*AttributeType* **T**>

struct **AttributeTypeGenerator**

template<>

struct **AttributeTypeGenerator**<*AttributeType*::*BOOL*>

Public Types

using **type** = bool

Public Static Attributes

static enum *dvConfigAttributeType* **underlyingType** = *DVCFG_TYPE_BOOL*

template<>

struct **AttributeTypeGenerator**<*AttributeType*::*DOUBLE*>

Public Types

using **type** = double

Public Static Attributes

static enum *dvConfigAttributeType* **underlyingType** = *DVCFG_TYPE_DOUBLE*

template<>

struct **AttributeTypeGenerator**<*AttributeType*::*FLOAT*>

Public Types

using **type** = float

Public Static Attributes

static enum *dvConfigAttributeType* **underlyingType** = *DVCFG_TYPE_FLOAT*

template<>

struct **AttributeTypeGenerator**<*AttributeType*::*INT*>

Public Types

using **type** = int32_t

Public Static Attributes

static enum *dvConfigAttributeType* **underlyingType** = *DVCFG_TYPE_INT*

template<>

struct **AttributeTypeGenerator**<*AttributeType*::*LONG*>

Public Types

using **type** = int64_t

Public Static Attributes

static enum *dvConfigAttributeType* **underlyingType** = *DVCFG_TYPE_LONG*

template<>

struct **AttributeTypeGenerator**<*AttributeType*::*STRING*>

Public Types

using **type** = std::string

Public Static Attributes

static enum *dvConfigAttributeType* **underlyingType** = *DVCFG_TYPE_STRING*

template<*AttributeType* **T**>

struct **AttributeValue**

Public Types

using **valueType** = typename *AttributeTypeGenerator*<*T*>::type

Public Functions

inline **AttributeValue**(*valueType* v)

inline **AttributeValue**(const *dvConfigAttributeValue* v)

inline *dvConfigAttributeValue* **getCUnion**() const

Public Members

valueType **value**

template<>

struct **AttributeValue**<*AttributeType*::*STRING*>

Public Functions

inline **AttributeValue**(const std::string_view v)

inline **AttributeValue**(const *dvConfigAttributeValue* v)

inline *dvConfigAttributeValue* **getCUnion**() const

Public Members

AttributeTypeGenerator<*AttributeType*::*STRING*>::*type* **value**

struct **commitType**

class **ConfigOption**

#include </builds/inivation/dv/dv-runtime/include/dv-sdk/config.hpp> Non template class for a config option. Has a type independent unique pointer that points to the actual (templated) config object. Private constructor. Should only be used with static factory function.

Public Functions

inline void **setRateLimit**(int32_t messageRate, int32_t perMilliseconds)

inline *dv*::*Config*::*AttributeType* **getType**() const

Returns the type of this `ConfigOption`.

Returns
the configuration's type.

template<*dv*::*Config*::*AttributeType* **T**>

inline *_ConfigOption*<*T*> &**getConfigObject**()

Returns the underlying config object, casted to the specified configVariant. The config variant can be read out with the `getVariant` method.

Template Parameters
T – The config variant to be casted to.

Returns
The underlying _ConfigObject with the configuration data

template<*dv*::*Config*::*AttributeType* **T**>
inline const *_ConfigOption*<*T*> &**getConfigObject**() const

Returns the underlying config object, casted to the specified configVariant. The config variant can be read out with the `getVariant` method.

Template Parameters
T – The config variant to be casted to.

Returns
The underlying _ConfigObject with the configuration data

template<*dv*::*Config*::*AttributeType* **T**>
inline const *dv*::*Config*::*AttributeTypeGenerator*<*T*>::type &**get**() const

Returns the current value of this config option. Needs a template paramenter of the type `dv::ConfigVariant::*` to determine what type of config parameter to return.

Template Parameters
T – The config variant type

Returns
A simple value (long, string etc) that is the current value of the config option

template<*dv*::*Config*::*AttributeType* **T**>
inline void **set**(const typename *dv*::*Config*::*AttributeTypeGenerator*<*T*>::type &value, bool force)

Updates the current value of this config option. Needs a template paramenter of the type `dv::ConfigVariant::*` to determine what type of config parameter to return. The change is propagated to the configuration tree.

Template Parameters
T – The config variant type

Parameters

- **value** – A simple value (long, string etc) to update the config option with
- **force** – Force an update of the variable (useful for statistics variables)

inline void **createAttribute**(*dv*::*Config*::*Node* moduleNode, const std::string &fullKey)

Creates a dvConfig Attribute in the dv config tree for the object.

Parameters

- **moduleNode** – the module's configuration node.
- **fullKey** – the key under which the attribute is to be stored. Forward slashes (/) can be used to get sub-nodes.

inline void **updateValue**()

Updates the current value of the *ConfigOption* based on the value that is present in the dv config tree.

Public Static Functions

static inline *ConfigOption* **boolOption**(const std::string &description, bool defaultValue = false, bool readOnly = false)

Factory function. Creates boolean option (checkbox).

Parameters

- **description** – A description that describes the purpose of this option
- **defaultValue** – The default value of the option, defaults to false
- **readOnly** – If the option is read-only, defaults to false

Returns
A *ConfigOption* Object

static inline *ConfigOption* **buttonOption**(const std::string &description, const std::string &buttonLabel)

Factory function. Creates a button option. A button is a boolean that gets set when clicked. The button gets disabled as long as the boolean value is not reset

Parameters

- **description** – A description that describes the purpose of this option
- **defaultValue** – The default value of the option

Returns
A *ConfigOption* Object

static inline *ConfigOption* **intOption**(const std::string &description, int32_t defaultValue, int32_t minValue, int32_t maxValue)

Factory function. Creates a integer config option (32 bit).

Parameters

- **description** – A description that describes the purpose of this option
- **defaultValue** – The default value that this option shall have
- **minValue** – The min value a user can choose for this option
- **maxValue** – The max value a user can choose for this option

Returns
A *ConfigOption* Object

static inline *ConfigOption* **intOption**(const std::string &description, int32_t defaultValue)

Factory function. Creates a integer config option (32 bit).

Parameters

- **description** – A description that describes the purpose of this option
- **defaultValue** – The default value that this option shall have

Returns
A *ConfigOption* Object

static inline *ConfigOption* **longOption**(const std::string &description, int64_t defaultValue, int64_t minValue, int64_t maxValue)

Factory function. Creates a long integer config option (64 bit).

Parameters

- **description** – A description that describes the purpose of this option
- **defaultValue** – The default value that this option shall have
- **minValue** – The min value a user can choose for this option
- **maxValue** – The max value a user can choose for this option

Returns
A *ConfigOption* Object

static inline *ConfigOption* **longOption**(const std::string &description, int64_t defaultValue)

Factory function. Creates a long integer config option (64 bit).

Parameters

- **description** – A description that describes the purpose of this option
- **defaultValue** – The default value that this option shall have

Returns
A *ConfigOption* Object

static inline *ConfigOption* **floatOption**(const std::string &description, float defaultValue, float minValue, float maxValue)

Factory function. Creates a single-precision floating point config option.

Parameters

- **description** – A description that describes the purpose of this option
- **defaultValue** – The default value that this option shall have
- **minValue** – The min value a user can choose for this option
- **maxValue** – The max value a user can choose for this option

Returns
A *ConfigOption* Object

static inline *ConfigOption* **floatOption**(const std::string &description, float defaultValue)

Factory function. Creates a single-precision floating point config option.

Parameters

- **description** – A description that describes the purpose of this option
- **defaultValue** – The default value that this option shall have

Returns
A *ConfigOption* Object

static inline *ConfigOption* **doubleOption**(const std::string &description, double defaultValue, double minValue, double maxValue)

Factory function. Creates a double-precision floating point config option.

Parameters

- **description** – A description that describes the purpose of this option
- **defaultValue** – The default value that this option shall have
- **minValue** – The min value a user can choose for this option
- **maxValue** – The max value a user can choose for this option

Returns
A *ConfigOption* Object

static inline *ConfigOption* **doubleOption**(const std::string &description, double defaultValue)

Factory function. Creates a double-precision floating point config option.

Parameters

- **description** – A description that describes the purpose of this option
- **defaultValue** – The default value that this option shall have

Returns
A *ConfigOption* Object

static inline *ConfigOption* **stringOption**(const std::string &description, const std::string &defaultValue)

Factory function. Creates a string config option.

Parameters

- **description** – A description that describes the purpose of this option
- **defaultValue** – The default value that this option shall have

Returns
A *ConfigOption* Object

static inline *ConfigOption* **stringOption**(const std::string &description, const std::string &defaultValue, int32_t minLength, int32_t maxLength)

Factory function. Creates a string config option.

Parameters

- **description** – A description that describes the purpose of this option
- **defaultValue** – The default value that this option shall have
- **minValue** – The min string length a user can input
- **maxValue** – The max string length a user can input

Returns
A *ConfigOption* Object

static inline *ConfigOption* **listOption**(const std::string &description, size_t defaultChoice, const std::vector<std::string> &choices, bool allowMultipleSelection = false)

Factory function. Creates a list config option (list of strings).

Parameters

- **description** – A description that describes the purpose of this option
- **defaultChoice** – The index of the default choice, taken from the subsequent vector
- **choices** – Vector of possible choices
- **allowMultipleSelection** – wether to allow selecting multiple options at the same time

Returns
A *ConfigOption* Object

static inline *ConfigOption* **listOption**(const std::string &description, const std::string &defaultChoice, const std::vector<std::string> &choices, bool allowMultipleSelection = false)

Factory function. Creates a list config option (list of strings).

Parameters

- **description** – A description that describes the purpose of this option
- **defaultChoice** – The the string of the default choice
- **choices** – Vector of possible choices
- **allowMultipleSelection** – wether to allow selecting multiple options at the same time

Returns
A *ConfigOption* Object

static inline *ConfigOption* **fileOpenOption**(const std::string &description)

Factory function. Creates a file open config option.

Parameters
description – A description that describes the purpose of this option

Returns
A *ConfigOption* Object

static inline *ConfigOption* **fileOpenOption**(const std::string &description, const std::string &allowedExtensions)

Factory function. Creates a file open config option.

Parameters

- **description** – A description that describes the purpose of this option
- **allowedExtensions** – The allowed extensions of the file to be opened

Returns
A *ConfigOption* Object

static inline *ConfigOption* **fileOpenOption**(const std::string &description, const std::string &defaultValue, const std::string &allowedExtensions)

Factory function. Creates a file open config option.

Parameters

- **description** – A description that describes the purpose of this option
- **defaultValue** – The default path of the file
- **allowedExtensions** – The allowed extensions of the file to be opened

Returns
A *ConfigOption* Object

static inline *ConfigOption* **fileSaveOption**(const std::string &description)

Factory function. Creates a file save config option.

Parameters
description – A description that describes the purpose of this option

Returns
A *ConfigOption* Object

static inline *ConfigOption* **fileSaveOption**(const std::string &description, const std::string &allowedExtensions)

Factory function. Creates a file save config option.

Parameters

- **description** – A description that describes the purpose of this option

- **allowedExtensions** – The allowed extensions of the file to be saved

Returns
A *ConfigOption* Object

static inline *ConfigOption* **fileSaveOption**(const std::string &description, const std::string &defaultValue, const std::string &allowedExtensions)

Factory function. Creates a file save config option.

Parameters

- **description** – A description that describes the purpose of this option
- **defaultValue** – The default path of the file
- **allowedExtensions** – The allowed extensions of the file to be saved

Returns
A *ConfigOption* Object

static inline *ConfigOption* **directoryOption**(const std::string &description)

Factory function. Creates directory choose option

Parameters
description – A description that describes the purpose of this option

Returns
A *ConfigOption* Object

static inline *ConfigOption* **directoryOption**(const std::string &description, const std::string &defaultValue)

Factory function. Creates directory choose option

Parameters

- **description** – A description that describes the purpose of this option
- **defaultValue** – The default path of the directory

Returns
A *ConfigOption* Object

static inline *ConfigOption* **statisticOption**(const std::string &description)

Factory function. Creates read-only statistics option.

Parameters
description – A description that describes the purpose of this option

Returns
A *ConfigOption* Object

Private Functions

inline **ConfigOption**(*dv*::unique_ptr_void configOption_, *dv*::*Config*::*AttributeType* type_)

Private constructor Takes shared_ptr to templated *_ConfigOption* as well as the variant.

Parameters

- **configOption_** – A unique_ptr to an instantiated *_ConfigOption*
- **type_** – The config variant of the passed option

inline void **setNodeAttrLink**(*dv*::*Config*::*Node* moduleNode, const std::string &fullKey)

Set link to actual node and attribute for configuration tree operations. Must be set for tree operations (create, update etc.) to work.

Parameters

- **moduleNode** – the module's configuration node.
- **fullKey** – the key under which the attribute is to be stored.

template<*dv*::*Config*::*AttributeType* **T**>
inline void **_updateValue**()

Updates the current value of the *ConfigOption* based on the value that is present in the dv config tree.

Private Members

dv::unique_ptr_void **configOption**

dv::*Config*::*AttributeType* **type**

dv::*Config*::*Node* **node**

std::string **key**

std::unique_ptr<*_RateLimiter*> **rateLimit**

Private Static Functions

template<*dv*::*Config*::*AttributeType* **T**>
static inline *ConfigOption* **getOption**(const std::string &description, typename *dv*::*Config*::*AttributeTypeGenerator*<*T*>::type defaultValue, const *_ConfigAttributes*<*T*> &attributes, *dv*::*Config*::*AttributeFlags* flags, bool updateReadOnly = false)

Private base factory method. Used as a base for all the config factories defined Creates a new *ConfigOption* of the requested type. Works by first instantiating a `_ConfigOption` with the right templated variant, then creating a `unique_ptr` and creating a `ConfigOption` to return.

Returns

class **CycleTime** : public *dv*::*statistics*::*Stats*<float>

#include </builds/inivation/dv/dv-runtime/include/dv-sdk/stats.hpp> Class which provides the functionality to measure statistics for cycle time related data.

Public Functions

CycleTime() = delete

Deleted default constructor

inline **CycleTime**(const *dv*::*Config*::*Node* node, const std::string_view name)

Constructor which selects the default window size for the underlying statistics.

The data will be published to the node `node/name/cycleTime/` if a name is specified, or `node/cycleTime/` if none is specified.

See also

Stats.

See also

Stats

Parameters

- **node** – Parent node relative to which the node for the measured stats should be published
- **name** – Name of the node

inline **CycleTime**(const uint32_t windowSize, const *dv*::*Config*::*Node* node, const std::string_view name)

Constructor which applies the a custom window size for the underlying statistics.

The data will be published to the node `node/name/cycleTime/` if a name is specified, or `node/cycleTime/ if none is specified.

See also

Stats.

See also

Stats

Parameters

- **node** – Parent node relative to which the node for the measured stats should be published
- **name** – Name of the node

CycleTime(const *CycleTime* &other) = delete

Copy - Disallow: since node and name are the same, you'd suddenly have two objects feeding data into the configuration backend and overwriting each other. This is not what you want.

CycleTime &**operator=**(const *CycleTime* &rhs) = delete

CycleTime(*CycleTime* &&other) = default

Move

CycleTime &**operator=**(*CycleTime* &&rhs) = default

~CycleTime() noexcept = default

Destructor

inline void **start**() noexcept

Starts a measurement

inline void **finish**() noexcept

Finishes a measurement and adds it to the underlying statistical accumulators

Private Members

std::chrono::time_point<std::chrono::steady_clock> **mStartTime** = {std::chrono::steady_clock::now()}

union **dvConfigAttributeRange**

Public Members

int32_t **intRange**

int64_t **longRange**

float **floatRange**

double **doubleRange**

int32_t **stringRange**

struct **dvConfigAttributeRanges**

Public Members

union *dvConfigAttributeRange* **min**

union *dvConfigAttributeRange* **max**

union **dvConfigAttributeValue**

Public Members

bool **boolean**

int32_t **iint**

int64_t **ilong**

float **ffloat**

double **ddouble**

char ***string**

struct **dvModuleDataS**

Public Members

dvConfigNode **moduleNode**

void ***moduleState**

struct **dvModuleFunctionsS**

Public Members

bool (*const **moduleStaticInit**)(*dvModuleData* moduleData)

bool (*const **moduleInit**)(*dvModuleData* moduleData)

void (*const **moduleRun**)(*dvModuleData* moduleData)

void (*const **moduleConfig**)(*dvModuleData* moduleData)

void (*const **moduleExit**)(*dvModuleData* moduleData)

struct **dvModuleInfoS**

Public Members

int32_t **version**

const char ***description**

size_t **memSize**

dvModuleFunctions **functions**

struct **dvType**

Public Members

uint32_t **id**

const char ***identifier**

const char ***description**

size_t **sizeOfType**

dvTypePackFuncPtr **pack**

dvTypeUnpackFuncPtr **unpack**

dvTypeConstructPtr **construct**

dvTypeDestructPtr **destruct**

dvTypeTimeElementExtractorPtr **timeElementExtractor**

dvTypeUnpackTimeElementRangeFuncPtr **unpackTimeElementRange**

struct **dvTypedObject**

Public Members

uint32_t **typeId**

size_t **objSize**

void ***obj**

struct **dvTypeTimeElementExtractor**

Public Members

int64_t **startTimestamp**

int64_t **endTimestamp**

int64_t **numElements**

class **Helper**

Public Static Functions

static inline std::string **typeToStringConverter**(*AttributeType* type)

static inline *AttributeType* **stringToTypeConverter**(const std::string &typeString)

static inline std::string **valueToStringConverter**(*AttributeType* type, union *dvConfigAttributeValue* value)

static inline union *dvConfigAttributeValue* **stringToValueConverter**(*AttributeType* type, const std::string &valueString)

static inline std::string **flagsToStringConverter**(int flags)

static inline int **stringToFlagsConverter**(const std::string &flagsString)

static inline std::string **rangesToStringConverter**(*AttributeType* type, struct *dvConfigAttributeRanges* ranges)

static inline struct *dvConfigAttributeRanges* **stringToRangesConverter**(*AttributeType* type, const std::string &rangesString)

template<typename **T**>

class **InputDataWrapper** : public *dv*::*_InputDataWrapperCommon*<*T*>

#include </builds/inivation/dv/dv-runtime/include/dv-sdk/data/wrappers.hpp> Generic case input data wrapper. Used for all types that do not have a more specific input data wrapper.

Template Parameters
T – The type of the input data

Public Functions

inline **InputDataWrapper**(std::shared_ptr<const *T*> p)

inline const *T* &**operator***() const noexcept

inline const *T* ***operator->**() const noexcept

template<>

class **InputDataWrapper**<*dv*::Frame> : public *dv*::*_InputDataWrapperCommon*<*dv*::Frame>

Public Functions

inline **InputDataWrapper**(std::shared_ptr<const *dv*::Frame> p)

inline *dv*::FrameFormat **format**() const noexcept

Returns the dv frame format of the current frame

Returns
the dv frame format of the current frame

inline int16_t **sizeX**() const noexcept

The width of the current frame

Returns
the width of the current frame

inline int16_t **sizeY**() const noexcept

The height of the current frame

Returns
the height of the current frame

inline int16_t **positionX**() const noexcept

Returns the position in x of the region of interest (ROI) of the current frame

Returns
the x coordinate of the start of the region of interest

inline int16_t **positionY**() const noexcept

Returns the position in y of the region of interest (ROI) of the current frame

Returns
the y coordinate of the start of the region of interest

inline int64_t **timestamp**() const noexcept

Returns the timestamp of the start of exposure for this frame, or whatever closest value is available for this sensor.

Returns
the timestamp of the start of exposure for this frame

inline *dv*::Duration **exposure**() const noexcept

Returns the exposure time of this frame

Returns
The exposure time of this frame

inline int64_t **timestampStartOfExposure**() const noexcept

Returns the timestamp of the start of exposure for this frame

Returns
the timestamp of the start of exposure for this frame

inline int64_t **timestampEndOfExposure**() const noexcept

Returns the timestamp of the end of exposure for this frame

Returns
the timestamp of the end of exposure for this frame

inline *dv*::FrameSource **source**() const noexcept

Returns the source (creation method) of this frame

Returns
The source (creation method) of this frame

inline std::unique_ptr<const cv::Mat> **getMatPointer**() const noexcept

Return a read-only OpenCV Mat representing this frame. It is possible to bypass the read-only restriction on a cv::Mat rather easily by dereferencing it and copy-constructing a new cv::Mat, which will point to the same data because that's how cv::Mat works. Please don't do this, it will introduce subtle and terrifying bugs.

Returns
a read-only OpenCV Mat pointer

inline cv::Mat **getMatCopy**() const noexcept

Return an OpenCV Mat representing a copy of this frame. Contents are freely modifiable.

Returns
an OpenCV Mat

inline cv::Size **size**() const

Returns the size of the current frame. This is generally the same as the input dimension, but depending on the ROI setting, can be less

Returns
The size of the current frame

inline cv::Point **position**() const

Returns the position of the ROI of the current frame.

Returns
The position of the start of the ROI of the current frame

inline cv::Rect **roi**() const

Returns the rectangle defining the region of interest of the current frame

Returns
The rectangle describing the region of interest of the current frame

class **InputDefinition**

#include </builds/inivation/dv/dv-runtime/include/dv-sdk/module_io.hpp> Definition of a module input. Every input to a module has a unique (for the module) name, as well as a type. The type is the 4-character flatbuffers identifier

Public Functions

inline **InputDefinition**(const std::string &n, const std::string &t, bool opt = false)

Public Members

std::string **name**

std::string **typeName**

bool **optional**

class **InputDefinitionList**

#include </builds/inivation/dv/dv-runtime/include/dv-sdk/module_io.hpp> Vector decorator that gives convenience methods to add various kinds of inputs to a module

Public Functions

inline void **addInput**(const std::string &name, const std::string &typeIdentifier, bool optional = false)

Adds an input of a generic type to this module.

Parameters

- **name** – The name of this input
- **typeIdentifier** – The identifier of the FlatBuffers data type used for this input
- **optional** – A flag that describes if this input is optional or not. Optional inputs are not required to be connected for successful module startup.

inline void **addEventInput**(const std::string &name, bool optional = false)

Adds an event data input to this module.

Parameters

- **name** – The name of the event data input
- **optional** – A flag to set this input as optional

inline void **addFrameInput**(const std::string &name, bool optional = false)

Adds a frame input to this module.

Parameters

- **name** – The name of the frame input
- **optional** – A flag to set this input as optional

inline void **addIMUInput**(const std::string &name, bool optional = false)

Adds an IMU input to this module.

Parameters

- **name** – The name of the IMU input
- **optional** – A flag to set this input as optional

inline void **addTriggerInput**(const std::string &name, bool optional = false)

Adds a trigger input to this module.

Parameters

- **name** – The name of the trigger input
- **optional** – A flag to set this input as optional

inline void **addBoundingBoxInput**(const std::string &name, bool optional = false)

Adds a bounding box input to this module.

Parameters

- **name** – The name of the bounding box input
- **optional** – A flag to set this input as optional

inline const std::vector<*InputDefinition*> &**getInputs**() const

INTERNAL USE Returns the list of configured input definitions

Returns
The list of configured input definitions

Private Members

std::vector<*InputDefinition*> **inputs**

template<typename **T**, typename **U**>

class **InputVectorDataWrapper** : public *dv*::*_InputVectorDataWrapperCommon*<*T*, *U*>

#include </builds/inivation/dv/dv-runtime/include/dv-sdk/data/wrappers.hpp> Generic case input data wrapper for vector-like data (must have a std::vector<U> called 'elements'). Used for all vector types that do not have a more specific input data wrapper.

Template Parameters

- **T** – The type of the vector input data
- **U** – The type of the vector input data's elements

Public Functions

inline **InputVectorDataWrapper**(std::shared_ptr<const *T*> p)

inline const *T* &**operator***() const noexcept

inline const *T* ***operator->**() const noexcept

template<>

class **InputVectorDataWrapper**<*dv*::BoundingBoxPacket, *dv*::BoundingBox> : public *dv*::*_InputVectorDataWrapperCommon*<*dv*::BoundingBoxPacket, *dv*::BoundingBox>

Public Functions

inline **InputVectorDataWrapper**(std::shared_ptr<const *dv*::BoundingBoxPacket> p)

template<>

class **InputVectorDataWrapper**<*dv*::EventPacket, *dv*::Event> : public *dv*::*_InputVectorDataWrapperCommon*<*dv*::EventPacket, *dv*::Event>

Public Functions

inline **InputVectorDataWrapper**(std::shared_ptr<const *dv*::EventPacket> p)

inline **operator *dv*::EventStore**() const noexcept

Conversion operator for dv::EventStore. Provides backwards compatibility, due to dv::EventStore having stricter, explicit constructors now.

Returns

an EventStore sharing this input EventPacket.

template<>

class **InputVectorDataWrapper**<*dv*::IMUPacket, *dv*::IMU> : public *dv*::*_InputVectorDataWrapperCommon*<*dv*::IMUPacket, *dv*::IMU>

Public Functions

inline **InputVectorDataWrapper**(std::shared_ptr<const IMUPacket> p)

template<>

class **InputVectorDataWrapper**<*dv*::TriggerPacket, *dv*::Trigger> : public *dv*::*_InputVectorDataWrapperCommon*<*dv*::TriggerPacket, *dv*::Trigger>

Public Functions

inline **InputVectorDataWrapper**(std::shared_ptr<const *dv*::TriggerPacket> p)

struct **logEndType**

#include </builds/inivation/dv/dv-runtime/include/dv-sdk/log.hpp> DV custom log end type. Can be piped into the log to commit it. Eg. `log.info << "Hello " << "world" << dv::logEnd`.

class **Logger**

#include </builds/inivation/dv/dv-runtime/include/dv-sdk/log.hpp> *Logger* class. This class provides logging streams for different log levels to log to.

Public Members

LogStream<*dv*::logLevel::DEBUG> **debug**

LogStream<*dv*::logLevel::INFO> **info**

LogStream<*dv*::logLevel::WARNING> **warning**

LogStream<*dv*::logLevel::ERROR> **error**

template<*dv*::logLevel **L**>

class **LogStream**

#include </builds/inivation/dv/dv-runtime/include/dv-sdk/log.hpp> *LogStream* class. To be instantiated only by *dv::Logger*. Handles the actual logging and provides functions to stream logging data in as well as to format logging strings.

Template Parameters
L – The log level for which this stream should be configured.

Public Functions

template<class **T**>
inline void **operator()**(const *T* &val)

Logs the given argument value.

Template Parameters
T – the type of the argument value

Parameters
val – the value to be logged

template<class **T**>
inline *LogStream* &**operator<<**(const *T* &val)

Appends the given argument value to the current log message. To commit a log message (write it out and flush), one can stream in `dv::logEnd` or `std::endl`.

Template Parameters
T – The type of the value to be appended to the log

Parameters
val – The value to be appended to the log

Returns
A reference to the stream, to stream in further values

inline *LogStream* &**operator<<**(std::ostream &(*)(std::ostream&))

Commits a log message upon reception of an object of the type of `std::endl`. Committing a message means writing it out and flushing the buffer.

Returns
a reference to the log stream for the reception of further values to be logged.

inline *LogStream* &**operator<<**(*logEndType*)

Commits a log message upon reception of `dv::logEnd`. Committing a message means writing it out and flushing the buffer.

Returns
a reference to the log stream for the reception of further values to be logged.

template<class **T**>
inline void **write**(const *T* &val)

Appends the given argument value to the current log message. To commit a log message (write it out and flush), one can use the function `commit()`

Template Parameters
T – The type of the value to be appended to the log

Parameters
val – The value to be appended to the log

inline void **commit**()

Commits the current log message. Committing means writing the message out and flushing the buffer.

inline void **flush**()

Flushes the buffer without writing out the message.

template<typename ...**Args**>
inline void **format**(const fmt::format_string<*Args*...> format, *Args*&&... args)

Formats the given format string with the given arguments and writes it to the log. Takes any `fmt::format` format string.

Template Parameters
Args – The types of the arguments

Parameters

- **format** – The format string
- **args** – The argument values to be formatted

Private Members

std::ostringstream **stream_**

internal string stream that gets used to assemble the log message.

class **ModuleBase**

#include </builds/inivation/dv/dv-runtime/include/dv-sdk/module_base.hpp> The dv *ModuleBase*. Every module shall inherit from this module. The base Module provides the following:

- Abstraction of the configuration
- Input / output management

Public Functions

inline **ModuleBase**()

Base module constructor. The base module constructor initializes the logger and config members of the class, by utilizing the `static_thread` local pointer to the DV moduleData pointer provided prior to constructrion. This makes sure, that logger and config are available at the time the subclass constructor is called.

virtual **~ModuleBase**() = default

ModuleBase(const *ModuleBase* &m) = delete

ModuleBase(*ModuleBase* &&m) = delete

ModuleBase &**operator=**(const *ModuleBase* &rhs) = delete

ModuleBase &**operator=**(*ModuleBase* &&rhs) = delete

inline void **configInternal**()

Internal method that gets called whenever a config gets changed. It first merges the new config from the config tree into a runtime dictionary (map). After that, a user-provided function is called with which the user has the possibility to copy out config values.

inline virtual void **configUpdate**()

Virtual function to be implemented by the user. Can be left empty. Called on configuration update, allows more advanced control of how configuration values are updated.

inline void **runInternal**()

Internal method that gets called to run the main user code. Will call the user-provided *run()* function and then advance all input data objects by one.

virtual void **run**() = 0

Virtual function to be implemented by the user. Main function that runs the module and handles data.

Public Members

dvModuleData **moduleData**

DV low-level module data. Use it to access the low-level DV API.

Logger **log**

Loggers for the module. Each module has their own to avoid interference.

dv::*Config*::*Node* **moduleNode**

The module configuration node. Use it to access configuration at a lower level than the RuntimeConfigMap.

RuntimeConfig **config**

Allows easy access to configuration data and is automatically updated with new values on changes in the configuration tree.

RuntimeInputs **inputs**

Access data inputs and related actions in a type-safe manner.

RuntimeOutputs **outputs**

Access data outputs and related actions in a type-safe manner.

Public Static Functions

static inline void **staticConfigInit**(*dv*::*Config*::*Node* moduleNode)

Static config init function. Calles the user provided `initConfigOptions` function which exists in this class as a static called `__getDefaultConfig`. It generates the default config and creates the elements for the default config in the DV config tree.

Parameters

moduleNode – The dvConfig node for which the config should be generated.

static inline void **__setStaticModuleData**(*dvModuleData* _moduleData)

INTERNAL USE ONLY Sets the static, thread local module data to be used by a subsequent constructor. This shall only be used prior to

Parameters

_moduleData – The moduleData param to be used for *ModuleBase* member initialization upon constructor

static inline void **__setStaticGetDefaultConfig**(void (*_getDefaultConfig)(*RuntimeConfig*&))

INTERNAL USE ONLY Sets the `__getDefaultConfig` static function to the user provided static function that generates the default config map. The reference to this function is used since there is no access to the child - subclass static functions possible from this class. The default config is both generated before instantiation in a call to `staticConfigInit` as well as in the constructor at runtime.

Parameters

_getDefaultConfig –

Private Static Attributes

static thread_local *dvModuleData* **__moduleData** = {nullptr}

moduleData and getDefaultConfig are thread-local to avoid race-conditions between them being initialized here as nullptr and being set in module.hpp. moduleData is different at runtime for each module and must be thread-local. getDefaultConfig is non-atomic and is thread-local for extra safety.

static thread_local void (***__getDefaultConfig**)(*RuntimeConfig*&)

template<class **T**>

class **ModuleStatics**

#include </builds/inivation/dv/dv-runtime/include/dv-sdk/module.hpp> Pure static template class that provides the static C interface functions for the module class `T` to be exposed to DV. It essentially wraps the the functions of the given C++ module class to the stateless functions and external state of the DV interface. Template parameter T must be a valid DV module, e.g. it has to extend *dv::ModuleBase* and provide certain static functions. Upon (static) instantiation, the *ModuleStatics* class performs static (compile time) validations to check if `T` is a valid module.

Template Parameters

T – A valid DV Module class that extends *dv::ModuleBase*

Public Static Functions

static inline bool **staticInit**(*dvModuleData* moduleData)

Wrapper for the `staticInit` DV function. Performs a static call to the `configInit<T>` function of *ModuleBase*, which in turn gets the config from the user defined module `T`. The config then gets parsed and injected as DvConfig nodes.

Parameters

moduleData – The DV provided moduleData.

Returns

true if static initialization succeeded, false if it failed.

static inline bool **init**(*dvModuleData* moduleData)

Wrapper for the `init` DV function. Constructs the user defined `T` module into the module state. Configuration is updated by the *ModuleBase* constructor.

Parameters
moduleData – The DV provided moduleData.

Returns
true if runtime initialization succeeded, false if it failed.

static inline void **run**(*dvModuleData* moduleData)

Wrapper for the stateless `run` DV function. Relays the call to the user defined `run` function of the user defined `T` module, which exists as the module state.

Parameters

- **moduleData** – The DV provided moduleData. Used to extract the state.
- **in** – The input data to be processed by the module.
- **out** – Pointer to the output data.

static inline void **exit**(*dvModuleData* moduleData)

Deconstructs the user defined `T` module in the state by calling its destructor.

Parameters
moduleData – The DV provided moduleData.

static inline void **config**(*dvModuleData* moduleData)

Wrapper for the DV config function. Relays the call to the stateful `configUpdate` function of the `T` module. If not overloaded by a the user, the `configUpdate` function of *`ModuleBase`* is called which reads out all config from the DvConfig node and updates a runtime dict of configs.

Parameters
moduleData – The moduleData provided by DV.

Public Static Attributes

static const struct *dvModuleFunctionsS* **functions** = {&*ModuleStatics*<*T*>::staticInit, &*ModuleStatics*<*T*>::init, &*ModuleStatics*<*T*>::run, &*ModuleStatics*<*T*>::config, &*ModuleStatics*<*T*>::exit}

Static definition of the *dvModuleFunctionsS* struct. This struct gets filles with the static wrapper functions provided in this class at compile time.

Static definition of the *`ModuleStatics::functions`* struct. This struct contains the addresses to all the wrapper functions instantiated to the template module `T`. This struct is then passed to DV to allow it to access the functionalities inside the module.

Template Parameters
T – The user defined module. Must inherit from *`dv::ModuleBase`*

static const struct *dvModuleInfoS* **info** = {1, *T*::initDescription(), sizeof(*T*), &*functions*}

Static definition of the *dvModuleInfoS* struct. This struct gets filled with the static information from the user provided `T` module.

Static definition of the info struct, which gets passed to DV. DV reads this and uses the information to call into the module. It gets instantiated with the template parameter T, which has to be a valid module and inherit from *`dv::ModuleBase`*.

Template Parameters
T – The user defined module. Must inherit from *`dv::ModuleBase`*

class **Node**

Public Functions

inline **Node**(*dvConfigNode* n)

inline explicit **operator** ***dvConfigNode***()

inline explicit **operator** ***dvConfigNodeConst***() const

inline explicit **operator bool**() const

inline std::string **getName**() const

inline std::string **getPath**() const

inline *Node* **getParent**() const

This returns a reference to a node, and as such must be carefully mediated with any dvConfigNodeRemoveNode() calls.

Throws

`std::out_of_range` – *Node* is root and has no parent.

inline std::vector<*Node*> **getChildren**() const

This returns a reference to a node, and as such must be carefully mediated with any dvConfigNodeRemoveNode() calls.

inline void **addNodeListener**(void *userData, *dvConfigNodeChangeListener* node_changed)

inline void **removeNodeListener**(void *userData, *dvConfigNodeChangeListener* node_changed)

inline void **removeAllNodeListeners**()

inline void **addAttributeListener**(void *userData, *dvConfigAttributeChangeListener* attribute_changed)

inline void **removeAttributeListener**(void *userData, *dvConfigAttributeChangeListener* attribute_changed)

inline void **removeAllAttributeListeners**()

inline void **removeNode**()

Careful, only use if no references exist to this node and all its children. References are created by *dv::Config::Tree::getNode()*, *dv::Config::Node::getRelativeNode()*, *dv::Config::Node::getParent()* and *dv::Config::Node::getChildren()*.

inline void **removeSubTree**()

Careful, only use if no references exist to this node's children. References are created by dvConfigTreeGetNode(), dvConfigNodeGetRelativeNode(), dvConfigNodeGetParent() and dvConfigNodeGetChildren().

inline void **clearSubTree**(bool clearThisNode)

inline void **copyTo**(*dv*::*Config*::*Node* destination) const

template<*AttributeType* **T**>
inline void **createAttribute**(const std::string &key, const *AttributeValue*<*T*> &defaultValue, const *AttributeRanges*<*T*> &ranges, *AttributeFlags* flags, const std::string &description)

inline void **createAttribute**(const std::string &key, *AttributeType* type, const *dvConfigAttributeValue* defaultValue, const *dvConfigAttributeRanges* &ranges, *AttributeFlags* flags, const std::string &description)

template<*AttributeType* **T**>

inline void **removeAttribute**(const std::string &key)

inline void **removeAttribute**(const std::string &key, *AttributeType* type)

inline void **removeAllAttributes**()

template<*AttributeType* **T**>
inline bool **existsAttribute**(const std::string &key) const

inline bool **existsAttribute**(const std::string &key, *AttributeType* type) const

template<*AttributeType* **T**>
inline bool **putAttribute**(const std::string &key, const *AttributeValue*<*T*> &value)

inline bool **putAttribute**(const std::string &key, *AttributeType* type, const *dvConfigAttributeValue* value)

template<*AttributeType* **T**>
inline *AttributeValue*<*T*> **getAttribute**(const std::string &key) const

inline *dvConfigAttributeValue* **getAttribute**(const std::string &key, *AttributeType* type) const

template<*AttributeType* **T**>
inline bool **updateReadOnlyAttribute**(const std::string &key, const *AttributeValue*<*T*> &value)

inline bool **updateReadOnlyAttribute**(const std::string &key, *AttributeType* type, const *dvConfigAttributeValue* value)

template<*AttributeType* **T**>
inline void **create**(const std::string &key, typename *AttributeTypeGenerator*<*T*>::type defaultValue, const *AttributeRanges*<*T*> &ranges, *AttributeFlags* flags, const std::string &description)

template<*AttributeType* **T**>
inline void **remove**(const std::string &key)

template<*AttributeType* **T**>
inline bool **exists**(const std::string &key) const

template<*AttributeType* **T**>
inline bool **put**(const std::string &key, typename *AttributeTypeGenerator*<*T*>::type value)

template<*AttributeType* **T**>
inline bool **updateReadOnly**(const std::string &key, typename *AttributeTypeGenerator*<*T*>::type value)

template<*AttributeType* **T**>
inline *AttributeTypeGenerator*<*T*>::type **get**(const std::string &key) const

inline bool **putInt**(const std::string &key, int32_t val)

inline bool **putLong**(const std::string &key, int64_t val)

inline bool **putFloat**(const std::string &key, float val)

inline bool **putDouble**(const std::string &key, double val)

inline bool **putString**(const std::string &key, const std::string &val)

inline bool **putBool**(const std::string &key, bool val)

inline int32_t **getInt**(const std::string &key) const

inline int64_t **getLong**(const std::string &key) const

inline float **getFloat**(const std::string &key) const

inline double **getDouble**(const std::string &key) const

inline std::string **getString**(const std::string &key) const

inline bool **getBool**(const std::string &key) const

inline bool **exportNodeToXML**(const std::string &filePath, bool exportAll = false) const

inline bool **exportSubTreeToXML**(const std::string &filePath, bool exportAll = false) const

inline bool **importNodeFromXML**(const std::string &filePath, bool strict = true)

inline bool **importSubTreeFromXML**(const std::string &filePath, bool strict = true)

inline std::string **exportNodeToXMLString**(bool exportAll = false) const

inline std::string **exportSubTreeToXMLString**(bool exportAll = false) const

inline bool **importNodeFromXMLString**(const std::string &xmlString, bool strict = true)

inline bool **importSubTreeFromXMLString**(const std::string &xmlString, bool strict = true)

inline bool **stringToAttributeConverter**(const std::string &key, const std::string &type, const std::string &value, bool overrideReadOnly = false)

inline std::vector<std::string> **getChildNames**() const

inline std::vector<std::string> **getAttributeKeys**() const

inline *AttributeType* **getAttributeType**(const std::string &key) const

template<*AttributeType* **T**>
inline *AttributeRanges*<*T*> **getAttributeRanges**(const std::string &key) const

inline struct *dvConfigAttributeRanges* **getAttributeRanges**(const std::string &key, *AttributeType* type) const

template<*AttributeType* **T**>
inline int **getAttributeFlags**(const std::string &key) const

inline int **getAttributeFlags**(const std::string &key, *AttributeType* type) const

template<*AttributeType* **T**>
inline std::string **getAttributeDescription**(const std::string &key) const

inline std::string **getAttributeDescription**(const std::string &key, *AttributeType* type) const

inline void **attributeModifierButton**(const std::string &key, const std::string &buttonLabel)

inline void **attributeModifierListOptions**(const std::string &key, const std::string &listOptions, bool allowMultipleSelections)

inline void **attributeModifierFileChooser**(const std::string &key, const std::string &typeAndExtensions)

inline void **attributeModifierUnit**(const std::string &key, const std::string &unitInformation)

inline void **attributeModifierPriorityAttributes**(const std::string &priorityAttributes)

inline void **attributeModifierGUISupport**()

inline void **attributeBooleanReset**(const std::string &key)

inline bool **existsRelativeNode**(const std::string &nodePath) const

inline *Node* **getRelativeNode**(const std::string &relativeNodePath) const

This returns a reference to a node, and as such must be carefully mediated with any dvConfigNodeRemoveNode() calls.

Throws

`std::out_of_range` – Invalid relative node path.

inline void **attributeUpdaterAdd**(const std::string &key, *AttributeType* type, *dvConfigAttributeUpdater* updater, void *updaterUserData, bool runOnce = false)

inline void **attributeUpdaterRemove**(const std::string &key, *AttributeType* type, *dvConfigAttributeUpdater* updater, void *updaterUserData)

inline void **attributeUpdaterRemoveAll**()

Private Members

dvConfigNode **node**

template<typename **T**>

class **OutputDataWrapper** : public *dv*::*_OutputDataWrapperCommon*<*T*>

#include </builds/inivation/dv/dv-runtime/include/dv-sdk/data/wrappers.hpp> Generic case output data wrapper. Used for all types that do not have a more specific output data wrapper.

Template Parameters

T – The type of the output data

Public Functions

inline **OutputDataWrapper**(*T* *p, *dvModuleData* m, const std::string &n)

inline *T* &**operator***() noexcept

inline const *T* &**operator***() const noexcept

inline *T* ***operator->**() noexcept

inline const *T* ***operator->**() const noexcept

template<>

class **OutputDataWrapper**<*dv*::Frame> : public *dv*::*_OutputDataWrapperCommon*<*dv*::Frame>

Public Functions

inline **OutputDataWrapper**(*dv*::Frame *p, *dvModuleData* m, const std::string &n, const cv::Size maxSize)

inline *OutputDataWrapper* &**operator=**(const *_InputDataWrapperCommon*<*dv*::Frame> &rhs)

inline void **commit**()

inline *OutputDataWrapper* &**operator<<**(*dv*::*commitType*)

inline void **setMat**(const cv::Mat &mat)

Copies the supplied OpenCV matrix and sends it to the output.

Parameters
mat – The matrix to be sent to the output

inline void **setMat**(cv::Mat &&mat)

Moves the supplied OpenCV matrix and sends it to the output. Do not use it anymore afterwards!

Parameters
mat – The matrix to be sent to the output

inline *OutputDataWrapper*<*dv*::Frame> &**operator<<**(const cv::Mat &mat)

Copies the supplied OpenCV matrix and sends it to the output.

Parameters
mat – The matrix to be sent to the output

Returns
A reference to this output, to send more data

inline *OutputDataWrapper*<*dv*::Frame> &**operator<<**(cv::Mat &&mat)

Moves the supplied OpenCV matrix and sends it to the output.

Parameters
mat – The matrix to be sent to the output

Returns
A reference to this output, to send more data

inline *dv*::FrameFormat **format**() const noexcept

Returns the dv frame format of the current frame

Returns
the dv frame format of the current frame

inline int16_t **sizeX**() const noexcept

The width of the current frame

Returns
the width of the current frame

inline int16_t **sizeY**() const noexcept

The height of the current frame

Returns
the height of the current frame

inline int16_t **positionX**() const noexcept

Returns the position in x of the region of interest (ROI) of the current frame

Returns
the x coordinate of the start of the region of interest

inline int16_t **positionY**() const noexcept

Returns the position in y of the region of interest (ROI) of the current frame

Returns
the y coordinate of the start of the region of interest

inline int64_t **timestamp**() const noexcept

Returns the timestamp of this frame (start of exposure or closest)

Returns
The timestamp of this frame (start of exposure or closest)

inline *dv*::Duration **exposure**() const noexcept

Returns the exposure time of this frame

Returns
The exposure time of this frame

inline int64_t **timestampStartOfExposure**() const noexcept

Returns the timestamp of the start of exposure for this frame

Returns
the timestamp of the start of exposure for this frame

inline int64_t **timestampEndOfExposure**() const noexcept

Returns the timestamp of the end of exposure for this frame

Returns
the timestamp of the end of exposure for this frame

inline *dv*::FrameSource **source**() const noexcept

Returns the source (creation method) of this frame

Returns
The source (creation method) of this frame

inline void **setPosition**(int16_t positionX, int16_t positionY) noexcept

Sets the ROI start position for the output frame

inline void **setTimestamp**(int64_t timestamp) noexcept

Sets the timestamp of the output frame (start of exposure or closest)

inline *OutputDataWrapper*<*dv*::Frame> &**operator<<**(int64_t timestamp)

Sets the timestamp of the current frame on the output (start of exposure or closest).

Parameters
timestamp – The timestamp to be set

Returns
A reference to this data, to commit more data

inline void **setExposure**(const *dv*::Duration exposure) noexcept

Sets the exposure of this frame

inline void **setSource**(const *dv*::FrameSource source) noexcept

Sets the source (creation method) of this frame

inline cv::Size **size**() const

Returns the size of the current frame. This is generally the same as the input dimension, but depending on the ROI setting, can be less

Returns
The size of the current frame

inline cv::Point **position**() const

Returns the position of the ROI of the current frame.

Returns
The position of the start of the ROI of the current frame

inline cv::Rect **roi**() const

Returns the rectangle defining the region of interest of the current frame

Returns
The rectangle describing the region of interest of the current frame

inline void **setPosition**(const cv::Point &position) noexcept

Sets the position of the ROI of the output frame.

Private Members

cv::Size **mMaxSize**

class **OutputDefinition**

#include </builds/inivation/dv/dv-runtime/include/dv-sdk/module_io.hpp> Definition of a module output. Every output has a unique (for the module) name, as well as a type. THe type is the 4-character flatbuffers identifier.

Public Functions

inline **OutputDefinition**(const std::string &n, const std::string &t)

Public Members

std::string **name**

std::string **typeName**

class **OutputDefinitionList**

#include </builds/inivation/dv/dv-runtime/include/dv-sdk/module_io.hpp> Vector decorator that exposes convenience functions to add various types of outputs to a module

Public Functions

inline void **addOutput**(const std::string &name, const std::string &typeIdentifier)

Adds an output of a generic type to this module.

Parameters

- **name** – The name of this output
- **typeIdentifier** – The identifier of the FlatBuffers data type used for this input

inline void **addEventOutput**(const std::string &name)

Adds an event output to this module.

Parameters
name – The name of the event output

inline void **addFrameOutput**(const std::string &name)

Adds a frame output to this module.

Parameters
name – The name of the frame output

inline void **addIMUOutput**(const std::string &name)

Adds an IMU output to this module.

Parameters
name – The name of the IMU output

inline void **addTriggerOutput**(const std::string &name)

Adds a trigger output to this module.

Parameters
name – the name of the trigger output

inline void **addBoundingBoxOutput**(const std::string &name)

Adds a bounding box output to this module.

Parameters
name – the name of the bounding box output

inline const std::vector<*OutputDefinition*> &**getOutputs**() const

INTERNAL USE Returns the list of configured outputs

Returns
the list of configured outputs

Private Members

std::vector<*OutputDefinition*> **outputs**

template<typename **T**, typename **U**>

class **OutputVectorDataWrapper** : public *dv*::*_OutputVectorDataWrapperCommon*<*T*, *U*>

#include </builds/inivation/dv/dv-runtime/include/dv-sdk/data/wrappers.hpp> Generic case output vector data wrapper for vector-like data (must have a std::vector<U> called 'elements'). Used for all vector types that do not have a more specific output data wrapper.

Template Parameters

- **T** – The type of the vector output data
- **U** – The type of the vector output data's elements

Public Functions

inline **OutputVectorDataWrapper**(*T* *p, *dvModuleData* m, const std::string &n)

inline *T* &**operator***() noexcept

inline const *T* &**operator***() const noexcept

inline *T* ***operator->**() noexcept

inline const *T* ***operator->**() const noexcept

template<>

class **OutputVectorDataWrapper**<*dv*::BoundingBoxPacket, *dv*::BoundingBox> : public *dv*::*_OutputVectorDataWrapperCommon*<*dv*::BoundingBoxPacket, *dv*::BoundingBox>

Public Functions

inline **OutputVectorDataWrapper**(*dv*::BoundingBoxPacket *p, *dvModuleData* m, const std::string &n)

template<>

class **OutputVectorDataWrapper**<*dv*::EventPacket, *dv*::Event> : public *dv*::*_OutputVectorDataWrapperCommon*<*dv*::EventPacket, *dv*::Event>

Public Functions

inline **OutputVectorDataWrapper**(*dv*::EventPacket *p, *dvModuleData* m, const std::string &n, const cv::Size maxSize)

inline *OutputVectorDataWrapper* &**operator=**(const *_InputDataWrapperCommon*<*dv*::EventPacket> &rhs)

inline void **commit**()

inline *OutputVectorDataWrapper* &**operator<<**(*dv*::*commitType*)

Private Members

cv::Size **mMaxSize**

template<>

class **OutputVectorDataWrapper**<*dv*::IMUPacket, *dv*::IMU> : public *dv*::*_OutputVectorDataWrapperCommon*<*dv*::IMUPacket, *dv*::IMU>

Public Functions

inline **OutputVectorDataWrapper**(IMUPacket *p, *dvModuleData* m, const std::string &n)

template<>

class **OutputVectorDataWrapper**<*dv*::TriggerPacket, *dv*::Trigger> : public *dv*::*_OutputVectorDataWrapperCommon*<*dv*::TriggerPacket, *dv*::Trigger>

Public Functions

inline **OutputVectorDataWrapper**(*dv*::TriggerPacket *p, *dvModuleData* m, const std::string &n)

class **RuntimeConfig**

Public Functions

inline **RuntimeConfig**(*dv*::*Config*::*Node* mn)

inline void **add**(const std::string &key, *ConfigOption* config)

Adds a new configuration option to the set of config options. Configuration options can be generated using the `dv::ConfigOption::...` factory functions. The key is the name of the config option and has to be a name that complies with the specifications of a C variable name.

Parameters

- **key** – The key or name of this config option. Camel case C compliant name
- **config** – The config object. Use `dv::ConfigOption::...` factory functions to generate them

inline void **setPriorityOptions**(const std::vector<std::string> &priorityAttributes)

Sets the priority options for this module. The priority options for the module are the options that are shown to the user in the gui in the sidebar, without having to click on the plus sign. Priority options should be the list of the most important options.

Parameters

priorityAttributes – The list of the keys of the most important configuration options

template<*dv*::*Config*::*AttributeType* **T**>
inline const *dv*::*Config*::*AttributeTypeGenerator*<*T*>::type &**get**(const std::string &key) const

Returns the current value of this configuration option that is stored under the given key. This is the templated generic function, there are overloaded helper functions available for all supported types

Template Parameters

T – The AttributeType type of the attribute

Parameters

key – The name under which this configuration option is stored

Returns

The currently stored value of this configuration option

template<*dv*::*Config*::*AttributeType* **T**>
inline void **set**(const std::string &key, const typename *dv*::*Config*::*AttributeTypeGenerator*<*T*>::type &value, bool force = false)

Sets the configuration option under the given name to the provided value. Requires the attribute type as a template argument. There are non-templated convenience functions available for all supported attribute types. Note that this function is only guaranteed when called within `configUpdate`, calling it outside of configUpdate, the behaviour is undefined.

Template Parameters

T – The type of the attribute

Parameters

- **key** – The name under which this configuration option is stored
- **value** – The value to be stored under the config options
- **force** – Force an update of the variable (useful for statistics variables)

inline void **update**()

inline void **setBool**(const std::string &key, bool value)

Sets the value of a bool config option. Not that this function is only guaranteed when called in the `configUpdate` function. Calling it outside of it can result in undefined behaviour

Parameters

- **key** – The key of the config option

- **value** – The value to be set

inline void **setInt**(const std::string &key, int32_t value)

Sets the value of an int config option. Not that this function is only guaranteed when called in the `configUpdate` function. Calling it outside of it can result in undefined behaviour

Parameters

- **key** – The key of the config option
- **value** – The value to be set

inline void **setLong**(const std::string &key, int64_t value)

Sets the value of a long config option. Not that this function is only guaranteed when called in the `configUpdate` function. Calling it outside of it can result in undefined behaviour

Parameters

- **key** – The key of the config option
- **value** – The value to be set

inline void **setFloat**(const std::string &key, float value)

Sets the value of a float config option. Not that this function is only guaranteed when called in the `configUpdate` function. Calling it outside of it can result in undefined behaviour

Parameters

- **key** – The key of the config option
- **value** – The value to be set

inline void **setDouble**(const std::string &key, double value)

Sets the value of a double config option. Not that this function is only guaranteed when called in the `configUpdate` function. Calling it outside of it can result in undefined behaviour

Parameters

- **key** – The key of the config option
- **value** – The value to be set

inline void **setString**(const std::string &key, const std::string &value)

Sets the value of an int config option. Not that this function is only guaranteed when called in the `configUpdate` function. Calling it outside of it can result in undefined behaviour

Parameters

- **key** – The key of the config option
- **value** – The value to be set

inline bool **getBool**(const std::string &key) const

Returns the value of the config option with the given key. Note that this function will only work if the type of the option is actually bool

Parameters

key – the key of the option to look up

Returns

the value of the option

inline int32_t **getInt**(const std::string &key) const

Returns the value of the config option with the given key. Note that this function will only work if the type of the option is actually int

Parameters
key – the key of the option to look up

Returns
the value of the option

inline int64_t **getLong**(const std::string &key) const

Returns the value of the config option with the given key. Note that this function will only work if the type of the option is actually long

Parameters
key – the key of the option to look up

Returns
the value of the option

inline float **getFloat**(const std::string &key) const

Returns the value of the config option with the given key. Note that this function will only work if the type of the option is actually float

Parameters
key – the key of the option to look up

Returns
the value of the option

inline double **getDouble**(const std::string &key) const

Returns the value of the config option with the given key. Note that this function will only work if the type of the option is actually double

Parameters
key – the key of the option to look up

Returns
the value of the option

inline const std::string &**getString**(const std::string &key) const

Returns the value of the config option with the given key. Note that this function will only work if the type of the option is actually string

Parameters
key – the key of the option to look up

Returns
the value of the option

inline void **set**(const std::string &key, bool value)

inline void **set**(const std::string &key, int32_t value)

inline void **set**(const std::string &key, uint32_t value)

inline void **set**(const std::string &key, int64_t value)

inline void **set**(const std::string &key, uint64_t value)

inline void **set**(const std::string &key, float value)

inline void **set**(const std::string &key, double value)

inline void **set**(const std::string &key, const std::string &value)

inline void **get**(const std::string &key, bool &value) const

inline void **get**(const std::string &key, int32_t &value) const

inline void **get**(const std::string &key, uint32_t &value) const

inline void **get**(const std::string &key, int64_t &value) const

inline void **get**(const std::string &key, uint64_t &value) const

inline void **get**(const std::string &key, float &value) const

inline void **get**(const std::string &key, double &value) const

inline void **get**(const std::string &key, std::string &value) const

Private Members

std::unordered_map<std::string, *ConfigOption*> **configMap**

dv::*Config*::*Node* **moduleNode**

template<typename **T**>

class **RuntimeInput** : public *dv*::*_RuntimeInputCommon*<*T*>

#include </builds/inivation/dv/dv-runtime/include/dv-sdk/data/wrappers.hpp> Describes a generic input at runtime. A generic input can be instantiated for any type. This class basically just inherits from *_RuntimeInputCommon*<T> and does not add any specializations.

Template Parameters
T – The type of the input data

Public Functions

inline **RuntimeInput**(const std::string &name, *dvModuleData* moduleData)

template<>

class **RuntimeInput**<*dv*::Frame> : public *dv*::*_RuntimeInputCommon*<*dv*::Frame>

Public Functions

inline **RuntimeInput**(const std::string &name, *dvModuleData* moduleData)

inline const *InputDataWrapper*<*dv*::Frame> **frame**() const

inline int **sizeX**() const

inline int **sizeY**() const

inline const cv::Size **size**() const

class **RuntimeInputs**

Public Functions

inline **RuntimeInputs**(*dvModuleData* m)

template<typename **T**>
inline const *RuntimeInput*<*T*> **getInput**(const std::string &name) const

Returns the information about the input with the specified name. The type of the input has to be specified as well.

Template Parameters
T – The type of the input

Parameters
name – The name of the input

Returns
An object to access the information about the input

template<typename **T**, typename **U**>
inline const *RuntimeVectorInput*<*T*, *U*> **getVectorInput**(const std::string &name) const

Returns the information about the vector input with the specified name. The type of the vector input has to be specified as well.

Template Parameters

- **T** – The type of the vector input
- **U** – The type of the vector input's elements

Parameters
name – The name of the input

Returns
An object to access the information about the input

inline const *RuntimeVectorInput*<*dv*::EventPacket, *dv*::Event> **getEventInput**(const std::string &name) const

(Convenience) Function to get an event input

Parameters
name – the name of the event input stream

Returns
An object to access information about the input stream

inline const *RuntimeInput*<*dv*::Frame> **getFrameInput**(const std::string &name) const

(Convenience) Function to get an frame input

Parameters
name – the name of the frame input stream

Returns
An object to access information about the input stream

inline const *RuntimeVectorInput*<*dv*::IMUPacket, *dv*::IMU> **getIMUInput**(const std::string &name) const

(Convenience) Function to get an IMU input

Parameters
name – the name of the IMU input stream

Returns
An object to access information about the input stream

inline const *RuntimeVectorInput*<*dv*::TriggerPacket, *dv*::Trigger> **getTriggerInput**(const std::string &name) const

(Convenience) Function to get an trigger input

Parameters
name – the name of the trigger input stream

Returns
An object to access information about the input stream

inline const *RuntimeVectorInput*<*dv*::BoundingBoxPacket, *dv*::BoundingBox> **getBoundingBoxInput**(const std::string &name) const

(Convenience) Function to get an boundingBox input

Parameters
name – the name of the boundingBox input stream

Returns
An object to access information about the input stream

inline const *dv*::*Config*::*Node* **infoNode**(const std::string &name) const

Returns an info node about the specified input. Can be used to determine dimensions of an input/output

Returns
A node that contains the specified inputs information, such as "sizeX" or "sizeY"

inline bool **isConnected**(const std::string &name) const

Returns true, if this optional input is actually connected to an output of another module

Returns
true, if this input is connected

Private Members

dvModuleData **moduleData**

template<typename **T**>

class **RuntimeOutput** : public *dv*::*_RuntimeOutputCommon*<*T*>

#include </builds/inivation/dv/dv-runtime/include/dv-sdk/data/wrappers.hpp> Class that describes an output of a generic type at runtime. Can be used to obtain information about the output, as well as getting a new output object to send data to.

Template Parameters
T – The type of the output data

Public Functions

inline **RuntimeOutput**(const std::string &name, *dvModuleData* moduleData)

template<>

class **RuntimeOutput**<*dv*::Frame> : public *dv*::*_RuntimeOutputCommon*<*dv*::Frame>

#include </builds/inivation/dv/dv-runtime/include/dv-sdk/data/frame.hpp> Specialization of the runtime output for frame outputs Provides convenience setup functions for setting up the frame output

Public Functions

inline **RuntimeOutput**(const std::string &name, *dvModuleData* moduleData)

inline *OutputDataWrapper*<*dv*::Frame> **data**()

Returns a writeable output wrapper for the given type of this output. Allocates new output memory if necessary. The output can be committed by calling commit on the returned object.

Returns
A wrapper to allocated output memory to write to

inline *OutputDataWrapper*<*dv*::Frame> **frame**()

Returns the current output frame to set up values and to get the back data buffer

Returns
The output frame to set up

void **setup**(const std::string &originDescription) = delete

inline void **setup**(int sizeX, int sizeY, const std::string &originDescription)

Sets up this frame output with the provided parameters

Parameters

- **sizeX** – The width of the frames supplied on this output
- **sizeY** – The height of the frames supplied on this output
- **originDescription** – A description of the original creator of the data

template<typename **U**>
inline void **setup**(const *RuntimeInput*<*U*> &input)

Sets this frame output up with the same parameters the the supplied input.

Parameters
input – An input to copy the data from

template<typename **U**, typename **TT**>
inline void **setup**(const *RuntimeVectorInput*<*U*, *TT*> &input)

Sets this frame output up with the same parameters as the supplied vector input.

Parameters
input – A vector input to copy the information from

inline int **sizeX**() const

inline int **sizeY**() const

inline const cv::Size **size**() const

inline void **operator<<**(*dv*::*commitType*)

Convenience shortcut to commit the data on this output directly

inline *RuntimeOutput*<*dv*::Frame> &**operator<<**(int64_t timestamp)

Sets the timestamp of the current frame on the output (start of exposure or closest).

Parameters
timestamp – The timestamp to be set

Returns
A reference to this output, to commit more data

inline *RuntimeOutput*<*dv*::Frame> &**operator<<**(const cv::Mat &mat)

Convenience shorthand to commit an OpenCV mat onto this output. If not using this function, call `data()` to get an output frame to fill into.

Parameters
mat – The OpenCV Mat to submit

Returns
A reference to the this

inline *RuntimeOutput*<*dv*::Frame> &**operator<<**(cv::Mat &&mat)

Convenience shorthand to commit an OpenCV mat onto this output. If not using this function, call `data()` to get an output frame to fill into.

Parameters
mat – The OpenCV Mat to submit

Returns
A reference to the this

class **RuntimeOutputs**

Public Functions

inline **RuntimeOutputs**(*dvModuleData* m)

template<typename **T**>
inline *RuntimeOutput*<*T*> **getOutput**(const std::string &name)

Function to get an output

Parameters
name – the name of the output stream

Returns
An object to access the modules output

template<typename **T**, typename **U**>
inline *RuntimeVectorOutput*<*T*, *U*> **getVectorOutput**(const std::string &name)

Function to get a vector output

Parameters
name – the name of the output stream

Returns
An object to access the modules output

inline *RuntimeVectorOutput*<*dv*::EventPacket, *dv*::Event> **getEventOutput**(const std::string &name)

(Convenience) Function to get an event output

Parameters
name – the name of the event output stream

Returns
An object to access the modules output

inline *RuntimeOutput*<*dv*::Frame> **getFrameOutput**(const std::string &name)

(Convenience) Function to get an frame output

Parameters
name – the name of the frame output stream

Returns
An object to access the modules output

inline *RuntimeVectorOutput*<*dv*::IMUPacket, *dv*::IMU> **getIMUOutput**(const std::string &name)

(Convenience) Function to get an imu output

Parameters
name – the name of the imu output stream

Returns
An object to access the modules output

inline *RuntimeVectorOutput*<*dv*::TriggerPacket, *dv*::Trigger> **getTriggerOutput**(const std::string &name)

(Convenience) Function to get an trigger output

Parameters
name – the name of the trigger output stream

Returns
An object to access the modules output

inline *RuntimeVectorOutput*<*dv*::BoundingBoxPacket, *dv*::BoundingBox> **getBoundingBoxOutput**(const std::string &name)

(Convenience) Function to get a bounding box output

Parameters
name – the name of the trigger output stream

Returns
An object to access the modules output

inline *dv*::*Config*::*Node* **infoNode**(const std::string &name)

Returns an info node about the specified output, can be used toset output information.

Returns
A node that can contain output information, such as "sizeX" or "sizeY"

Private Members

dvModuleData **moduleData**

template<typename **T**, typename **U**>

class **RuntimeVectorInput** : public *dv*::*_RuntimeVectorInputCommon*<*T*, *U*>

#include </builds/inivation/dv/dv-runtime/include/dv-sdk/data/wrappers.hpp> Describes a generic vector input at runtime. A generic vecotr input can be instantiated for any vector type (a type with a member called 'elements' of type std::vector<U>). This class basically just inherits from *_RuntimeInputCommon*<T> and does not add any specializations.

Template Parameters

- **T** – The type of the vector input data
- **U** – The type of the vector input data's elements

Public Functions

inline **RuntimeVectorInput**(const std::string &name, *dvModuleData* moduleData)

template<>

class **RuntimeVectorInput**<*dv*::BoundingBoxPacket, *dv*::BoundingBox> : public *dv*::*_RuntimeVectorInputCommon*<*dv*::BoundingBoxPacket, *dv*::BoundingBox>

#include </builds/inivation/dv/dv-runtime/include/dv-sdk/data/bounding_box.hpp> Describes an input for bounding boxes. Offers convenience functions to obtain informations about the bounding box sizes as well as to get the data.

Public Functions

inline **RuntimeVectorInput**(const std::string &name, *dvModuleData* moduleData)

inline int **sizeX**() const

Returns
The width of the input region in pixels. Any event on this input will have a x-coordinate smaller than the return value of this function.

inline int **sizeY**() const

Returns
The height of the input region in pixels. Any event on this input will have a y-coordinate smaller than the return value of this function

inline const cv::Size **size**() const

Returns
the input region size in pixels as an OpenCV size object

template<>

class **RuntimeVectorInput**<*dv*::EventPacket, *dv*::Event> : public *dv*::*_RuntimeVectorInputCommon*<*dv*::EventPacket, *dv*::Event>

#include </builds/inivation/dv/dv-runtime/include/dv-sdk/data/event.hpp> Describes an input for event packets. Offers convenience functions to obtain informations about the event input as well as to get the event data.

Public Functions

inline **RuntimeVectorInput**(const std::string &name, *dvModuleData* moduleData)

inline const *InputVectorDataWrapper*<*dv*::EventPacket, *dv*::Event> **events**() const

Returns an iterable container of the latest events that arrived at this input.

Returns
An iterable container of the newest events.

inline int **sizeX**() const

Returns
The width of the input region in pixels. Any event on this input will have a x-coordinate smaller than the return value of this function.

inline int **sizeY**() const

Returns
The height of the input region in pixels. Any event on this input will have a y-coordinate smaller than the return value of this function

inline *dv*::EventColor **colorForEvent**(const *dv*::Event &event) const

Determine the color of the Bayer color filter for a specific event, based on its address. Please take into account that there are usually twice as many green pixels as there are red or blue ones.

Parameters
event – event to determine filter color for.

Returns
filter color.

inline cv::Size **size**() const

Returns
the input region size in pixels as an OpenCV size object

Private Members

dv::PixelArrangement **mPixelArrangement** = {*dv*::PixelArrangement::MONO}

template<typename **T**, typename **U**>

class **RuntimeVectorOutput** : public *dv*::*_RuntimeVectorOutputCommon*<*T*, *U*>

#include </builds/inivation/dv/dv-runtime/include/dv-sdk/data/wrappers.hpp> Class that describes an output of a generic vector type at runtime. A generic vector input can be instantiated for any vector type (a type with a member called 'elements' of type std::vector<U>). Can be used to obtain information about the output, as well as getting a new output object to send data to.

Template Parameters

- **T** – The type of the vector output data
- **U** – The type of the vector output data's elements

Public Functions

inline **RuntimeVectorOutput**(const std::string &name, *dvModuleData* moduleData)

template<>

class **RuntimeVectorOutput**<*dv*::BoundingBoxPacket, *dv*::BoundingBox> : public *dv*::*_RuntimeVectorOutputCommon*<*dv*::BoundingBoxPacket, *dv*::BoundingBox>

#include </builds/inivation/dv/dv-runtime/include/dv-sdk/data/bounding_box.hpp> Specialization of the runtime output for bounding boxes. Provides convenience setup functions for setting up the bounding boxes output.

Public Functions

inline **RuntimeVectorOutput**(const std::string &name, *dvModuleData* moduleData)

void **setup**(const std::string &originDescription) = delete

inline void **setup**(int sizeX, int sizeY, const std::string &originDescription)

Sets up this bounding boxes output by setting the provided arguments to the output info node

Parameters

- **sizeX** – The width of this output
- **sizeY** – The height of this output
- **originDescription** – A description that describes the original generator of the data

template<typename **U**>
inline void **setup**(const *RuntimeInput*<*U*> &input)

Sets this bounding box output up with the same parameters as the supplied input.

Parameters
input – An input to copy the information from

template<typename **U**, typename **TT**>
inline void **setup**(const *RuntimeVectorInput*<*U*, *TT*> &input)

Sets this bounding box output up with the same parameters as the supplied vector input.

Parameters
input – A vector input to copy the information from

inline int **sizeX**() const

Returns
The width of the input region in pixels. Any event on this input will have a x-coordinate smaller than the return value of this function.

inline int **sizeY**() const

Returns
The height of the input region in pixels. Any event on this input will have a y-coordinate smaller than the return value of this function

inline const cv::Size **size**() const

Returns
the input region size in pixels as an OpenCV size object

template<>

class **RuntimeVectorOutput**<*dv*::EventPacket, *dv*::Event> : public *dv*::*_RuntimeVectorOutputCommon*<*dv*::EventPacket, *dv*::Event>

#include </builds/inivation/dv/dv-runtime/include/dv-sdk/data/event.hpp> Specialization of the runtime output for event outputs. Provides convenience setup functions for setting up the event output

Public Functions

inline **RuntimeVectorOutput**(const std::string &name, *dvModuleData* moduleData)

inline *OutputVectorDataWrapper*<*dv*::EventPacket, *dv*::Event> **data**()

Returns a writeable output wrapper for the given type of this output. Allocates new output memory if necessary. The output can be committed by calling commit on the returned object.

Returns
A wrapper to allocated output memory to write to

inline *OutputVectorDataWrapper*<*dv*::EventPacket, *dv*::Event> **events**()

void **setup**(const std::string &originDescription) = delete

inline void **setup**(int sizeX, int sizeY, const std::string &originDescription)

Sets up this event output by setting the provided arguments to the output info node

Parameters

- **sizeX** – The width of this event output
- **sizeY** – The height of this event output

- **originDescription** – A description that describes the original generator of the data

template<typename **U**>
inline void **setup**(const *RuntimeInput*<*U*> &input)

Sets this event output up with the same parameters as the supplied input.

Parameters
input – An input to copy the information from

template<typename **U**, typename **TT**>
inline void **setup**(const *RuntimeVectorInput*<*U*, *TT*> &input)

Sets this event output up with the same parameters as the supplied vector input.

Parameters
input – A vector input to copy the information from

inline int **sizeX**() const

Returns
The width of the input region in pixels. Any event on this input will have a x-coordinate smaller than the return value of this function.

inline int **sizeY**() const

Returns
The height of the input region in pixels. Any event on this input will have a y-coordinate smaller than the return value of this function

inline cv::Size **size**() const

Returns
the input region size in pixels as an OpenCV size object

template<typename **K**>
inline *RuntimeVectorOutput*<*dv*::EventPacket, *dv*::Event> &**operator<<**(const *K* &rhs)

Convenience shortcut to commit the data on this output directly This gets resolved to one of the various << functions of the data wrapper.

inline void **operator<<**(*commitType*)

Convenience shortcut to commit the data on this output directly

template<typename **T**>

class **Stats**

#include </builds/inivation/dv/dv-runtime/include/dv-sdk/stats.hpp> Class which provides the functionality to measure statistics for numeric data types. The following statistics are computed and published, along with the most recently added value itself:

- mean
- variance
- minimum
- maximum
- sample count

Public Functions

Stats() = delete

Deleted default constructor

inline **Stats**(const *dv*::*Config*::*Node* node, const std::string_view name)

Constructor which selects the default window size

Creates a node with the name `name` relative to `node`. The stats will be published to the attributes `current`, `mean`, `var`, `min`, `max` and `count` at this node.

See also

DEFAULT_WINDOW_SIZE for the rolling *mean* and variance.

Parameters

- **node** – Parent node relative to which the node for the measured stats should be published
- **name** – Name of the node

inline **Stats**(const uint32_t windowSize, const *dv*::*Config*::*Node* node, const std::string_view name)

Constructor which applies a custom window size for the rolling mean and variance.

Creates a node with the name `name` relative to `node`. The stats will be published to the attributes `current`, `mean`, `var`, `min`, `max` and `count` at this node.

Parameters

- **windowSize** – size of the accumulator window
- **node** – Parent node relative to which the node for the measured stats should be published
- **name** – Name of the node

Stats(const *Stats* &other) = delete

Copy - Disallow: since node and name are the same, you'd suddenly have two objects feeding data into the configuration backend and overwriting each other. This is not what you want.

Stats &**operator=**(const *Stats* &rhs) = delete

Stats(*Stats* &&other) = default

Move

Stats &**operator=**(*Stats* &&rhs) = default

virtual **~Stats**() noexcept = default

Destructor

inline void **operator()**(const *T* value) noexcept

Adds a new value to the underlying accumulator

inline auto **currentSample**() const noexcept

Returns the most recently added sample

inline auto **mean**() const noexcept

Returns the rolling mean

inline auto **var**() const noexcept
: Returns the rolling variance

inline auto **min**() const noexcept
: Returns the minimum value

inline auto **max**() const noexcept
: Returns the maximum value

inline auto **count**() const noexcept
: Returns the number of added samples

Protected Types

template<typename ...**ARGS**>

using **accumulator_set** = boost::accumulators::accumulator_set<*ARGS*...>

template<typename ...**ARGS**>

using **stats** = boost::accumulators::stats<*ARGS*...>

using **tag_lazy_rolling_mean** = boost::accumulators::tag::lazy_rolling_mean

using **tag_lazy_rolling_variance** = boost::accumulators::tag::lazy_rolling_variance

using **tag_min** = boost::accumulators::tag::min

using **tag_max** = boost::accumulators::tag::max

using **tag_count** = boost::accumulators::tag::count

using **accumulator** = *accumulator_set*<*T*, *stats*<*tag_lazy_rolling_mean*, *tag_lazy_rolling_variance*, *tag_min*, *tag_max*, *tag_count*>>

Protected Attributes

accumulator **mAccumulator** = {boost::accumulators::tag::rolling_window::window_size = *DEFAULT_WINDOW_SIZE*}

T **mCurrent** = {}

uint32_t **mWindowSize** = {*DEFAULT_WINDOW_SIZE*}

dv::*_RateLimiter* **mRateLimiter** = {1, *PUBLISHING_RATE_MS*}

std::string **mName**

dv::*Config*::*Node* **mNode**

Protected Static Attributes

static constexpr uint32_t **DEFAULT_WINDOW_SIZE** = {500}

static constexpr uint32_t **PUBLISHING_RATE_MS** = {300}

Private Functions

inline void **init**()

Initializes the underlying accumulators to zero, and creates the node attributes to which data will be published.

inline void **publish**() noexcept

Publishes the statistical data to the node attributes

class **TCPTLSSocket**

Public Functions

inline **TCPTLSSocket**(*asioTCP*::socket s, bool tlsEnabled, asioSSL::context *tlsContext)

inline **~TCPTLSSocket**()

inline void **close**()

template<typename **StartupHandler**>
inline void **start**(*StartupHandler* &&stHandler, asioSSL::stream_base::handshake_type type)

Startup handler needs following signature: void (const boost::system::error_code &)

template<typename **WriteHandler**>
inline void **write**(const asio::const_buffer &buf, *WriteHandler* &&wrHandler)

Write handler needs following signature: void (const boost::system::error_code &, size_t)

template<typename **ReadHandler**>
inline void **read**(const asio::mutable_buffer &buf, *ReadHandler* &&rdHandler)

Read handler needs following signature: void (const boost::system::error_code &, size_t)

inline *asioTCP*::endpoint **local_endpoint**() const

inline asioIP::address **local_address**() const

inline uint16_t **local_port**() const

inline *asioTCP*::endpoint **remote_endpoint**() const

inline asioIP::address **remote_address**() const

inline uint16_t **remote_port**() const

Private Functions

inline *asioTCP*::socket &**baseSocket**()

Private Members

asioTCP::endpoint **localEndpoint**

asioTCP::endpoint **remoteEndpoint**

asioSSL::stream<*asioTCP*::socket> **socket**

bool **socketClosed**

bool **secureConnection**

class **Throughput** : public *dv*::*statistics*::*Stats*<float>

#include </builds/inivation/dv/dv-runtime/include/dv-sdk/stats.hpp> Class which provides the functionality to measure statistics for throughput related data.

Public Functions

Throughput() = delete

Deleted default constructor

inline **Throughput**(const *dv*::*Config*::*Node* node, const std::string_view name)

Constructor which selects the default measurement interval

The data will be published to the node `node/name/throughput/` if a name is specified, or `node/throughput/ if none is specified.

See also

DEFAULT_MEASUREMENT_INTERVAL and window size for the underlying statistics.

See also

Stats.

See also

Stats

Parameters

- **node** – Parent node relative to which the node for the measured stats should be published
- **name** – Name of the node

inline **Throughput**(const std::chrono::microseconds measurementInterval, const *dv*::*Config*::*Node* node, const std::string_view name)

Constructor which applies a custom measurement interval and the default window size for the underlying statistics.

The data will be published to the node `node/name/throughput/` if a name is specified, or `node/throughput/ if none is specified.

See also

Stats.

See also

Stats

Parameters

- **measurementInterval** – The measurement interval with which the throughput shall be evaluated.
- **node** – Parent node relative to which the node for the measured stats should be published
- **name** – Name of the node

inline **Throughput**(const uint32_t windowSize, const std::chrono::microseconds measurementInterval, const *dv*::*Config*::*Node* node, const std::string_view name)

Constructor which applies a custom measurement interval and a custom window size for the underlying statistics.

The data will be published to the node `node/name/throughput/` if a name is specified, or `node/throughput/ if none is specified.

See also

Stats.

See also

Stats

Parameters

- **windowSize** – the window size for the statistical accumulators. @See *Stats*
- **measurementInterval** – The measurement interval with which the throughput shall be evaluated.
- **node** – Parent node relative to which the node for the measured stats should be published
- **name** – Name of the node

Throughput(const *Throughput* &other) = delete

Copy - Disallow: since node and name are the same, you'd suddenly have two objects feeding data into the configuration backend and overwriting each other. This is not what you want.

Throughput &**operator=**(const *Throughput* &rhs) = delete

Throughput(*Throughput* &&other) = default

Move

Throughput &**operator=**(*Throughput* &&rhs) = default

~Throughput() noexcept = default

Destructor

inline void **add**(const uint64_t elements) noexcept

Updates the statistics by adding elements to the number of elements already received within this interval.

If an interval has finished since we previously called this method, the throughput is computed and added to the underlying statistical accumulators.

See also

Stats.

Parameters

elements – The number of elements to be added to the throughput computation

Private Members

std::chrono::time_point<std::chrono::steady_clock> **mStartTime** = {std::chrono::steady_clock::now()}

uint64_t **mNumElements** = {0}

std::chrono::microseconds **mMeasurementInterval** = {*DEFAULT_MEASUREMENT_INTERVAL*}

Private Static Attributes

static constexpr std::chrono::duration **DEFAULT_MEASUREMENT_INTERVAL** = {std::chrono::milliseconds(1)}

class **Tree**

Public Functions

inline **Tree**(*dvConfigTree* t)

inline explicit **operator** ***dvConfigTree***() const

inline void **deleteTree**()

inline bool **existsNode**(const std::string &nodePath) const

inline *Node* **getRootNode**() const

inline *Node* **getNode**(const std::string &nodePath) const

This returns a reference to a node, and as such must be carefully mediated with any dvConfigNodeRemoveNode() calls.

Throws

`std::out_of_range` – Invalid absolute node path.

inline void **attributeUpdaterRemoveAll**()

inline bool **attributeUpdaterRun**()

inline void **globalNodeListenerSet**(*dvConfigNodeChangeListener* node_changed, void *userData)

Listener must be able to deal with userData being NULL at any moment. This can happen due to concurrent changes from this setter.

inline void **globalAttributeListenerSet**(*dvConfigAttributeChangeListener* attribute_changed, void *userData)

Listener must be able to deal with userData being NULL at any moment. This can happen due to concurrent changes from this setter.

Public Static Functions

static inline *Tree* **globalTree**()

static inline *Tree* **newTree**()

static inline void **errorLogCallbackSet**(*dvConfigTreeErrorLogCallback* error_log_cb)

static inline *dvConfigTreeErrorLogCallback* **errorLogCallbackGet**()

Private Members

dvConfigTree **tree**

template<class **T**>

class **vectorConstProxy**

Subclassed by *dv::_InputVectorDataWrapperCommon< dv::BoundingBoxPacket, dv::BoundingBox >*, *dv::_InputVectorDataWrapperCommon< dv::EventPacket, dv::Event >*, *dv::_InputVectorDataWrapperCommon< dv::IMUPacket, dv::IMU >*, *dv::_InputVectorDataWrapperCommon< dv::TriggerPacket, dv::Trigger >*, *dv::vectorProxy< dv::BoundingBox >*, *dv::vectorProxy< dv::Event >*, *dv::vectorProxy< dv::IMU >*, *dv::vectorProxy< dv::Trigger >*, *dv::vectorProxy< U >*, *dv::vectorProxy< T >*

Public Types

using **value_type** = typename std::vector<*T*>::value_type

using **pointer** = typename std::vector<*T*>::pointer

using **const_pointer** = typename std::vector<*T*>::const_pointer

using **reference** = typename std::vector<*T*>::reference

using **const_reference** = typename std::vector<*T*>::const_reference

using **size_type** = typename std::vector<*T*>::size_type

using **difference_type** = typename std::vector<*T*>::difference_type

using **iterator** = typename std::vector<*T*>::iterator

using **const_iterator** = typename std::vector<*T*>::const_iterator

using **reverse_iterator** = typename std::vector<*T*>::reverse_iterator

using **const_reverse_iterator** = typename std::vector<*T*>::const_reverse_iterator

Public Functions

inline **vectorConstProxy**(const std::vector<*T*> *vec)

~vectorConstProxy() = default

inline **vectorConstProxy**(const *vectorConstProxy* &vec)

vectorConstProxy(*vectorConstProxy* &&vec) = delete

vectorConstProxy &**operator=**(const *vectorConstProxy* &rhs) = delete

vectorConstProxy &**operator=**(*vectorConstProxy* &&rhs) = delete

inline bool **operator==**(const *vectorConstProxy* &rhs) const noexcept

inline bool **operator!=**(const *vectorConstProxy* &rhs) const noexcept

inline bool **operator<**(const *vectorConstProxy* &rhs) const noexcept

inline bool **operator>**(const *vectorConstProxy* &rhs) const noexcept

inline bool **operator<=**(const *vectorConstProxy* &rhs) const noexcept

inline bool **operator>=**(const *vectorConstProxy* &rhs) const noexcept

inline bool **operator==**(const std::vector<*value_type*> &rhs) const noexcept

inline bool **operator!=**(const std::vector<*value_type*> &rhs) const noexcept

inline bool **operator<**(const std::vector<*value_type*> &rhs) const noexcept

inline bool **operator>**(const std::vector<*value_type*> &rhs) const noexcept

inline bool **operator<=**(const std::vector<*value_type*> &rhs) const noexcept

inline bool **operator>=**(const std::vector<*value_type*> &rhs) const noexcept

inline *const_pointer* **data**() const noexcept

inline *size_type* **size**() const noexcept

inline *size_type* **capacity**() const noexcept

inline *size_type* **max_size**() const noexcept

inline bool **empty**() const noexcept

template<typename **INT**>
inline *const_reference* **operator[]**(const *INT* index) const

template<typename **INT**>
inline *const_reference* **at**(const *INT* index) const

inline explicit **operator std::vector<*value_type*>**() const

inline *const_reference* **front**() const

inline *const_reference* **back**() const

template<typename **U**>
inline bool **contains**(const *U* &item) const

template<typename **Pred**>
inline bool **containsIf**(*Pred* predicate) const

inline *const_iterator* **begin**() const noexcept

inline *const_iterator* **end**() const noexcept

inline *const_iterator* **cbegin**() const noexcept

inline *const_iterator* **cend**() const noexcept

inline *const_reverse_iterator* **rbegin**() const noexcept

inline *const_reverse_iterator* **rend**() const noexcept

inline *const_reverse_iterator* **crbegin**() const noexcept

inline *const_reverse_iterator* **crend**() const noexcept

inline std::vector<*value_type*> **operator+**(const *vectorConstProxy* &rhs) const

inline std::vector<*value_type*> **operator+**(const std::vector<*value_type*> &rhs) const

inline std::vector<*value_type*> **operator+**(*const_reference* value) const

inline std::vector<*value_type*> **operator+**(std::initializer_list<*value_type*> rhs_list) const

Public Static Attributes

static constexpr *size_type* **npos** = {std::vector<*T*>::npos}

Protected Attributes

std::vector<*T*> ***mVectorPtr**

Private Static Attributes

static std::vector<*T*> **EMPTY_VECTOR** = {}

Friends

inline friend bool **operator==**(const std::vector<*value_type*> &lhs, const *vectorConstProxy* &rhs) noexcept

inline friend bool **operator!=**(const std::vector<*value_type*> &lhs, const *vectorConstProxy* &rhs) noexcept

inline friend bool **operator<**(const std::vector<*value_type*> &lhs, const *vectorConstProxy* &rhs) noexcept

inline friend bool **operator>**(const std::vector<*value_type*> &lhs, const *vectorConstProxy* &rhs) noexcept

inline friend bool **operator<=**(const std::vector<*value_type*> &lhs, const *vectorConstProxy* &rhs) noexcept

inline friend bool **operator>=**(const std::vector<*value_type*> &lhs, const *vectorConstProxy* &rhs) noexcept

inline friend std::vector<*value_type*> **operator+**(const std::vector<*value_type*> &lhs, const *vectorConstProxy* &rhs)

inline friend std::vector<*value_type*> **operator+**(*const_reference* value, const *vectorConstProxy* &rhs)

inline friend std::vector<*value_type*> **operator+**(std::initializer_list<*value_type*> lhs_list, const *vectorConstProxy* &rhs)

template<class **T**>

class **vectorProxy** : public *dv*::*vectorConstProxy*<*T*>

Subclassed by *dv::_OutputVectorDataWrapperCommon< dv::BoundingBoxPacket, dv::BoundingBox >*, *dv::_OutputVectorDataWrapperCommon< dv::EventPacket, dv::Event >*, *dv::_OutputVectorDataWrapperCommon< dv::IMUPacket, dv::IMU >*, *dv::_OutputVectorDataWrapperCommon< dv::TriggerPacket, dv::Trigger >*

Public Types

using **value_type** = typename std::vector<*T*>::value_type

using **pointer** = typename std::vector<*T*>::pointer

using **const_pointer** = typename std::vector<*T*>::const_pointer

using **reference** = typename std::vector<*T*>::reference

using **const_reference** = typename std::vector<*T*>::const_reference

using **size_type** = typename std::vector<*T*>::size_type

using **difference_type** = typename std::vector<*T*>::difference_type

using **iterator** = typename std::vector<*T*>::iterator

using **const_iterator** = typename std::vector<*T*>::const_iterator

using **reverse_iterator** = typename std::vector<*T*>::reverse_iterator

using **const_reverse_iterator** = typename std::vector<*T*>::const_reverse_iterator

Public Functions

inline **vectorProxy**(std::vector<*T*> *vec)

~vectorProxy() = default

inline void **reassign**(std::vector<*T*> *vec)

inline **vectorProxy**(const *vectorProxy* &vec)

vectorProxy(*vectorProxy* &&vec) = delete

vectorProxy &**operator=**(*vectorProxy* &&rhs) = delete

inline *vectorProxy* &**operator=**(const *vectorProxy* &rhs)

inline *vectorProxy* &**operator=**(const std::vector<*value_type*> &rhs)

inline *vectorProxy* &**operator=**(*const_reference* value)

inline *vectorProxy* &**operator=**(std::initializer_list<*value_type*> rhs_list)

inline *vectorProxy* &**assign**(const *vectorProxy* &vec, const *size_type* pos = 0, const *size_type* count = *npos*)

inline *vectorProxy* &**assign**(const std::vector<*value_type*> &vec, const *size_type* pos = 0, const *size_type* count = *npos*)

inline *vectorProxy* &**assign**(*const_pointer* vec, const *size_type* vecLength, const *size_type* pos = 0, const *size_type* count = *npos*)

inline *vectorProxy* &**assign**(*const_reference* value)

inline *vectorProxy* &**assign**(const *size_type* count, *const_reference* value)

template<typename **InputIt**, std::enable_if_t<std::is_base_of_v<std::input_iterator_tag, typename std::iterator_traits<*InputIt*>::iterator_category>, bool> = true>
inline *vectorProxy* &**assign**(*InputIt* first, *InputIt* last)

inline *vectorProxy* &**assign**(std::initializer_list<*value_type*> init_list)

inline *pointer* **data**() noexcept

inline void **resize**(const *size_type* newSize)

inline void **resize**(const *size_type* newSize, *const_reference* value)

inline void **reserve**(const *size_type* minCapacity)

inline void **shrink_to_fit**()

template<typename **INT**>

inline *reference* **operator[]**(const *INT* index)

template<typename **INT**>
inline *reference* **at**(const *INT* index)

inline *reference* **front**()

inline *reference* **back**()

inline void **push_back**(*const_reference* value)

inline void **push_back**(*value_type* &&value)

template<class ...**Args**>
inline *reference* **emplace_back**(*Args*&&... args)

inline void **pop_back**()

inline void **clear**() noexcept

inline void **swap**(*vectorProxy* &rhs) noexcept

inline void **swap**(std::vector<*value_type*> &rhs) noexcept

inline void **sortUnique**()

template<typename **Compare**>
inline void **sortUnique**(*Compare* comp)

template<typename **U**>
inline *size_type* **remove**(const *U* &item)

template<typename **Pred**>
inline *size_type* **removeIf**(*Pred* predicate)

inline *iterator* **begin**() noexcept

inline *iterator* **end**() noexcept

inline *reverse_iterator* **rbegin**() noexcept

inline *reverse_iterator* **rend**() noexcept

inline *iterator* **insert**(*const_iterator* pos, *const_reference* value)

inline *iterator* **insert**(*const_iterator* pos, *value_type* &&value)

inline *iterator* **insert**(*const_iterator* pos, const *size_type* count, *const_reference* value)

template<typename **InputIt**, std::enable_if_t<std::is_base_of_v<std::input_iterator_tag, typename std::iterator_traits<*InputIt*>::iterator_category>, bool> = true>
inline *iterator* **insert**(*const_iterator* pos, *InputIt* first, *InputIt* last)

inline *iterator* **insert**(*const_iterator* pos, std::initializer_list<*value_type*> init_list)

template<class ...**Args**>
inline *iterator* **emplace**(*const_iterator* pos, *Args*&&... args)

inline *iterator* **erase**(*const_iterator* pos)

inline *iterator* **erase**(*const_iterator* first, *const_iterator* last)

inline *vectorProxy* &**append**(const *vectorProxy* &vec, const *size_type* pos = 0, const *size_type* count = *npos*)

inline *vectorProxy* &**append**(const std::vector<*value_type*> &vec, const *size_type* pos = 0, const *size_type* count = *npos*)

inline *vectorProxy* &**append**(*const_pointer* vec, const *size_type* vecLength, const *size_type* pos = 0, const *size_type* count = *npos*)

inline *vectorProxy* &**append**(*const_reference* value)

inline *vectorProxy* &**append**(const *size_type* count, *const_reference* value)

template<typename **InputIt**, std::enable_if_t<std::is_base_of_v<std::input_iterator_tag, typename std::iterator_traits<*InputIt*>::iterator_category>, bool> = true>
inline *vectorProxy* &**append**(*InputIt* first, *InputIt* last)

inline *vectorProxy* &**append**(std::initializer_list<*value_type*> init_list)

inline *vectorProxy* &**operator+=**(const *vectorProxy* &rhs)

inline *vectorProxy* &**operator+=**(const std::vector<*value_type*> &rhs)

inline *vectorProxy* &**operator+=**(*const_reference* value)

inline *vectorProxy* &**operator+=**(std::initializer_list<*value_type*> rhs_list)

Public Static Attributes

static constexpr *size_type* **npos** = {std::vector<*T*>::npos}

template<class **T**>

class **WriteOrderedSocket** : public *T*

Public Functions

template<typename ...**ARGS**>
inline **WriteOrderedSocket**(*ARGS*&&... args)

template<typename **WriteHandler**>
inline void **write**(const asio::const_buffer &buf, *WriteHandler* &&wrHandler)

Write handler needs following signature: void (const boost::system::error_code &, size_t)

Private Members

std::deque<std::pair<asio::const_buffer, std::function<void(const boost::system::error_code&, size_t)>>> **writeQueue**

namespace **dv**

Header file that includes the C++ wrapper for the DV logging facilities.

Enums

enum class **FileDialogMode**

Different opening modes for a File Dialog config option.

Values:

enumerator **OPEN**

enumerator **SAVE**

enumerator **DIRECTORY**

enum class **_StringAttributeType**

INTERNAL: select between different string types.

Values:

enumerator **NORMAL**

enumerator **LIST**

enumerator **FILE**

enum class **_BooleanAttributeType**

INTERNAL: select between different boolean types

Values:

enumerator **CHECKBOX**

enumerator **BUTTON**

Functions

template<class **T**>
inline *T* **sgn**(*T* x)

Returns the sign of the given number as -1 or 1. Returns 1 for 0.

Template Parameters
T – The data type of the number and return value

Parameters
x – the data to be checked

Returns
-1 iff x < 0, 1 otherwise

Variables

constexpr *commitType* **commit** = {}

constexpr *logEndType* **logEnd** = {}

template<typename **T**>

constexpr bool **has_initDescription** = has_static_member_function_initDescription<*T*, const char*>::value

Trait for the existence of a static initDescription method with const char* return value

Template Parameters
T – The class to be tested

template<typename **T**>

constexpr bool **has_initConfigOptions** = has_static_member_function_initConfigOptions<*T*, void(*RuntimeConfig*&)>::value

Trait for the existence of a static initConfigOptions method with map argument

Template Parameters
T – The class to be tested

template<typename **T**>

constexpr bool **has_advancedStaticInit** = has_static_member_function_advancedStaticInit<*T*, void(*dvModuleData*)>::value

template<typename **T**>

constexpr bool **has_initTypes** = has_static_member_function_initTypes<*T*, void(std::vector<*dv*::Types::Type>&)>::value

template<typename **T**>

constexpr bool **has_initInputs** = has_static_member_function_initInputs<*T*, void(*dv*::*InputDefinitionList*&)>::value

template<typename **T**>

constexpr bool **has_initOutputs** = has_static_member_function_initOutputs<*T*, void(*dv*::*OutputDefinitionList*&)>::value

namespace **Config**

Enums

enum class **AttributeType**

Values:

enumerator **UNKNOWN**

enumerator **BOOL**

enumerator **INT**

enumerator **LONG**

enumerator **FLOAT**

enumerator **DOUBLE**

enumerator **STRING**

enum class **AttributeFlags** : uint32_t

Values:

enumerator **NORMAL**

enumerator **READ_ONLY**

enumerator **NO_EXPORT**

enumerator **IMPORTED**

Functions

inline *AttributeFlags* **operator|**(*AttributeFlags* lhs, *AttributeFlags* rhs)

inline *AttributeFlags* &**operator|=**(*AttributeFlags* &lhs, *AttributeFlags* rhs)

inline int **getCFlags**(*AttributeFlags* f)

Variables

static *Tree* **GLOBAL** = *Tree*::*globalTree*()

namespace **runtime**

Variables

static constexpr int **VERSION_MAJOR** = {1}

static constexpr int **VERSION_MINOR** = {7}

static constexpr int **VERSION_PATCH** = {0}

static constexpr int **VERSION** = {((1 * 10000) + (7 * 100) + 0)}

static constexpr std::string_view **NAME_STRING** = {"dv-runtime"}

static constexpr std::string_view **VERSION_STRING** = {"1.7.0"}

namespace **statistics**

file **config.hpp**

#include "cross/portable_io.h"#include "utils.h"#include <boost/algorithm/string/join.hpp>#include <chrono>#include <cmath>#include <string>#include <unordered_map>#include <vector>

Defines

DV_MODULE_MULTI_THREAD_SAFE

file **dvConfig.h**

#include <errno.h>#include <inttypes.h>#include <stdbool.h>#include <stdint.h>#include <stdlib.h>#include "dv-sdk/api_visibility.h"

Typedefs

typedef struct dv_config_node ***dvConfigNode**

typedef const struct dv_config_node ***dvConfigNodeConst**

typedef void (***dvConfigNodeChangeListener**)(*dvConfigNode* node, void *userData, enum *dvConfigNodeEvents* event, const char *changeNode)

typedef void (***dvConfigAttributeChangeListener**)(*dvConfigNode* node, void *userData, enum *dvConfigAttributeEvents* event, const char *changeKey, enum *dvConfigAttributeType* changeType, union *dvConfigAttributeValue* changeValue)

typedef struct dv_config_tree ***dvConfigTree**

typedef const struct dv_config_tree ***dvConfigTreeConst**

typedef void (***dvConfigTreeErrorLogCallback**)(const char *msg, bool fatal)

typedef union *dvConfigAttributeValue* (***dvConfigAttributeUpdater**)(void *userData, const char *key, enum *dvConfigAttributeType* type)

Enums

enum **dvConfigAttributeType**

Values:

enumerator **DVCFG_TYPE_UNKNOWN**

enumerator **DVCFG_TYPE_BOOL**

enumerator **DVCFG_TYPE_INT**

enumerator **DVCFG_TYPE_LONG**

enumerator **DVCFG_TYPE_FLOAT**

enumerator **DVCFG_TYPE_DOUBLE**

enumerator **DVCFG_TYPE_STRING**

enum **dvConfigAttributeFlags**

Values:

enumerator **DVCFG_FLAGS_NORMAL**

enumerator **DVCFG_FLAGS_READ_ONLY**

enumerator **DVCFG_FLAGS_NO_EXPORT**

enumerator **DVCFG_FLAGS_IMPORTED**

enum **dvConfigNodeEvents**

Values:

enumerator **DVCFG_NODE_CHILD_ADDED**

enumerator **DVCFG_NODE_CHILD_REMOVED**

enum **dvConfigAttributeEvents**

Values:

enumerator **DVCFG_ATTRIBUTE_ADDED**

enumerator **DVCFG_ATTRIBUTE_MODIFIED**

enumerator **DVCFG_ATTRIBUTE_REMOVED**

enumerator **DVCFG_ATTRIBUTE_MODIFIED_CREATE**

Functions

DVSDK_EXPORT const char * dvConfigNodeGetName (dvConfigNodeConst node)

DVSDK_EXPORT const char * dvConfigNodeGetPath (dvConfigNodeConst node)

DVSDK_EXPORT dvConfigNode dvConfigNodeGetParent (dvConfigNodeConst node)

This returns a reference to a node, and as such must be carefully mediated with any dvConfigNodeRemoveNode() calls.

DVSDK_EXPORT dvConfigNode * dvConfigNodeGetChildren (dvConfigNodeConst node, size_t *numChildren)

Remember to free the resulting array. This returns references to nodes, and as such must be carefully mediated with any dvConfigNodeRemoveNode() calls.

DVSDK_EXPORT void dvConfigNodeAddNodeListener (dvConfigNode node, void *userData, dvConfigNodeChangeListener node_changed)

DVSDK_EXPORT void dvConfigNodeRemoveNodeListener (dvConfigNode node, void *userData, dvConfigNodeChangeListener node_changed)

DVSDK_EXPORT void dvConfigNodeRemoveAllNodeListeners (dvConfigNode node)

DVSDK_EXPORT void dvConfigNodeAddAttributeListener (dvConfigNode node, void *userData, dvConfigAttributeChangeListener attribute_changed)

DVSDK_EXPORT void dvConfigNodeRemoveAttributeListener (dvConfigNode node, void *userData, dvConfigAttributeChangeListener attribute_changed)

DVSDK_EXPORT void dvConfigNodeRemoveAllAttributeListeners (dvConfigNode node)

DVSDK_EXPORT void dvConfigNodeRemoveNode (dvConfigNode node)

Careful, only use if no references exist to this node and all its children. References are created by dvConfigTreeGetNode(), dvConfigNodeGetRelativeNode(), dvConfigNodeGetParent() and dvConfigNodeGetChildren().

DVSDK_EXPORT void dvConfigNodeRemoveSubTree (dvConfigNode node)

Careful, only use if no references exist to this node's children. References are created by dvConfigTreeGetNode(), dvConfigNodeGetRelativeNode(), dvConfigNodeGetParent() and dvConfigNodeGetChildren().

DVSDK_EXPORT void dvConfigNodeClearSubTree (dvConfigNode startNode, bool clearStartNode)

DVSDK_EXPORT void dvConfigNodeCopy (dvConfigNodeConst source, dvConfigNode destination)

```
DVSDK_EXPORT void dvConfigNodeCreateAttribute (dvConfigNode node, const char *key,
enum dvConfigAttributeType type, union dvConfigAttributeValue defaultValue,
const struct dvConfigAttributeRanges ranges, int flags, const char *description)

DVSDK_EXPORT void dvConfigNodeRemoveAttribute (dvConfigNode node, const char *key,
enum dvConfigAttributeType type)

DVSDK_EXPORT void dvConfigNodeRemoveAllAttributes (dvConfigNode node)

DVSDK_EXPORT bool dvConfigNodeExistsAttribute (dvConfigNodeConst node,
const char *key, enum dvConfigAttributeType type)

DVSDK_EXPORT bool dvConfigNodePutAttribute (dvConfigNode node, const char *key,
enum dvConfigAttributeType type, union dvConfigAttributeValue value)

DVSDK_EXPORT union dvConfigAttributeValue dvConfigNodeGetAttribute (dvConfigNode-
Const node, const char *key, enum dvConfigAttributeType type)

DVSDK_EXPORT bool dvConfigNodeUpdateReadOnlyAttribute (dvConfigNode node,
const char *key, enum dvConfigAttributeType type, union dvConfigAttributeValue value)

DVSDK_EXPORT void dvConfigNodeCreateBool (dvConfigNode node, const char *key,
bool defaultValue, int flags, const char *description)

DVSDK_EXPORT bool dvConfigNodePutBool (dvConfigNode node, const char *key,
bool value)

DVSDK_EXPORT bool dvConfigNodeGetBool (dvConfigNodeConst node, const char *key)

DVSDK_EXPORT void dvConfigNodeCreateInt (dvConfigNode node, const char *key,
int32_t defaultValue, int32_t minValue, int32_t maxValue, int flags,
const char *description)

DVSDK_EXPORT bool dvConfigNodePutInt (dvConfigNode node, const char *key,
int32_t value)

DVSDK_EXPORT int32_t dvConfigNodeGetInt (dvConfigNodeConst node, const char *key)

DVSDK_EXPORT void dvConfigNodeCreateLong (dvConfigNode node, const char *key,
int64_t defaultValue, int64_t minValue, int64_t maxValue, int flags,
const char *description)

DVSDK_EXPORT bool dvConfigNodePutLong (dvConfigNode node, const char *key,
int64_t value)

DVSDK_EXPORT int64_t dvConfigNodeGetLong (dvConfigNodeConst node, const char *key)
```

```
DVSDK_EXPORT void dvConfigNodeCreateFloat (dvConfigNode node, const char *key,
float defaultValue, float minValue, float maxValue, int flags,
const char *description)

DVSDK_EXPORT bool dvConfigNodePutFloat (dvConfigNode node, const char *key,
float value)

DVSDK_EXPORT float dvConfigNodeGetFloat (dvConfigNodeConst node, const char *key)

DVSDK_EXPORT void dvConfigNodeCreateDouble (dvConfigNode node, const char *key,
double defaultValue, double minValue, double maxValue, int flags,
const char *description)

DVSDK_EXPORT bool dvConfigNodePutDouble (dvConfigNode node, const char *key,
double value)

DVSDK_EXPORT double dvConfigNodeGetDouble (dvConfigNodeConst node, const char *key)

DVSDK_EXPORT void dvConfigNodeCreateString (dvConfigNode node, const char *key,
const char *defaultValue, int32_t minLength, int32_t maxLength, int flags,
const char *description)

DVSDK_EXPORT bool dvConfigNodePutString (dvConfigNode node, const char *key,
const char *value)

DVSDK_EXPORT char * dvConfigNodeGetString (dvConfigNodeConst node, const char *key)

DVSDK_EXPORT bool dvConfigNodeExportNodeToXML (dvConfigNodeConst node,
const char *filePath, bool exportAll)

DVSDK_EXPORT bool dvConfigNodeExportSubTreeToXML (dvConfigNodeConst node,
const char *filePath, bool exportAll)

DVSDK_EXPORT bool dvConfigNodeImportNodeFromXML (dvConfigNode node,
const char *filePath, bool strict)

DVSDK_EXPORT bool dvConfigNodeImportSubTreeFromXML (dvConfigNode node,
const char *filePath, bool strict)

DVSDK_EXPORT char * dvConfigNodeExportNodeToXMLString (dvConfigNodeConst node,
bool exportAll)

DVSDK_EXPORT char * dvConfigNodeExportSubTreeToXMLString (dvConfigNodeConst node,
bool exportAll)

DVSDK_EXPORT bool dvConfigNodeImportNodeFromXMLString (dvConfigNode node,
const char *xmlStr, bool strict)
```

```
DVSDK_EXPORT bool dvConfigNodeImportSubTreeFromXMLString (dvConfigNode node,
const char *xmlStr, bool strict)

DVSDK_EXPORT bool dvConfigNodeStringToAttributeConverter (dvConfigNode node,
const char *key, const char *type, const char *value, bool overrideReadOnly)

DVSDK_EXPORT const char ** dvConfigNodeGetChildNames (dvConfigNodeConst node,
size_t *numNames)

DVSDK_EXPORT const char ** dvConfigNodeGetAttributeKeys (dvConfigNodeConst node,
size_t *numKeys)

DVSDK_EXPORT enum dvConfigAttributeType dvConfigNodeGetAttributeType (dvConfigN-
odeConst node, const char *key)

DVSDK_EXPORT struct dvConfigAttributeRanges dvConfigNodeGetAttributeRanges (dvCon-
figNodeConst node, const char *key, enum dvConfigAttributeType type)

DVSDK_EXPORT int dvConfigNodeGetAttributeFlags (dvConfigNodeConst node,
const char *key, enum dvConfigAttributeType type)

DVSDK_EXPORT char * dvConfigNodeGetAttributeDescription (dvConfigNodeConst node,
const char *key, enum dvConfigAttributeType type)

DVSDK_EXPORT void dvConfigNodeAttributeModifierButton (dvConfigNode node,
const char *key, const char *buttonLabel)

DVSDK_EXPORT void dvConfigNodeAttributeModifierListOptions (dvConfigNode node,
const char *key, const char *listOptions, bool allowMultipleSelections)

DVSDK_EXPORT void dvConfigNodeAttributeModifierFileChooser (dvConfigNode node,
const char *key, const char *typeAndExtensions)

DVSDK_EXPORT void dvConfigNodeAttributeModifierUnit (dvConfigNode node,
const char *key, const char *unitInformation)

DVSDK_EXPORT void dvConfigNodeAttributeModifierPriorityAttributes (dvConfigN-
ode node, const char *priorityAttributes)

DVSDK_EXPORT void dvConfigNodeAttributeModifierGUISupport (dvConfigNode node)

DVSDK_EXPORT void dvConfigNodeAttributeBooleanReset (dvConfigNode node,
const char *key)

DVSDK_EXPORT bool dvConfigNodeExistsRelativeNode (dvConfigNodeConst node,
const char *nodePath)
```

DVSDK_EXPORT dvConfigNode dvConfigNodeGetRelativeNode (dvConfigNode node, const char *nodePath)

This returns a reference to a node, and as such must be carefully mediated with any dvConfigNodeRemoveNode() calls.

DVSDK_EXPORT const char * dvConfigHelperTypeToStringConverter (enum dvConfigAttributeType type)

DVSDK_EXPORT enum dvConfigAttributeType dvConfigHelperStringToTypeConverter (const char *typeString)

DVSDK_EXPORT char * dvConfigHelperValueToStringConverter (enum dvConfigAttributeType type, union dvConfigAttributeValue value)

DVSDK_EXPORT union dvConfigAttributeValue dvConfigHelperStringToValueConverter (enum dvConfigAttributeType type, const char *valueString)

DVSDK_EXPORT char * dvConfigHelperFlagsToStringConverter (int flags)

DVSDK_EXPORT int dvConfigHelperStringToFlagsConverter (const char *flagsString)

DVSDK_EXPORT char * dvConfigHelperRangesToStringConverter (enum dvConfigAttributeType type, struct dvConfigAttributeRanges ranges)

DVSDK_EXPORT struct dvConfigAttributeRanges dvConfigHelperStringToRangesConverter (enum dvConfigAttributeType type, const char *rangesString)

DVSDK_EXPORT dvConfigTree dvConfigTreeGlobal (void)

DVSDK_EXPORT dvConfigTree dvConfigTreeNew (void)

DVSDK_EXPORT void dvConfigTreeDelete (dvConfigTree tree)

DVSDK_EXPORT void dvConfigTreeErrorLogCallbackSet (dvConfigTreeErrorLogCallback error_log_cb)

DVSDK_EXPORT dvConfigTreeErrorLogCallback dvConfigTreeErrorLogCallbackGet (void)

DVSDK_EXPORT bool dvConfigTreeExistsNode (dvConfigTreeConst st, const char *nodePath)

DVSDK_EXPORT dvConfigNode dvConfigTreeGetNode (dvConfigTree st, const char *nodePath)

This returns a reference to a node, and as such must be carefully mediated with any dvConfigNodeRemoveNode() calls.

DVSDK_EXPORT void dvConfigNodeAttributeUpdaterAdd (dvConfigNode node, const char *key, enum dvConfigAttributeType type, dvConfigAttributeUpdater updater, void *updaterUserData, bool runOnce)

DVSDK_EXPORT void dvConfigNodeAttributeUpdaterRemove (dvConfigNode node, const char *key, enum dvConfigAttributeType type, dvConfigAttributeUpdater updater, void *updaterUserData)

DVSDK_EXPORT void dvConfigNodeAttributeUpdaterRemoveAll (dvConfigNode node)

DVSDK_EXPORT void dvConfigTreeAttributeUpdaterRemoveAll (dvConfigTree tree)

DVSDK_EXPORT bool dvConfigTreeAttributeUpdaterRun (dvConfigTree tree)

DVSDK_EXPORT void dvConfigTreeGlobalNodeListenerSet (dvConfigTree tree, dvConfigNodeChangeListener node_changed, void *userData)

Listener must be able to deal with userData being NULL at any moment. This can happen due to concurrent changes from this setter.

DVSDK_EXPORT void dvConfigTreeGlobalAttributeListenerSet (dvConfigTree tree, dvConfigAttributeChangeListener attribute_changed, void *userData)

Listener must be able to deal with userData being NULL at any moment. This can happen due to concurrent changes from this setter.

file **dvConfig.hpp**

*#include "*dvConfig.h*"#include <stdexcept>#include <string>#include <string_view>#include <vector>*

file **asio_tcptlssocket.hpp**

#include <utility>#include <boost/asio.hpp>#include <boost/asio/ssl.hpp>#include <boost/version.hpp>#include <deque>

Defines

BOOST_ASIO_NET_NEW_INTERFACE

Typedefs

using **asioTCP** = asioIP::tcp

file **portable_endian.h**

#include <stdint.h>#include <string.h> Endianness conversion functions for a wide variety of systems, including Linux, FreeBSD, MacOS X and Windows.

Functions

static inline float **htobeflt** (float val)

static inline float **htoleflt** (float val)

static inline float **beflttoh** (float val)

static inline float **leflttoh** (float val)

file **portable_io.h**

#include <stdlib.h>#include "dv-sdk/api_visibility.h"#include <limits.h>

Functions

DVSDK_EXPORT char * portable_get_user_home_directory (void)

Get the user's home directory path as a string. Returned string is a dynamically allocated copy, always remember to free() it to avoid a memory leak.

Returns
string containing user home directory path. Always remember to free() this!

DVSDK_EXPORT char * portable_get_executable_location (void)

Get the current executable's location as a string. Returned string is a dynamically allocated copy, always remember to free() it to avoid a memory leak.

Returns
string containing current executable path. Always remember to free() this!

DVSDK_EXPORT char * portable_get_user_name (void)

Get the user name of the user that's currently executing the program.

Returns
string containing user name. Always remember to free() this!

file **portable_threads.h**

#include <stdbool.h>#include <stdlib.h>#include "dv-sdk/api_visibility.h"

Functions

DVSDK_EXPORT bool portable_thread_set_name (const char *name)

DVSDK_EXPORT bool portable_thread_set_priority_highest (void)

file **portable_time.h**

#include <stdbool.h>#include <stdlib.h>#include <time.h>#include "dv-sdk/api_visibility.h"

Functions

DVSDK_EXPORT bool portable_clock_gettime_monotonic (struct timespec *monoTime)

DVSDK_EXPORT bool portable_clock_gettime_realtime (struct timespec *realTime)

file **bounding_box.hpp**

#include "wrappers.hpp"#include <dv-processing/data/bounding_box_base.hpp>#include <opencv2/core.hpp>#include <opencv2/core/utility.hpp>

file **cvector_proxy.hpp**

#include <stdexcept>#include <vector>

file **imu.hpp**

#include "wrappers.hpp"#include <dv-processing/data/imu_base.hpp>

file **trigger.hpp**

#include "wrappers.hpp"#include <dv-processing/data/trigger_base.hpp>

file **types.hpp**

#include "../utils.h"#include <cassert>

Typedefs

typedef uint32_t (***dvTypePackFuncPtr**)(void *toFlatBufferBuilder, const void *fromObject)

typedef void (***dvTypeUnpackFuncPtr**)(void *toObject, const void *fromFlatBuffer)

typedef void *(***dvTypeConstructPtr**)(size_t sizeOfObject)

typedef void (***dvTypeDestructPtr**)(void *object)

typedef struct *dvTypeTimeElementExtractor* (***dvTypeTimeElementExtractorPtr**)(const void *object)

typedef bool (***dvTypeUnpackTimeElementRangeFuncPtr**)(void *toObject, const void *fromFlatBuffer, struct *dvTypeTimeElementExtractor* range)

Functions

DVSDK_EXPORT struct dvType dvTypeSystemGetInfoByIdentifier (const char *tIdentifier)

DVSDK_EXPORT struct dvType dvTypeSystemGetInfoByID (uint32_t tId)

file **wrappers.hpp**

#include "../module.h"#include "../utils.h"#include "cvector_proxy.hpp"

file **log.hpp**

#include "utils.h"#include <iostream>#include <sstream>

file **module.h**

#include "data/types.hpp"#include "utils.h"

Typedefs

typedef struct *dvModuleDataS* ***dvModuleData**

typedef struct *dvModuleFunctionsS* const ***dvModuleFunctions**

typedef struct *dvModuleInfoS* const ***dvModuleInfo**

Enums

enum **dvModuleHooks**

Values:

enumerator **DV_HOOK_DEVICE_DISCOVERY**

Functions

DVSDK_EXPORT dvModuleInfo dvModuleGetInfo170 (void)

Function to be implemented by modules. Must return a *dvModuleInfoS* structure pointer, with all the information from your module.

DVSDK_EXPORT void * dvModuleGetHooks (enum dvModuleHooks hook)

Optional function to be implemented by modules. Must return a pointer to the right structure based on given enum value, or NULL if not supported.

```
DVSDK_EXPORT void dvModuleRegisterType (dvModuleData moduleData,
const struct dvType type)
```

```
DVSDK_EXPORT void dvModuleRegisterOutput (dvModuleData moduleData,
const char *name, const char *typeName)
```

```
DVSDK_EXPORT void dvModuleRegisterInput (dvModuleData moduleData,
const char *name, const char *typeName, bool optional)
```

```
DVSDK_EXPORT struct dvTypedObject * dvModuleOutputAllocate (dvModuleData moduleData,
const char *name)
```

```
DVSDK_EXPORT void dvModuleOutputCommit (dvModuleData moduleData, const char *name)
```

```
DVSDK_EXPORT const struct dvTypedObject * dvModuleInputGet (dvModuleData moduleData,
const char *name)
```

```
DVSDK_EXPORT void dvModuleInputAdvance (dvModuleData moduleData, const char *name)
```

```
DVSDK_EXPORT void dvModuleInputDismiss (dvModuleData moduleData, const char *name,
const struct dvTypedObject *data)
```

```
DVSDK_EXPORT dvConfigNode dvModuleOutputGetInfoNode (dvModuleData moduleData,
const char *name)

DVSDK_EXPORT dvConfigNodeConst dvModuleInputGetInfoNode (dvModuleData moduleData,
const char *name)

DVSDK_EXPORT bool dvModuleInputIsConnected (dvModuleData moduleData,
const char *name)
```

file **module.hpp**

#include "module_base.hpp"#include <boost/core/demangle.hpp>#include <boost/tti/has_static_member_function.hpp>#include <typeinfo>

Defines

registerModuleClass(MODULE)

Macro that expands into the global `dvModuleGetInfo170` function, exposed to the API for DV. The function instantiates the `ModuleStaticDefinition` class with the given Module (A subclass) of *dv::ModuleBase* and returns the static info section.

Parameters

- **MODULE**

file **module_base.hpp**

#include "config.hpp"#include "log.hpp"#include "module.h"#include "module_io.hpp"#include "utils.h"#include <utility>

file **module_io.hpp**

#include "data/bounding_box.hpp"#include "data/event.hpp"#include "data/frame.hpp"#include "data/imu.hpp"#include "data/trigger.hpp"#include "data/wrappers.hpp"#include "module.h"#include "utils.h"#include <opencv2/core.hpp>

file **processing.hpp**

#include "processing/core.hpp"#include "processing/event.hpp"#include "processing/frame.hpp"

file **core.hpp**

#include "../utils.h"#include <dv-processing/core/core.hpp>

file **event.hpp**

#include "wrappers.hpp"#include <dv-processing/core/core.hpp>#include <dv-processing/core/event_color.hpp>#include <dv-processing/data/event_base.hpp>#include <opencv2/core.hpp>#include <opencv2/core/utility.hpp>

file **event.hpp**

#include "../utils.h"#include <dv-processing/core/event.hpp>

file **frame.hpp**

#include "wrappers.hpp"#include <dv-processing/data/frame_base.hpp>#include <opencv2/core.hpp>#include <opencv2/core/utility.hpp>#include <opencv2/imgproc.hpp>

file **frame.hpp**

#include "../utils.h"#include <dv-processing/core/frame.hpp>

file **stats.hpp**

#include "config.hpp"#include <boost/accumulators/accumulators.hpp>#include <boost/accumulators/statistics/max.hpp>#include <boost/accumulators/statistics/min.hpp>#include <boost/accumulators/statistics/rolling_mean.hpp>#include <boost/accumulators/statistics/rolling_variance.hpp>#include <boost/algorithm/string/predicate.hpp>#include <chrono>#include <type_traits>

file **utils.h**

#include <errno.h>#include <inttypes.h>#include <stdbool.h>#include <stddef.h>#include <stdint.h>#include <stdlib.h>#include "config/dvConfig.h"

Defines

UNUSED_ARGUMENT (arg)

Enums

enum **dvLogLevel**

Values:

enumerator **DVLOG_ERROR**

enumerator **DVLOG_WARNING**

enumerator **DVLOG_INFO**

enumerator **DVLOG_DEBUG**

Functions

DVSDK_EXPORT void dvLog (enum dvLogLevel level, const char *message)

file **version.hpp**

#include <string_view>

Defines

DV_RUNTIME_VERSION_MAJOR

dv-runtime version (MAJOR * 10000 + MINOR * 100 + PATCH).

DV_RUNTIME_VERSION_MINOR

DV_RUNTIME_VERSION_PATCH

DV_RUNTIME_VERSION

DV_RUNTIME_NAME_STRING
dv-runtime name string.

DV_RUNTIME_VERSION_STRING
dv-runtime version string.

dir **/builds/inivation/dv/dv-runtime/include/dv-sdk/config**

dir **/builds/inivation/dv/dv-runtime/include/dv-sdk/cross**

dir **/builds/inivation/dv/dv-runtime/include/dv-sdk/data**

dir **/builds/inivation/dv/dv-runtime/include/dv-sdk**

dir **/builds/inivation/dv/dv-runtime/include**

dir **/builds/inivation/dv/dv-runtime/include/dv-sdk/processing**

INDEX

H

L

R

T

U

W