HELICS Documentation

Philip Top, Jeff Daily, Ryan Mast, Dheepak Krishnamurthy, Andrew

Oct 15, 2020

Contents

This is the documentation for the Hierarchical Engine for Large-scale Infrastructure Co-Simulation (HELICS). HELICS is an open-source cyber-physical-energy co-simulation framework for energy systems, with a strong tie to the electric power system. Although, HELICS was designed to support very-large-scale (100,000+ federates) co-simulations with off-the-shelf power-system, communication, market, and end-use tools; it has been built to provide a general-purpose, modular, highly-scalable co-simulation framework that runs cross-platform (Linux, Windows, and Mac OS X) and supports both event driven and time series simulation. It provides users a high-performance way for multiple individual simulation model "federates" from various domains to interact during execution–exchanging data as time advances–and create a larger co-simulation "federation" able to capture rich interactions. Written in modern C++ (C++14), HELICS provides a rich set of APIs for other languages including Python, C, Java, and MATLAB, and has native support within a growing number of energy simulation tools.

Brief History: HELICS began as the core software development of the Grid Modernization Laboratory Consortium (GMLC) project on integrated Transmission-Distribution-Communication simulation (TDC, GMLC project 1.4.15) supported by the U.S. Department of Energy's Offices of Electricity Delivery and Energy Reliability (OE) and Energy Efficiency and Renewable Energy (EERE). As such, its first use cases center around modern electric power systems, though it can be used for co-simulation in other domains. HELICS's layered, high-performance, co-simulation framework builds on the collective experience of multiple national labs.

Motivation: Energy systems and their associated information and communication technology systems are becoming increasingly intertwined. As a result, effectively designing, analyzing, and implementing modern energy systems increasingly relies on advanced modeling that simultaneously captures both the cyber and physical domains in combined simulations. It is designed to increase scalability and portability in modeling advanced features of highly integrated power system and cyber-physical energy systems.

CHAPTER 1

Installation

The first step to using HELICS is to install it. You'll need an internet connection to run the commands in this chapter, as we'll be downloading HELICS from the internet.

We'll be showing a number of commands as code snippets in the following presentation using a terminal, and those lines may start with . You don't need to type in the character; they are there to indicate the start of each command. Lines that don't start with are typically showing the output of the previous command.

1.1 Quick start

Get the latest installers from GitHub Releases.

OR

Use conda to install the HELICS Python interface and apps:

```
conda install -c gmlc-tdc helics
```

OR

Use pip to install the HELICS Python interface and apps:

```
pip install helics
pip install helics-apps
```

OR

Use Spack (requires Spack develop branch or versions released after v0.14.1) on HPC systems (Linux/macOS) to install C/C++ HELICS components and apps:

```
spack install helics
```

For more information on supported options (e.g. using a custom HELICS build with MPI support) and troubleshooting tips, see the package managers page for more details, or the documentation for your package manager.

1.2 Using an installer for your operating system

Download pre-compiled libraries from the releases page and add them to your path. Windows users should install the latest version of the Visual C++ Redistributable. The installers come with bindings for Python (3.6), MATLAB, and Java extensions precompiled as part of the installation. All you need to do is add the relevant folders to your User's PATH variables.

On Windows, you can visit `Control Panel -> System -> Advanced System Settings -> Environment Variables` and edit your user environment variables to add the necessary Path, PYTHONPATH, JAVAPATH, MATLABPATH environment variables to the corresponding HELICS installed locations.

On MacOS or Linux, you can edit your `~/.bashrc` to add the necessary PATH, PYTHONPATH, JAVAPATH, MATLABPATH environment variables to the corresponding HELICS installed locations.

Be sure to restart your CMD prompt on Windows or Terminal on your MacOS/Linux to ensure the new environment variables are in effect.

1.3 Using a package manager for your operating system

You can install it using one of the supported package managers.

1.3.1 Package Manager

Install using conda (Windows, MacOS, Linux)

Recommended

Install Anaconda or Miniconda. It is a Python distribution but also provides a cross platform package manager called `conda`.

You can then use `conda` to install HELICS.

```
conda install -c gmlc-tdc helics
```

Install using pip (Windows, macOS, Linux, other)

Install Python with pip. Upgrade pip to a recent version using `python -m pip install --upgrade`.

If you're on a supported version of Windows, macOS, or Linux (see the HELICS PyPI page for details) you can then use `pip` to install the HELICS Python interface and helics-apps.

```
pip install helics
pip install helics-apps
```

If you are on an unsupported OS or Python version, you will need to install a copy of HELICS first. Depending on your OS, there could be a copy in the package manager, or you may need to build HELICS from source. From there, you can use `pip install helics` as above (NOTE: `pip install helics-apps` *will not work*, your package manager or HELICS build from source should install these). The source distributions section of the PyPI page has some additional useful information on this process.

Install using Spack (macOS, Linux)

Install Spack (a HELICS package is included in the Spack develop branch and Spack releases after v0.14.1).

Run the following command to install HELICS (this may take a while, Spack builds all dependencies from source!):

```
spack install helics
```

To get a list of installation options, run:

```
spack info helics
```

To enable or disable options, use +, −, and ~. For example, to build with MPI support on the command run would be:

```
spack install helics +mpi
```

Troubleshooting shared library errors on Windows

If you encounter an error along the lines of `DLL load failed: The specified module could not be found` when attempting to use the C shared library installed by a package manager, it is likely a required system dependency is missing. You can determine which DLL it is unable to find using a tool like https://github.com/lucasg/Dependencies to see what dependency is missing for the helics C shared library DLL. It is fine if it shows it can't find `WS2_32.dll`, but all other DLLs should be found. The most likely to be missing is `vcruntime140_1.dll`, which can be fixed by downloading the latest Visual C++ Redistributable from https://support.microsoft.com/en-us/help/2977003/the-latest-supported-visual-c-downloads and installing it.

Alternatively, you can install from source. See the next section for more information.

1.4 OS Specific installation from source

1.4.1 Windows Installation

Windows Installers

Windows installers are available with the different releases. The release includes zip archives with static libraries containing both the Debug version and Release version for several versions of Visual Studio. There is also an installer and zip file for getting the HELICS apps and shared library along with pre-built Python 3.6 and Java 1.8 interfaces. There is also an archive with just the C shared library and headers, intended for use with 3rd party interfaces.

Build Requirements

- Microsoft Visual C++ 2015 or newer (MS Build Tools also works)
- CMake 3.4 or newer(CMake should be newer than the Visual Studio and Boost version you are using)
- git
- Boost 1.58 or newer
- MS-MPI v8 or newer (if MPI support is needed)

Setup for Visual Studio

Note: Keep in mind that your CMake version should be newer than the boost version and your visual studio version. If you have an older CMake, you may want an older boost version. Alternatively, you can choose to upgrade your version of CMake.

To set up your environment:

1. Install Microsoft Visual C++ 2015 or newer (2017 or later is recommended)MSVC
2. Install Boost Windows downloads 1.61 or later recommended (core library should build with 1.58, but tests will not). For CMake to detect it automatically either extract Boost to the root of your drive, or set the `BOOST_INSTALL_PATH` environment variable to the install location. The CMake will only automatically find boost 1.58 or newer. Building with Visual Studio 2017 will require boost 1.65.1 or newer and CMake 3.9 or newer. Use 14.0 versions for Visual Studio 2015, 14.1 files for Visual studio 2017. Visual studio 2019 will require CMake 3.14 or later. Boost 1.70 with CMake 3.14+ is the current recommended configuration.
3. *Optional* Only if you need a global Install of ZeroMQ ZeroMQ. We **highly recommend skipping** this step and running CMake with the `HELICS_ZMQ_SUBPROJECT=ON` option enabled(which is default on windows) to automatically set up a project-only copy of ZeroMQ. The ZeroMQ Windows installer is **very** outdated and will not work with new versions of Visual Studio. The CMake generator from ZeroMQ on windows is also functional and can be used to store ZMQ in another location that will need to be specified for HELICS.
4. *Optional* Install MS-MPI if you need MPI support.
5. *Optional* Install SWIG if you wish to generate the interface libraries, appropriate build files are included in the repository so it shouldn't be necessary to regenerate unless the libraries are modified. If you want to generate the MATLAB interface a modified version of swig is necessary see *MATLAB Swig*. For Matlab, Python 3, and Java swig is not necessary. For Octave, Python2, and C# swig install is necessary. The simplest way to install swig is to use chocolatey and use

```
choco install swig
```

from windows power shell.

1. Open a Visual Studio Command Prompt, and go to your working directory.
2. Make sure *CMake* and *git* are available in the Command Prompt. If they aren't, add them to the system PATH variable.

Getting and building from source:

1. Set up your environment.
2. Open a command prompt. Use git clone to check out a copy of HELICS.

```
git clone https://github.com/GMLC-TDC/HELICS.git
```

1. Go to the checked out HELICS project folder (the default folder name is HELICS). Create a build folder and open the build folder. Alternatively, cmake-gui can be used.

```
cd HELICS
mkdir build
cd build
```

1. Run CMake. It should automatically detect where MPI is installed if the system path variables are set up correctly, otherwise you will have to set the CMake path manually. `ZMQ_LOCAL_BUILD` is set to ON so ZeroMQ will automatically be built unless the option is changed.

```
cmake ..
```

If you need CMake to use a generator other than the default (ex: selecting between a 32-bit or 64-bit project), the -G option can be used to specify one of the generators listed by CMake –help. For Visual Studio 2017, the generator name would be `Visual Studio 15 2017 [arch]`, where [arch] is optional and can be either Win64 for a 64-bit project, or left out to generate a 32-bit project. To avoid problems when building later, this should match the version of the Boost libraries you are using.

If you installed boost into the root of the C or D drives with the default location (or the BOOST_INSTALL_PATH environment variable has been set), CMake should automatically detect their location. Otherwise the location will need to be manually given to CMake. NOTE: CMake 3.14 and later separate the architecture into a separate field for the generator

2. Open the Visual Studio solution generated by CMake. Under the *Build* menu, select *Build the Solution*. Alternatively, in the MSBuild command prompt, run the command msbuild HELICS.sln from the build folder to compile the entire solution. HELICS.sln can be replaced with the name of one of the projects to build only that part of HELICS.

Testing

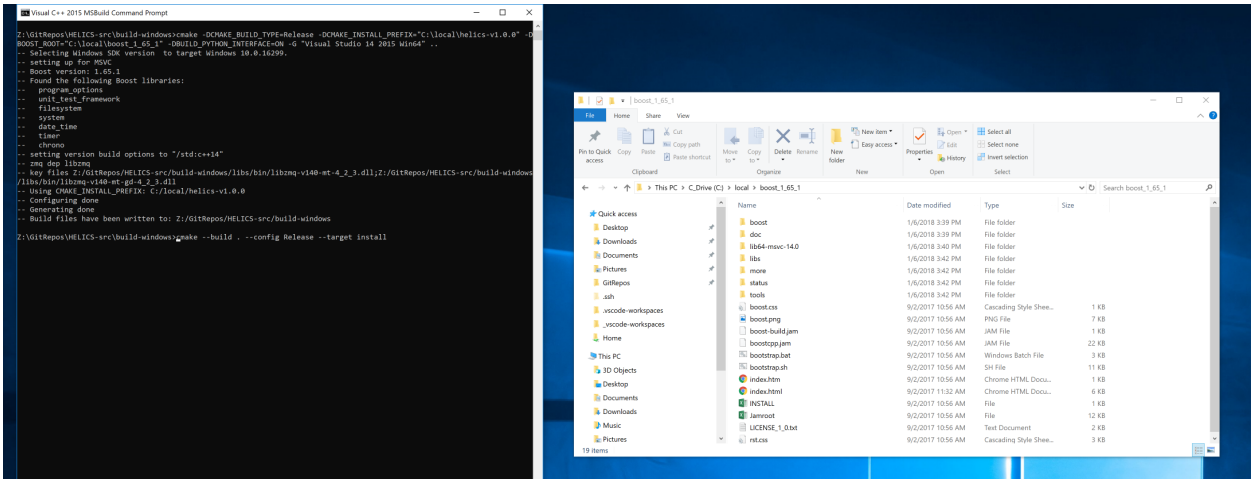
A quick test is to double check the versions of the HELICS player and recorder (located in the 'build/src/helics/apps/player/Debug' folder):

```
> cd C:/Path/To/build/src/helics/apps/Debug

> helics_player.exe --version
x.x.x 20XX-XX-XX

> helics_recorder.exe --version
x.x.x 20XX-XX-XX
```

there may be additional build information if a non tagged version is built.

Building HELICS with python support

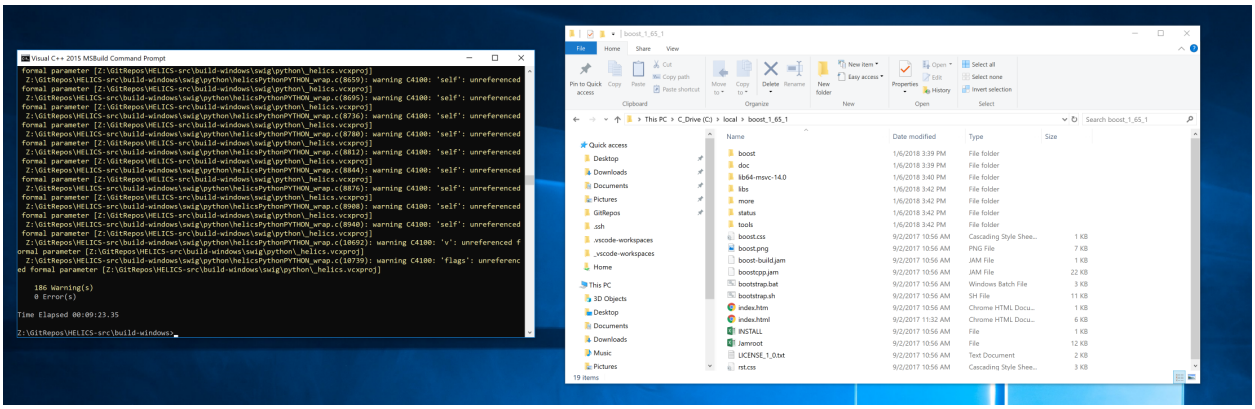
Setting `-DBUILD_PYTHON_INTERFACE=ON` will generate a project to build the python interface, if python is installed to a system path then the appropriate libraries and flags will be automatically found. If SWIG is available and you wish to regenerate the interface, `ENABLE_SWIG` can be set to ON to use swig to generate the interface files. `SWIG_EXECUTABLE` can be set to the path of the swig.exe if We highly recommend using Anaconda3/Miniconda3 for the Python distribution. Additionally, you will need to ensure that the Python distribution used is built using the same compiler architecture (x86/x64) as the one you are using to build HELICS, as well as the one that was used to build Boost (as mentioned above). ZeroMQ will be built using the CMake build process.



```
CMake -DCMake_BUILD_TYPE=Release -DCMake_INSTALL_PREFIX="C:\local\helics-X.X.X"  -
↪DBUILD_PYTHON_INTERFACE=ON -G "Visual Studio 14 2015 Win64" ..
CMake --build . --config Release --target install
```

otherwise they can be set through CMake flags

```
CMake -DCMake_BUILD_TYPE=Release -DCMake_INSTALL_PREFIX="C:\local\helics-X.X.X"  -
↪DBUILD_PYTHON_INTERFACE=ON -G "Visual Studio 14 2015 Win64" ..
CMake --build . --config Release --target install
```

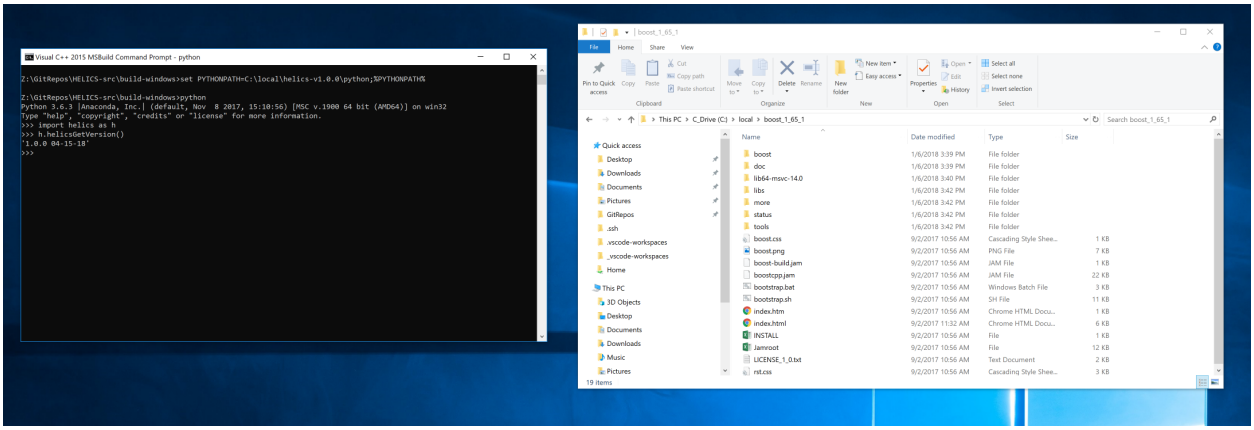


Add the following to the Windows PYTHONPATH environment variable or run the following in the command line.

```
set PYTHONPATH=C:\local\helics-X.X.X\python;%PYTHONPATH%
```

If you open a interactive Python session and import HELICS, you should be able to get the version of `helics` and an output that is similar to the following.

```
$ ipython
Python 3.6.4 |Anaconda, Inc.| (default, Jan 16 2018, 12:04:33)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.2.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import helics

In [2]: helics.helicsGetVersion()
Out[2]: 'x.x.x (20XX-XX-XX)'
```



MSYS2

MSYS2 provides a Linux like terminal environment on your Windows system. MSYS2 can be installed from here. Once MSYS2 has been installed start up msys2.exe. Follow first time updates as described on the MSYS2 website.

Using pacman package manager

HELICS is available on the Mingw-32 and Mingw-64 environments through the MSYS2 repositories. From the MINGW64 shell

```
$ pacman -Sy mingw64/mingw-w64-x86_64-helics
:: Synchronizing package databases...
 mingw32                453.3 KiB  2.86 MiB/s 00:00 [######################] 100%
 mingw32.sig            119.0   B  0.00   B/s 00:00 [######################] 100%
 mingw64                456.0 KiB  2.77 MiB/s 00:00 [######################] 100%
 mingw64.sig            119.0   B  0.00   B/s 00:00 [######################] 100%
 msys                   185.9 KiB  1804 KiB/s 00:00 [######################] 100%
 msys.sig               119.0   B  0.00   B/s 00:00 [######################] 100%
resolving dependencies...
looking for conflicting packages...

Packages (8) mingw-w64-x86_64-gcc-libs-9.2.0-2  mingw-w64-x86_64-gmp-6.2.0-1
             mingw-w64-x86_64-libsodium-1.0.18-1
             mingw-w64-x86_64-libwinpthread-git-8.0.0.5574.33e5a2ac-1
             mingw-w64-x86_64-mpc-1.1.0-1  mingw-w64-x86_64-mpfr-4.0.2-2
             mingw-w64-x86_64-zeromq-4.3.2-1  mingw-w64-x86_64-helics-2.4.0-1

Total Download Size:    9.17 MiB
Total Installed Size:  65.78 MiB

:: Proceed with installation? [Y/n] y
```

you will be asked to proceed with installation, answering `y` will install HELICS and the required dependencies.

```
$ helics_broker --version
2.4.0 (2020-02-16)
```

The helics apps and libraries are now installed, and can be updated when HELICS gets an update. For the MINGw32 use

```
$ pacman -Sy mingw32/mingw-w64-i686-helics
```

if you are installing both the 32 and 64 bit versions or just want a simpler command to type

```
$ pacboy -Sy helics
:: Synchronizing package databases...
```

if the python interface is needed on MSYS2 it can be installed through pip but requires some setup first.

```
$export CMAKE_GENERATOR="MSYS Makefiles"
$pip install helics
```

This will install the HELICS python extension in the correct location. The pacman package should be installed first

Building HELICS From Source on Windows with MSYS2

After MSYS2 has been successfully updated Some packages need to be installed in order to configure and build HELICS. The following packages need to be installed:

- base-devel
- mingw-w64-x86_64-toolchain
- git
- mingw-w64-x86_64-CMake
- mingw-w64-x86_64-boost
- mingw-w64-x86_64-qt5 (only if you want to be able to run cmake-gui which this guide recommends.)
- mingw-w64-x86_64-zeromq

All packages can be installed by typing the following:

```
$ pacman -Sy base-devel mingw-w64-x86_64-toolchain git mingw-w64-x86_64-CMake mingw-
↪w64-x86_64-boost mingw-w64-x86_64-qt5 mingw-w64-x86_64-zeromq
```

For base-devel and mingw-w64-x86_64-toolchain you may have to hit enter for installing all packages that are part of the group package. The qt5 package is quite large, if you are only using it once it might be faster to use ccmake which is a text based interface to CMake. After all the packages have been installed has been done /mingw64/bin must be in the PATH environment variable. If it isn't then it must be added. Please note that this is only necessary if you are compiling in MSYS2 shell. If you are compiling in the MSYS2 MINGW-64bit shell then /mingw64/bin will be automatically added to the PATH environment variable. If not

```
$ export PATH=$PATH:/mingw64/bin
```

Download HELICS Source Code

Now that the MSYS2 environment has been setup and all prerequisite packages have been installed the source code can be compiled and installed. The HELICS source code can be cloned from GitHub by performing the following:

```
$ git clone https://github.com/GMLC-TDC/HELICS.git
```

git will clone the source code into a folder in the current working directory called HELICS. This path will be referred to by this guide as HELICS_ROOT_DIR.

Compiling HELICS From Source

Change directories to HELICS_ROOT_DIR. Create a directory called helics-build. This can be accomplished by using the mkdir command. cd into this directory. Now type the following:

```
$ CMake-gui ../
```

If this fails that is because mingw-w64-x86_64-qt5 was not installed. If you did install it the CMake gui window should pop up. click the Advanced check box next to the search bar. Then click Configure. A window will pop up asking you to specify the generator for this project. Select "MSYS Makefiles" from the dropdown menu. The native compilers can be used and will most likely default to gcc. The compilers can also be specified manually. Select Finish; once the configure process completes finished several variables will show up highlighted in red. Since this is the first time setup the Boost and ZeroMQ library. Below are the following CMake variables that could to be verified.

- `HELICS_ENABLE_CXX_SHARED_LIB` should be checked if you are using HELICS with GridLAB-D, GridLAB-D dynamically links with the shared c++ library of HELICS, the default is off so you would need to change it

For others the advanced checkbox can be selected to see some other variables

- Boost_INCLUDE_DIR C:/msys64/mingw64/include
- Boost_LIBRARY_DIR_DEBUG/RELEASE C:/msys64/mingw64/bin
- CMake_INSTALL_PREFIX /usr/local or location of your choice
- ZeroMQ_INCLUDE_DIR C:/msys64/mingw64/include
- ZeroMQ_INSTALL_PATH C:/msys64/mingw64
- ZeroMQ_LIBRARY C:/msys64/mingw64/bin/libzmq.dll.a
- ZeroMQ_ROOT_DIR C:/msys64/mingw64

Once these CMake variables have been correctly verified click Configure if anything was changed. Once that is complete click Generate then once that is complete the CMake-gui can be closed.

Back in the MSYS2 command window[make sure you are in the build directory] type:

```
$ make -j x
```

where x is the number of threads you can give the make process to speed up the build. Then once that is complete type: `make -j` will just use the number of cores you have available

```
$ make install
```

unless you changed the value of CMake_INSTALL_PREFIX everything the default install location /usr/local/helics_2_1_0. This install path will be referred to as HELICS_INSTALL for the sections related to GridLab-D. If you want to build Gridlab-d on Windows with HELICS see Building with HELICS. Please use branch feature/1179 to build with HELICS 2.1 or later instead of the branch listed.

Compiling with clang

Clang does not work to compile on MSYS2 at this time. It has in the past but there are various issues with the clang standard library on MSYS yet so this will be updated if the situation changes. It is getting closer as of (1/30/2020) Mostly it compiles when linked with Libc++ and libc++abi, but there seems to be some missing functions as of yet, so cannot be used other than for some warning checks.

Building with mingw

HELICS can also be built with the standalone MinGW

- We assume you have MinGW installed or know how to install it.
- Boost; you can use the Windows installer for Boost installed in the default location
- Run CMake to configure and generate build files, using "MinGW Makefiles" as the generator,
- Run mingw32-make -j to build

Building with cygwin

Cygwin is another UNIX like environment on Windows. It has some peculiarities. HELICS will only build on the 32 bit version due to incompatibilities with ASIO and the 64 bit build. But it does build on the 32 bit versions completely and on the 64 bit version if `HELICS_DISABLE_ASIO=ON` is set Also the helics-config utility does not get built due to an incompatibility with the filesystem header.

- required packages include CMake, libboost-devel, make, gcc, g++, libzmq(if using zmq)
- use the unix makefiles generator

1.4.2 Mac Installation

Requirements

- C++11 compiler (C++14 preferred).
- CMake 3.4 or newer
- git
- Boost 1.58 or newer
- ZeroMQ 4.1.4 or newer (if ZeroMQ support is needed)
- MPI-2 implementation (if MPI support is needed)

Useful Resources

Some basics on using the macOS Terminal (or any Unix/Linux shell) will be useful to fully understand this guide. Articles and tutorials you may find useful include:

- How to add a new path to PATH
- Getting to Understand Linux Shell(s)
- Paths - where's my command
- Unix/Linux for Beginners
- Settling into Unix.

Setup

Note: Keep in mind that your cmake version should be newer than the boost version. If you have an older cmake, you may want an older boost version. Alternatively, you can choose to upgrade your version of cmake.

To set up your environment:

1. (if needed) Install git on your system for easy access to the HELICS source. Download from git-scm. This installs the command line which is described here. GUI's interfaces such as SourceTree are another option.
2. (if desired) Many required libraries are easiest installed using the homebrew package manager. These directions assume this approach, so unless you prefer to track these libraries and dependencies down yourself, install it if you don't have it yet.
3. (if needed) Setup a command-line compile environment

   a) Install a C++11 compiler (C++14 preferred). e.g. `clang` from the Xcode command line tools. These can be installed from the command line in Terminal by typing `xcode-select --install` and following the on-screen prompts. b) Install cmake with `brew install cmake`. Alternately, a DMG file is available for cmake from their website.
4. Install most dependencies using homebrew.

```
brew install boost
brew install zeromq
brew install cmake
```

5. Make sure *cmake* and *git* are available in the Command Prompt with `which cmake` and `which git` If they aren't, add them to the system PATH variable.

Getting and building from source:

1. Use `git clone` to check out a copy of HELICS.
2. Create a build folder. Run cmake and give it the path that HELICS was checked out into.

```
git clone https://github.com/GMLC-TDC/HELICS
cd HELICS
mkdir build
cd build
```

Compile and Install

There are a number of different options and approaches at this point depending on your needs, in particular with respect to programming language support.

Note: For any of these options, if you want to install in a custom location, you can add the following CMake argument: `-DCMAKE_INSTALL_PREFIX=/path/to/install/folder/`. There are also many other options, and you can check them out by running `ccmake .` in the `build` folder.

Keep in mind running HELICS commands like `helics_app` will not work from just any old random folder with a custom install folder. You will either need to run them from inside the `bin` subfolder of your custom install, or provide a more complete path to the command. To run HELICS commands from any folder, you must add the `bin` subfolder of your custom install to the `PATH` environment variable. See the first link in the *Useful Resources* section for details.

Basic Install (without language bindings)

Run the following:

```
cmake ../
ccmake . # optional, to change install path or other configuration settings
make
make install
```

Building HELICS with python support

Run the following:

```
cmake -DBUILD_PYTHON_INTERFACE=ON -DCMAKE_INSTALL_PREFIX=$HOME/local/helics-master/ ..
make -j8
make install
```

Add the following to your `~/.bashrc` file.

```
export PYTHONPATH=$HOME/local/helics-master/python:$PYTHONPATH
```

Building HELICS with MATLAB support

To install HELICS with MATLAB support, you will need to add run cmake with the `-DBUILD_MATLAB_INTERFACE=ON` option.

The important thing to note is that the MATLAB binaries are in the PATH. Specifically, `mex` must be available in the PATH.

Note: To check if `mex` is in the PATH, type `which mex` and see if it returns a PATH to the `mex` compiler.

If it does not, you should install MATLAB and add the path to all the MATLAB binaries to your PATH.

```
export PATH="/Applications/MATLAB_R2017b.app/bin/:$PATH"
```

```
git clone https://github.com/GMLC-TDC/HELICS
cd HELICS
mkdir build-osx
cd build-osx
cmake -DBUILD_MATLAB_INTERFACE=ON -DCMAKE_INSTALL_PREFIX=$HOME/local/helics-master/ ..
make -j8
make install
```

Building HELICS MATLAB support manually

If you have changed the C-interface and want to regenerate the SWIG MATLAB bindings, you will need to use a custom version of SWIG to build the MATLAB interface. To do that, you can follow the following instructions.

- Install SWIG with MATLAB
- `./configure --prefix=$HOME/local/swig_install; make; make install;`
- Ensure that SWIG and MATLAB are in the PATH

The below generates the MATLAB interface using SWIG.

```
cd ~/GitRepos/GMLC-TDC/HELICS/interfaces/
mkdir matlab
swig -I../src/helics/shared_api_library -outdir ./matlab -matlab ./helics.i
mv helics_wrap.cxx matlab/helicsMEX.cxx
```

You can copy these files into the respective `HELICS/interfaces/matlab/` folder and run the cmake command above. Alternatively, you wish to build the MATLAB interface without using CMake, and you can do the following.

```
cd ~/GitRepos/GMLC-TDC/HELICS/interfaces/
mex -I../src/helics/shared_api_library ./matlab/helics_wrap.cxx -lhelicsSharedLib -L/
↪path/to/helics_install/lib/helics/
mv helicsMEX.* matlab/
```

You will need HELICS installed correctly before the above can be run successfully.

Building HELICS using gcc and python

Firstly, you'll need gcc. You can `brew install gcc`. Depending on the version of gcc you'll need to modify the following instructions slightly. These instructions are for `gcc-8.2.0`.

First you will need to build boost using gcc from source. Download the latest version of boost from the boost.org website. In the following example we are doing to use boost v1.69.0 Keep in mind that your cmake version should be newer than the boost version, so if you have an older cmake you may want an older boost version. Alternatively, you can choose to upgrade your version of cmake as well.

Unzip the folder `boost_1_69_0` to any location, for example Downloads.

```
$ cd ~/Downloads/boost_1_69_0
$ ./bootstrap.sh --prefix=/ --prefix=$HOME/local/boost-gcc-1.69.0
```

Open `project-config.jam` and changes the lines as follows:

```
# Compiler configuration. This definition will be used unless
# you already have defined some toolsets in your user-config.jam
# file.
# if ! darwin in [ feature.values <toolset> ]
# {
    # using darwin ;
# }

# project : default-build <toolset>darwin ;

using gcc : 8.2 : /usr/local/bin/g++-8 ;
```

```
$ ./b2
$ ./b2 install
$ # OR
$ ./bjam cxxflags='-fPIC' cflags='-fPIC' -a link=static install # For static linking
```

This will install boost in the `~/local/boost-gcc-1.69.0` folder

Next, you will need to build HELICS and tell it what the `BOOST_ROOT` is.

```
$ cmake -DCMAKE_INSTALL_PREFIX="$HOME/local/helics-gcc-X.X.X/" -DBOOST_ROOT="$HOME/
↪local/boost-gcc-1.69.0" -DBUILD_PYTHON_INTERFACE=ON -DCMAKE_C_COMPILER=/usr/local/
↪Cellar/gcc/8.2.0/bin/gcc-8 -DCMAKE_CXX_COMPILER=/usr/local/Cellar/gcc/8.2.0/bin/g++-
↪8 ../
$ make clean; make -j 4; make install
```

Testing HELICS

Basic test (without language bindings)

A quick test is to double check the versions of the HELICS player and recorder:

```
cd /path/to/helics_install/bin

$ helics_player --version
x.x.x (20XX-XX-XX)

$ helics_recorder --version
x.x.x (20XX-XX-XX)
```

Testing HELICS with python support

If you open a interactive Python session and import helics, you should be able to get the version of `helics` and an output that is similar to the following.

```
$ ipython
Python 3.6.4 |Anaconda, Inc.| (default, Jan 16 2018, 12:04:33)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.2.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import helics

In [2]: helics.helicsGetVersion()
Out[2]: 'x.x.x (20XX-XX-XX)'
```

Testing HELICS with MATLAB support

To run the MATLAB HELICS extension, one would have to load the `helicsSharedLib` in the MATLAB file. This is run by the `helicsStartup` function in the generated MATLAB files. You can test this by opening MATLAB from the terminal or using the icon.

```
/Applications/MATLAB_R2017b.app/bin/matlab -nodesktop -nosplash -nojvm
```

and running

```
>> helicsStartup
```

Note: See Helics issue #763, if your installation doesn't point the dylib to the correct location.

You can run the following in two separate windows to test an example from the following repository:

```
git clone https://github.com/GMLC-TDC/HELICS-examples
```

Run the following in one MATLAB instance

```
matlab -nodesktop -nosplash
cd ~/GitRepos/GMLC-TDC/HELICS-examples/matlab
pireceiver
```

Run the following in a separate MATLAB instance.

```
matlab -nodesktop -nosplash
cd ~/GitRepos/GMLC-TDC/HELICS-examples/matlab
pisender
```

1.4.3 Linux Installations

Ubuntu Installation

Requirements

- Ubuntu 16 or newer
- C++14 compiler
- CMake 3.4 or newer
- Gcc 4.9 or newer (GCC 7.3.1 has a bug and won't work)
- git
- Boost 1.58 or newer
- ZeroMQ 4.1.4 or newer (if ZeroMQ support is needed)
- MPI-2 implementation (if MPI support is needed)

Setup

Note: Keep in mind that your CMake version should be newer than the boost version. If you have an older CMake, you may want an older boost version. Alternatively, you can choose to upgrade your version of CMake.

To set up your environment:

1. Install dependencies using apt-get.

   ```
   sudo apt-get install libboost-dev
   sudo apt-get install libzmq5-dev
   ```

2. Make sure *CMake* and *git* are available in the Command Prompt. If they aren't, add them to the system PATH variable.

Getting and building from source:

1. Use `git clone` to to check out a copy of HELICS.
2. Create a build folder. Run CMake and give it the path that HELICS was checked out into.
3. Run "make".

Notes for Ubuntu

Building with GCC 4.9 and 5.X on Ubuntu requires some additional flags due to the way Ubuntu builds those compilers add `-DCMake_CXX_FLAGS="-D_GLIBCXX_USE_C99 -D_GLIBCXX_USE_C99_MATH"` to make it work. If you built the compilers from source this may not be required.

```
git clone https://github.com/GMLC-TDC/HELICS
cd HELICS
mkdir build
cd build
CMake ../
cCMake . # optional, to change install path or other configuration settings
make
make install
```

Testing

A quick test is to double check the versions of the HELICS player and recorder:

```
cd /path/to/helics_install/bin

$ helics_player --version
x.x.x (20XX-XX-XX)

$ helics_recorder --version
x.x.x (20XX-XX-XX)
```

Building HELICS with python support

Run the following:

```
$ sudo apt-get install python3-dev
$ CMake -DBUILD_PYTHON_INTERFACE=ON -DCMake_INSTALL_PREFIX=~/.local/helics-X.X.X/ ..
$ make -j8
$ make install
```

Add the following to your `~/.bashrc` file.

```
export PYTHONPATH=~/.local/helics-X.X.X/python:$PYTHONPATH
export PATH=~/.local/bin:$PATH
```

Testing HELICS with python support

If you open a interactive Python session and import HELICS, you should be able to get the version of `helics` and an output that is similar to the following.

```
$ ipython
Python 3.6.4 |Anaconda, Inc.| (default, Jan 16 2018, 12:04:33)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.2.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import helics
```

(continues on next page)

(continued from previous page)

```
In [2]: helics.helicsGetVersion()
Out[2]: 'x.x.x (20XX-XX-XX)'
```

A few Specialized Platforms

The HELICS build supports a few specialized platforms, more will be added as needed. Generally the build requirements are automatically detected but that is not always possible. So a system configuration can be specified in the HELICS_BUILD_CONFIGURATION variable of CMake.

Raspbery PI

To build on Raspberry PI system using Raspbian use `HELICS_BUILD_CONFIGURATION=PI` This will add a few required libraries to the build so it works without other configuration. Otherwise it is also possible to build using `-DCMAKE_CXX_FLAGS=-latomic`

1.4.4 Docker

Requirements

- Docker 17.05 or higher

Dockerfile

This `Dockerfile` will build and install HELICS in Ubuntu 18.04 with Python support.

```
FROM ubuntu:18.04 as builder

RUN apt update && apt install -y \
  libboost-dev \
  libboost-filesystem-dev \
  libboost-program-options-dev \
  libboost-test-dev \
  libzmq5-dev python3-dev \
  build-essential swig cmake git

WORKDIR /root/develop

RUN git clone https://github.com/GMLC-TDC/HELICS.git helics

WORKDIR /root/develop/helics/build

RUN cmake \
  -DBUILD_PYTHON_INTERFACE=ON \
  -DPYTHON_INCLUDE_DIR=/usr/include/python3.6/ \
  -DPYTHON_LIBRARY=/usr/lib/x86_64-linux-gnu/libpython3.6m.so \
  -DCMAKE_INSTALL_PREFIX=/helics \
  ..
RUN make -j8 && make install
```

(continues on next page)

(continued from previous page)

```
FROM ubuntu:18.04

RUN apt update && apt install -y --no-install-recommends \
  libboost-filesystem1.65.1 libboost-program-options1.65.1 \
  libboost-test1.65.1 libzmq5

COPY --from=builder /helics /usr/local/

ENV PYTHONPATH /usr/local/python

# Python must be installed after the PYTHONPATH is set above for it to
# recognize and import libhelicsSharedLib.so.
RUN apt install -y --no-install-recommends python3-dev \
  && rm -rf /var/lib/apt/lists/*

CMD ["python3", "-c", "import helics; print(helics.helicsGetVersion())"]
```

Build

To build the Docker image, run the following from the directory containing the `Dockerfile`:

```
$ docker build -t helics .
```

Run

To run the Docker image as a container, run the following:

```
$ docker run -it --rm helics
```

Doing so should print the version and exit.

1.4.5 HELICS with language bindings support

HELICS with Python3

Run the following:

```
$ cmake -DBUILD_PYTHON_INTERFACE=ON -DCMAKE_INSTALL_PREFIX=/Users/$(whoami)/local/
↪helics-x.x.x/ ..
$ make -j8
$ make install
```

Add the following to your `~/.bashrc` file.

```
export PYTHONPATH=/Users/$(whoami)/local/helics-x.x.x/python:$PYTHONPATH
```

If you open a interactive Python session and import helics, you should be able to get the version of `helics` and an output that is similar to the following.

```
$ ipython
Python 3.6.4 |Anaconda, Inc.| (default, Jan 16 2018, 12:04:33)
Type 'copyright', 'credits' or 'license' for more information
```

(continues on next page)

(continued from previous page)

```
IPython 6.2.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import helics

In [2]: helics.helicsGetVersion()
Out[2]: 'x.x.x (XXXX-XX-XX)'
```

Note: If you already have a HELICS installation with the C shared library, it is possible to build the Python3 interface using the `interfaces/python` folder as a standalone CMake project ("e.g. `cmake <path to interfaces/python folder>`"). This can be much faster than rebuilding HELICS.

HELICS with Python2

Run the following:

```
$ cmake -DBUILD_PYTHON2_INTERFACE=ON -DPYTHON_INCLUDE_DIR=$(python2-config --prefix)/
↪include/python2.7/ -DPYTHON_LIBRARY=$(python2-config --prefix)/lib/python2.7/
↪libpython2.7.dylib -DCMAKE_INSTALL_PREFIX=/Users/$(whoami)/local/helics-x.x.x/ ..
$ make -j8
$ make install
```

Add the following to your `~/.bashrc` file.

```
export PYTHONPATH=/Users/$(whoami)/local/helics-x.x.x/python:$PYTHONPATH
```

If you open a interactive Python session and import helics, you should be able to get the version of `helics` and an output that is similar to the following.

```
$ ipython
Python 2.7.11 |Anaconda, Inc.| (default, Jan 16 2018, 12:04:33)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.2.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import helics

In [2]: helics.helicsGetVersion()
Out[2]: 'x.x.x (20XX-XX-XX)'
```

HELICS with GCC and Python3

First you will need to build boost using gcc from source. Download boost from the boost.org website.

Unzip the folder boost_1_70_0 to any location, for example Downloads.

```
$ cd ~/Downloads/boost_1_70_0
$ ./bootstrap.sh --with-python=/Users/$USER/miniconda3/python3 --prefix=/usr/local/
↪Cellar/gcc/7.2.0_1/bin/gcc-7
$ ./bootstrap.sh --prefix=/ --prefix=/Users/$USER/local/boost-gcc-1.70
$ ./b2
$ ./b2 install
$ # OR
$ ./bjam cxxflags='-fPIC' cflags='-fPIC' -a link=static install # For static linking
```

This will install boost in the `~/local/boost-gcc-1.70` folder

Next, you will need to build HELICS and tell it what the BOOST_ROOT is.

```
$ cmake -DCMAKE_INSTALL_PREFIX="/Users/$USER/local/helics-gcc-x.x.x/" -DBOOST_ROOT="/
↪Users/$USER/local/boost-gcc-1.64" -DBUILD_PYTHON_INTERFACE=ON -DPYTHON_LIBRARY=
↪$(python3-config --prefix)/lib/libpython3.6m.dylib -DPYTHON_INCLUDE_DIR=$(python3-
↪config --prefix)/include/python3.6m -DCMAKE_C_COMPILER=/usr/local/Cellar/gcc/7.2.0_
↪1/bin/gcc-7 -DCMAKE_CXX_COMPILER=/usr/local/Cellar/gcc/7.2.0_1/bin/g++-7 ../
$ make clean; make -j 4; make install
```

HELICS with MATLAB

To install HELICS with MATLAB support, you will need to add `BUILD_MATLAB_INTERFACE=ON`.

```
git clone https://github.com/GMLC-TDC/HELICS
cd HELICS
mkdir build
cd build
cmake -DBUILD_MATLAB_INTERFACE=ON -DCMAKE_INSTALL_PREFIX=/Users/$(whoami)/local/
↪helics-develop/ ..
make -j8
make install
```

On windows using visual studio the command line cmake would look like

```
cmake -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX="C:\local\helics-develop" -
↪DENABLE_SWIG=OFF -DBUILD_MATLAB_INTERFACE=ON -G "Visual Studio 15 2017 Win64" ..

cmake --build . --config Release --target INSTALL
```

Cmake will search for the MATLAB executable and execute a mex command inside MATLAB to build the interface. For this operation to succeed MATLAB must be available and mex must be setup to use an appropriate C compiler. The setup is only required once for each MATLAB installation and does not need to repeated unless the compiler changes. From within MATLAB run

```
>>mex -setup
```

and follow the prompted instructions.

Reconstructing the files requires a specific version of SWIG with MATLAB support. If swig is not used then adding `-DENABLE_SWIG=OFF` to the cmake command will bypass swig and use the included interface files directly. If any modifications to the C library were made then swig must be used to regenerate the files appropriately, otherwise the existing files can be used. The CMAKE scripts will detect if the swig is the appropriate version and act accordingly.

Add the install directory path to the MATLAB files to your PATH. This can be the system path, or the MATLAB path through the addpath command or the graphical equivalent

```
>> addpath('path/to/helics/install/matlab')
```

Now in MATLAB, run the following:

```
helicsStartup
display(helics.helicsGetVersion())
```

```
                                    < M A T L A B (R) >
                          Copyright 1984-2017 The MathWorks, Inc.
                           R2017b (9.3.0.713579) 64-bit (maci64)
                                    September 14, 2017


For online documentation, see http://www.mathworks.com/support
For product information, visit www.mathworks.com.

>> helicsStartup
Loading HELICS ...
>> helics.helicsGetVersion()

ans =

    '1.2.0 (06-18-18)'

>>
```

The helics Startup usually isn't required on Windows systems. Alternatively, you can load the HELICS library manually, depending on which operating system you use.

```
loadlibrary(GetFullPath('path/to/helics/install/libhelicsSharedLib.dylib'));
loadlibrary(GetFullPath('path/to/helics/install/libhelicsSharedLib.so'));
loadlibrary(GetFullPath('C:\path\to\helics\install\helicsSharedLib.dll'));

display(helics.helicsGetVersion())
```

This should print the version number of HELICS.

Optional

If you have changed the C-interface, and want to regenerate the SWIG MATLAB bindings, you will need to use a custom version of SWIG to build the MATLAB interface. To do that, you can follow the following instructions.

- Install SWIG with MATLAB
- `./configure --prefix=/Users/$USER/local/swig_install; make; make install;`
- Ensure that SWIG and MATLAB are in the PATH or specify them through the cmake-gui or ccmake. Then make sure swig is enabled and the Matlab files will be generated by SWIG and compiled through MATLAB.

Alternatively, you wish to build the MATLAB interface without using CMake, and you can do the following.

```
cd ~/GitRepos/GMLC-TDC/HELICS/swig/
mex -I../src/helics/shared_api_library ./matlab/helics_wrap.cxx -lhelicsSharedLib -L/
↪path/to/helics_install/lib/helics/
mv helicsMEX.* matlab/
```

The above instructions will have to be modified slightly to support Windows, CMAKE is the recommended process for creating the MATLAB interface.

HELICS with Octave

To install HELICS with Octave support, you will need to add `BUILD_OCTAVE_INTERFACE=ON`. Swig is required to build the Octave interface from source; it can be installed via package managers such as apt on Ubuntu or chocolatey on Windows, Octave can also be installed in this manner.

```
git clone https://github.com/GMLC-TDC/HELICS
cd HELICS
mkdir build
cd build
cmake -DBUILD_OCTAVE_INTERFACE=ON -DCMAKE_INSTALL_PREFIX=/Users/$(whoami)/local/
↪helics-develop/ ..
make -j8
make install
```

add the octave folder in the install directory to the octave path

```
>>helics
>> helicsGetVersion()
ans = x.x.x (20XX-XX-XX)
```

Notes

Octave 4.2 will require swig 3.0.12, Octave 4.4 and 5.0 and higher will require swig 4.0 or higher. The Octave interface has built and run smoothly on Linux systems and on the Windows system with Octave 5.0 installed through Chocolatey. There is a regular CI test that builds and tests the interface on Octave 4.2.

1.4.6 Linking with the HELICS Library

Once HELICS is built or installed it needs to be integrated into a project.

Language bindings

If the project is in Python, Matlab, Java, C#, Octave, or Julia, the language binding specific to that language is the best bet.

C based project

The C based shared library is the way to go. Either link with helicsSharedLib.lib/so/dylib and add the include directory containing a helics folder or link the CMake target HELICS::helicsSharedLib.

C++98 or C++03

Then linking with the C shared library and using the C++98 header only wrapper is the most appropriate choice. The CMake target HELICS::helicsCpp98 can be used if using CMake.

C++14

If you are using C++14 and can install or generate the C++ shared library and make sure they are built with identical compilations configurations it is possible to link with the C++ shared library generated by HELICS. This can provide a richer interface and potentially slightly faster interaction. This can be done in a couple ways. If the project is CMake based then HELICS will install a configuration file that generates the targets for the HELICS::helics-shared library with the appropriate information for linking, this is the simplest approach. Unlike the C shared library, the C++ shared library requires the importing program to have the same compilation libraries since the library does not export std library symbols, or other symbols defined solely in header files, so those must have the same interpretation inside

and out of the library for this to work. If some of the extensions available in the helics::apps are needed then the HELICS::helics-apps-shared target is also appropriate.

If you not using cmake then link against the helics-shared.lib/so/dylib as appropriate for the operating system and include the headers directory. Also advisable (though not strictly necessary is to define HELICS_SHARED_LIBRARY as part of the compilation before including some headers). The apps library is `helics-apps-shared.lib/so/dylib` and the corresponding dll/so/dylib should be on the path or in the same directory for windows or added to the RPATH on other operating systems.

If you are using CMake and want to use static libraries or use the apps library as a static library then HELICS supports building as a subproject and linking the targets HELICS::apps or HELICS::application-api.

Troubleshooting shared library errors on Windows

If you encounter an error along the lines of `DLL load failed: The specified module could not be found` when attempting to use the C shared library, it is likely a required system dependency is missing. You can determine which DLL it is unable to find using a tool like https://github.com/lucasg/Dependencies to show which dependencies were not found when attempting to open the helics C shared library DLL. It is fine if it shows it can't find `WS2_32.dll`, but all other DLLs should be found. The most likely to be missing is `vcruntime140_1.dll`, which can be fixed by downloading the latest Visual C++ Redistributable from https://support.microsoft.com/en-us/help/2977003/the-latest-supported-visual-c-downloads and installing it.

Summary

- For Python(2,3), Java, Matlab, C#, Octave, Julia - use the defined interface
- For C use the C shared library
- For most C++ use the C++98 Wrapper to the C shared library
- For C++14 in common build configurations use the C shared library or the C++ shared library from the installers or build it yourself to ensure library compatibility.
- For specific C++ applications desiring static builds and using CMake, use HELICS as a subproject in CMake (the main use is some helics extensions, but other applications can use it as well)
- For specific C++ applications with particular build considerations or flags use HELICS as a subproject and link the static or shared C++ Libraries.
- For other languages - work with swig or use the foreign language capabilities of the language to import the C shared library.

1.4.7 HELICS CMake options

Main Options

- `CMake_INSTALL_PREFIX`: CMake variable listing where to install the files
- `HELICS_BUILD_APP_LIBRARY` : [Default=ON] Tell HELICS to build the app Library
- `HELICS_BUILD_APP_EXECUTABLES` : [Default=ON] Build some executables associated with the apps
- `HELICS_BUILD_BENCHMARKS` : [Default=OFF] Build some timing benchmarks associated with HELICS
- `HELICS_BUILD_CXX_SHARED_LIB` : [Default=OFF] Build C++ shared libraries of the Application API C++ interface to HELICS and if `HELICS_BUILD_APP_LIBRARY` is also enabled another C++ shared library with the APP library

- HELICS_BUILD_EXAMPLES : [Default=OFF] Build a few select examples using HELICS, this is mostly for testing purposes. The main examples repo is here
- HELICS_BUILD_TESTS : [Default=OFF] Build the HELICS unit and system test executables.
- HELICS_ENABLE_LOGGING : [Default=ON] Enable debug and higher levels of logging, if this is turned off that capability is completely removed from HELICS
- HELICS_ENABLE_PACKAGE_BUILD : [Default=OFF] Enable the generation of some installer packages for HELICS
- HELICS_GENERATE_DOXYGEN_DOC : [Default=OFF] Generate doxygen documentation for HELICS
- HELICS_WITH_CMAKE_PACKAGE : [Default=ON] Generate a HELICSConfig.cmake file on install for loading into other libraries
- BUILD_OCTAVE_INTERACE : [Default=OFF] Build the HELICS Octave Interface
- BUILD_PYTHON_INTERACE : [Default=OFF] Build the HELICS Python3 Interface
- BUILD_PYTHON2_INTERACE : [Default=OFF] Build the HELICS Python2 Interface (can be used at the same time as BUILD_PYTHON_INTERFACE)
- BUILD_JAVA_INTERACE : [Default=OFF] Build the HELICS Java Interface
- BUILD_MATLAB_INTERACE : [Default=OFF] Build the HELICS Matlab Interface
- BUILD_CSHARP_INTERACE : [Default=OFF] Build the HELICS C# Interface, NOTE: Only available for CMake 3.8 or higher.
- CMAKE_CXX_STANDARD : Specify the C++ standard to use in building, HELICS requires 14 or higher which will be used if nothing is specified, HELICS 3.0 will require 17 or higher.
- HELICS_INSTALL :[Default=ON] If set to off HELICS will not generate any install instructions

NOTE: Most HELICS options are prefixed with HELICS_ to separate them from other libraries so HELICS can be used cleanly as a subproject. The BUILD_XXX_INTERFACE options have not been changed since that would be a large change in an intermediate version, but they will be changed in HELICS 3.0 to HELICS_BUILD_XXXX_INTERFACE to complete the prefixing change for consistency across the library.

Advanced Options

There are several different additional options available to configure HELICS for particular situations, most of which are not needed for general use and the default options should suffice.

HELICS Configuration options

These options effect the configuration of HELICS itself and how/what gets built into the HELICS core libraries

- ENABLE_ZMQ_CORE : [Default=ON] Enable the HELICS ZeroMQ related core types
- ENABLE_TCP_CORE : [Default=ON] Enable the HELICS TCPIP related core types
- ENABLE_UDP_CORE : [Default=ON] Enable the HELICS UDP core type
- ENABLE_IPC_CORE : [Default=ON] Enable the HELICS interprocess shared memory related core types
- ENABLE_TEST_CORE : [Default=OFF] Enable the HELICS in process core type with some additional features for tests, required and enabled if the HELICS_BUILD_TESTS option is enabled
- ENABLE_INPROC_CORE : [Default=ON] Enable the HELICS in process core type, required if HELICS_BUILD_BENCHMARKS is on

- ENABLE_MPI_CORE : [Default=OFF] Enable the HELICS Message Passing interface(MPI) related core types, most commonly used for High performance computing application (HPC)

HELICS logging Options

- HELICS_ENABLE_TRACE_LOGGING : [Default=ON] Enable trace level of logging inside HELICS, if this is turned off that capability is completely removed from HELICS
- HELICS_ENABLE_DEBUG_LOGGING : [Default=ON] Enable debug levels of logging inside HELICS, if this is turned off that capability is completely removed from HELICS

Build configuration Options

Options effect the connection of libraries used in HELICS and how they are linked.

- HELICS_DISABLE_BOOST : [Default=OFF] Completely turn off searching and inclusion of boost libraries. This will disable the IPC core, disable the webserver and few other features, possibly more in the future.
- HELICS_DISABLE_WEBSERVER : [Default=OFF] Disable building the webserver part of the helics_broker_server and helics_broker. The webserver requires boost 1.70 or higher and HELICS_DISABLE_BOOST will take precedence.
- HELICS_DISABLE_ASIO : [Default=OFF] Completely turn off inclusion of ASIO libraries. This will disable all TCP and UDP cores, disable real time mode for HELICS, and disable all timeout features for the Library so **use with caution**.
- HELICS_ENABLE_SUBMODULE_UPDATE : [Default=ON] Enable CMake to automatically download the submodules and update them if necessary
- HELICS_ENABLE_ERROR_ON_WARNING :[Default=OFF] Turns on Werror or equivalent, probably not useful for normal activity, There isn't many warnings but left in to allow the possibility
- HELICS_ENABLE_EXTRA_COMPILER_WARNINGS : [Default=ON] Turn on higher levels of warnings in the compilers, can be turned off if you didn't need or want the warning checks.
- STATIC_STANDARD_LIB: [Default=""] link the standard library as a static library for no additional C++ system dependencies (recognized values are default, static, and dynamic, anything else is treated the same as default)
- HELICS_ENABLE_SWIG: [Default=OFF] Conditional option if BUILD_MATLAB_INTERACE or BUILD_PYTHON_INTERFACE or BUILD_JAVA_INTERACE is selected and no other option that requires swig is used. This enables swig usage in cases where it would not otherwise be necessary.
- HELICS_USE_NEW_PYTHON_FIND: [Default=OFF] If python is required, this option can be set to use newer FindPython routines from CMake, if CMake version in use is >=3.12, This does change the variables that need to be set to link to a specific python, but can be helpful in some situations with newer python versions.
- HELICS_ENABLE_GIT_HOOKS: install a git hook to check clang format before a push
- Boost_NO_BOOST_CMAKE: [Default=OFF] This is an option related to the Boost find module, but is occasionally needed if a specific version of boost is desired and there is a system copy of BoostConfig.cmake. So if an incorrect version of boost is being found even when BOOST_ROOT is being specified this option might need to be set to ON.
- HELICS_BUILD_CONFIGURATION: A string containing a specialized build configuration if any. The only platform this is currently used on is for building on a Raspberry PI system, in which case this should be set to "PI".

ZeroMQ related Options

- HELICS_USE_SYSTEM_ZEROMQ_ONLY: [Default=OFF] Only find Zeromq through the system libraries, never attempt a local build.
- HELICS_USE_ZMQ_STATIC_LIBRARY: [Default=OFF] Build and link Zeromq using a static library. (NOTE: This has licensing implications if the resulting binary is distributed)
- HELICS_ZMQ_SUBPROJECT: [Default=ON (MSVC) OFF(otherwise)] Allow ZeroMQ to be built as a subproject if a system library is not found
- HELICS_ZMQ_FORCE_SUBPROJECT: [Default=OFF] Force ZMQ to be built and linked as a subproject.
- ZeroMQ_INSTALL_PATH: Can be used to specify a path to ZeroMQ for inclusion.

Options related to helics tests and CI configurations

- HELICS_TEST_CODE_COVERAGE :[Default=OFF] Turn on code coverage testing, enables additional linkage and options inside HELICS for coverage testing, mainly useful inside the CI or for testing.
- HELICS_ENABLE_SUBPROJECT_TESTS: [Default=OFF] Turn on some additional tests for using HELICS as a subproject, mainly used in some of the CI testing to make sure HELICS works as a subproject.
- HELICS_ENABLE_CLANG_TOOLS: [Default=OFF] Enables some helper targets for using clang-tidy and clang-format.

Hidden Options

There are a few options in the CMake system that are not visible in the GUI they mainly deal with particular situations related to release, testing, benchmarks, and code generation and should not be normally used. They are all default off unless otherwise noted.

- HELICS_SWIG_GENERATE_INTERFACE_FILES_ONLY : Use swig to generate the interface files for the different languages but don't compile them.
- HELICS_OVERWRITE_INTERFACE_FILES : Instruct CMake to take the generated files, and overwrite the existing interface files for the given language, only applies to python, Matlab, and Java. This is used in the generation of the interface files for releases and the git repo. It is only active is HELICS_SWIG_GENERATE_INTERFACE_FILES_ONLY is enabled.
- HELICS_DISABLE_SYSTEM_CALL_TESTS : There are a few test that execute system calls, which could be problematic to compile or execute on certain platforms. This option removes those tests from compilation.
- INSTALL_SYSTEM_LIBRARIES : Install system libraries with the installation, mainly useful for making a complete installer package with all needed libraries included.
- HELICS_INSTALL_PACKAGE_TESTS : Set the find_package tests to only look for HELICS in the system install paths, and enable the package-config-tests
- HELICS_DISABLE_GIT_OPERATIONS : will turn off any of the helper tools that require git, this is useful in a couple cases for building packages and other situations where updates shouldn't be checked and no modifications should be made.
- HELICS_SKIP_ZMQ_INSTALL: This is only relevant if ZMQ is built as part of the compilation process, but it skips the installation of zmq as part of HELICS install in that case.
- HELICS_BENCHMARK_SHIFT_FACTOR: For running the benchmarks this shift factor can be used to scale the number of federates used for the benchmark tests. If used it is required to be a number and is power of 2 shift from nominal values. For example for a small system a shift factor of -1 or -2 might be appropriate for the

benchmarks not to take too long. The default for systems with 4 or fewer cores is `-1` and 0 for larger compute systems. For small 2 core systems a value of -2 might be appropriate. For some very large systems a bigger value might be able to be used.

The following are a few things that could be useful to know before starting out.

Firstly, you can follow HELICS development on our GitHub page. HELICS is open-source. The development team uses `git` for version control, and GitHub to host the code publicly. The latest HELICS will be on the `develop` branch. Tagged releases occur on the `master` branch. If you clone the HELICS repository, you will be placed in the `master` branch by default. To switch to the `develop` branch, you can type the following:

```
git checkout develop
```

To switch to a tagged release, you can type the following:

```
git checkout v2.3.0
```

You will not need a full understanding of how `git` works for installing HELICS, but if you are interested you can find a good `git` resource in this page.

Secondly, HELICS is a modern C++ cross-platform software application. One challenge while maintaining the same codebase across multiple operating systems is that we have to ensure that everything installs correctly everywhere. The development team uses `CMake` to build HELICS. `CMake` is a cross-platform tool designed to build, test and package software. Having the latest version of `CMake` can make the build process much smoother. `CMake` reads certain files (CMakeLists.txt) from the HELICS repository, and creates a bunch of build files. These build files specify how different files depend on each other and when these build files are run, HELICS is built. The exact instructions to run on each operating system are given in the individual installation instructions, but one important thing to remember is that these build files are essentially temporary files. If you have an issue building HELICS, once you make a change (installing/removing/adding anything), you probably need to delete these temporary files and re-generate them. We've found in practice that you don't have to do this too often, but it can save hours of frustration if you are already aware that this needs to be done.

Another valuable piece of information about `CMake` is that almost every "OPTION" is configurable while you generate the build files. This means you can pass it configurations settings as a key value pair by adding `-D{NAME_OF_OPTION}={VALUE_OF_OPTION}` to the `cmake` command line interface. For example, to build the Python extension all you need to do is pass in `-DBUILD_PYTHON_INTERFACE=ON`. You can also run `ccmake .` in the build folder, to get a command line interactive prompt to change configuration settings. On Windows, you can use the cmake GUI to do the same. Again, there are more instructions in the individual installation pages but a useful trick to know if something isn't documented or a slightly more advanced feature is required. Available CMake options for HELICS are documented here.

CHAPTER 2

Introduction

2.1 Hello World

Now that you have HELICS installed, you are ready to create your first HELICS federation. Let's create a simple `Hello, World` example with 2 federates.

Note: This tutorial assumes basic familiarity with the command line. The HELICS co-simulation framework itself makes no specific demands about your editing, tooling, or where your code lives. Feel free to use whatever editor or IDE you are comfortable with.

Create a federations directory

Linux and Mac:

```
$ mkdir -p ~/federations/hello_world
$ cd ~/federations/hello_world
```

Windows CMD:

```
> mkdir %USERPROFILE%\federations
> cd %USERPROFILE%\federations
> mkdir hello_world
> cd hello_world
```

Writing your first federation

Next, make a new source file and call it `hello_world_sender.c`. Copy the contents from hello_world_sender.c and paste it into the file.

Next, create a new source file and call it `hello_world_receiver.c`. Copy the contents from hello_world_receiver.c and paste it into the file.

We will go through in more detail the contents of these files. For now, save the files and open two terminals.

Compiling the federates

To compile the federates, you can use the following commands.

Linux and Mac:

```
$ cc hello_world_sender.c -o ./hello_world_sender -lhelicsSharedLib
$ cc hello_world_receiver.c -o ./hello_world_receiver -lhelicsSharedLib
```

You may need to include additional include paths and library paths in the above command.

Running a federation

Linux and Mac:

Next, open three terminals. In the first terminal, run the following command.

```
$ ./helics_broker -f2
```

In the second terminal, run the following command.

```
$ ./hello_world_sender
```

In a third terminal, run the following command.

```
$ ./hello_world_receiver
```

You should see `Hello, World` printed out in the terminal where you ran the `hello_world_receiver`.

For a guide to run this example in Visual Studio go to this link: hello-world-VS.

Anatomy of a HELICS federation

Now, let's go over what just happened in the `hello_world_sender.c` part of the "Hello, World" program in detail.

The following block creates a ValueFederate. We will discuss what `FederateInfo` is and what a `ValueFederate` is, along with other types of Federates in more detail in other documents.

```
fedinfo = helicsCreateFederateInfo ();
helicsFederateInfoSetCoreTypeFromString (fedinfo,"zmq",&err);
helicsFederateInfoSetCoreInitString (fedinfo,fedinitstring,&err);
helicsFederateInfoSetTimeProperty (fedinfo,helicsGetPropertyIndex("period"), 1.0,&
↪err);
vfed = helicsCreateValueFederate ("hello_world_sender",fedinfo,&err);
```

The following registers a global publication.

```
pub = helicsFederateRegisterGlobalPublication (vfed, "hello", helics_data_type_string,
↪ "",&err);
```

The following ensures that the federation has entered execution mode. If `helicsFederateEnterInitializingMode` is not included the call to `helicsFederateEnterExecutingMode` will automatically make the call in the background.

```
helicsFederateEnterInitializingMode (vfed,&err);
helicsFederateEnterExecutingMode (vfed,&err);
```

These functions publish a String and make a RequestTime function call to advance time in the simulation.

```
helicsPublicationPublishString(pub, "Hello, World",&err);
currenttime=helicsFederateRequestTime(vfed, 1.0, &err);
```

And finally, these functions free the Federate and close the HELICS library.

```
helicsFederateFinalize (vfed,&err);
helicsFederateFree (vfed);
helicsCloseLibrary ();
```

You can see that the `hello_world_receiver.c` is also very similar, but uses a Subscription instead. A snippet of the code is shown below.

```
fedinfo = helicsCreateFederateInfo ();
helicsFederateInfoSetCoreTypeFromString (fedinfo, "zmq",&err);
helicsFederateInfoSetCoreInitString (fedinfo, fedinitstring,&err);
helicsFederateInfoSetTimeProperty (fedinfo,helics_property_time_period, 1.0,&err);

vfed = helicsCreateValueFederate ("hello_world_receiver",fedinfo,&err);
sub = helicsFederateRegisterSubscription (vfed, "hello",NULL,&err);

helicsFederateEnterInitializingMode (vfed,&err);
helicsFederateEnterExecutingMode (vfed,&err);

/** request that helics grant the federate a time of 1.0
    the new time will be returned in currentime*/
currenttime=helicsFederateRequestTime (vfed, 1.0,&err);

isUpdated = helicsInputIsUpdated (sub);
helicsInputGetString (sub, value, 128,&actualLen,&err)
printf("%s\n", value);

helicsFederateFinalize (vfed,&err);
helicsFederateFree (vfed);
helicsCloseLibrary ();
```

A note on the `&err` term Many functions in the C API take a pointer to a helics_error structure. This can be created by a call to `helicsErrorInitialize` and can be reset by `helicsErrorClear(helics_error *err)`. If an error occurs during the execution of a function or some inputs were invalid an error code in the helics_error structure will be set and a message included. For all functions if an error structure that already has an error in place is passed as an argument the function short circuits and does nothing. So checks can be done after a sequence of calls if desired with no worry about side effects. In the C++98 API an error triggers an exception, and in the base C++ API these originate as exceptions.

2.2 Python Example

In the previous section, we covered the basics of a HELICS federate and how you can run multiple federates together to form a federation. In this section we will look at how to create a federation in Python. We will create a simple `pi-exchange` federation in Python with 2 federates.

HELICS Python Setup

Before we run the Python `pi-exchange` federation, it is necessary to ensure that we have Python installed and that we have the HELICS Python built successfully and correctly on the machine.

We recommend using pip with Python (version 2.7 or 3.5+) or Anaconda3/Miniconda3 to install a copy of the HELICS Python interface, although this should work with most versions of Python if you build the interface yourself using SWIG to generate the Python bindings to the `helicsSharedLib` shared library. SWIG claims to be compatible with most Python versions, dating back to Python 2.0. And recommends that for the best results, one should consider using Python 2.3 or newer.

See the Installation instructions page for more information regarding this.

Create a federations directory

Linux and Mac:

```
$ mkdir -p ~/federations/pi-exchange
$ cd ~/federations/pi-exchange
```

Windows CMD:

```
> mkdir %USERPROFILE%\federations
> cd %USERPROFILE%\federations
> mkdir pi-exchange
> cd pi-exchange
```

Writing the Python federation

Next, make a new source file and call it `pisender.py`. Copy the contents from pisender.py and paste it into the file.

Next, create a new source file and call it `pireceiver.py`. Copy the contents from pireceiver.py and paste it into the file.

Save the files.

Running a federation

Linux and Mac:

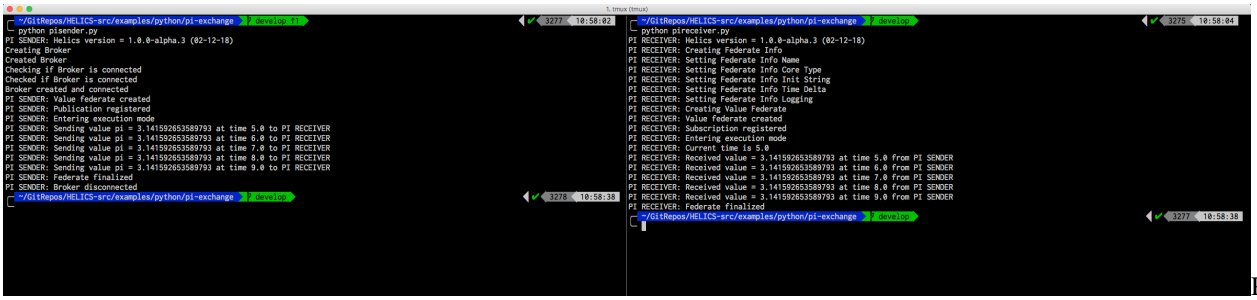
Next, open two terminals. In the first terminal, run the following command.

```
$ python pisender.py
```

In a second terminal, run the following command.

```
$ python pireceiver.py
```

If done correctly, you should see an output like so:



Python Example

You should see something like the following in the PI RECEIVER window (2nd one in directions above)

```
$ python pireceiver.py
PI RECEIVER: Helics version = x.x.x (XX-XX-XX)
PI RECEIVER: Creating Federate Info
PI RECEIVER: Setting Federate Info Name
PI RECEIVER: Setting Federate Info Core Type
PI RECEIVER: Setting Federate Info Init String
PI RECEIVER: Setting Federate Info Time Delta
PI RECEIVER: Setting Federate Info Logging
PI RECEIVER: Creating Value Federate
PI RECEIVER: Value federate created
```

(continues on next page)

(continued from previous page)

```
PI RECEIVER: Subscription registered
PI RECEIVER: Entering execution mode
PI RECEIVER: Current time is 5.0
PI RECEIVER: Received value = 3.142857142857143 at time 5.0 from PI SENDER
PI RECEIVER: Received value = 3.142857142857143 at time 6.0 from PI SENDER
PI RECEIVER: Received value = 3.142857142857143 at time 7.0 from PI SENDER
PI RECEIVER: Received value = 3.142857142857143 at time 8.0 from PI SENDER
PI RECEIVER: Received value = 3.142857142857143 at time 9.0 from PI SENDER
PI RECEIVER: Federate finalized
end of master Object Holder destructor
```

Corresponding output should appear from the PI SENDER (window 1).

Background: Running a HELICS federation (via low level commands) requires first starting a helics broker and then running the desired set of federates with it. In this case, the pisender starts this broker and then joins as a federate.

Tips

Ensure that the install location is added to your `PATH`. If you've installed to the default system location, you may not need to do this. To ensure that the Python extension works correctly, you may add the following to your `PYTHONPATH`. You can do so by pasting the following in your .bashrc file.

```
export PYTHONPATH="~/local/helics_install/python"
```

2.3 Java Minimal Example

Create a `HelloWorld.java` file with the following.

```
import com.java.helics.helics;

public class HelloWorld {

    public static void main(String[] args) {
        System.loadLibrary("JNIhelics");

        System.out.println("HELICS Version: " + helics.helicsGetVersion());
    }

}
```

Run the following to compile all Java classes. You will first have to create a `com/java/helics` folder relative to the source folder, and place all the swig generated java files in that folder.

```
javac com/java/helics/helics.java
javac HelloWorld.java
java -Djava.library.path="/Library/Java/Extensions:/Network/Library/Java/Extensions:/
↪System/Library/Java/Extensions:/usr/lib/java:/path/to/GitRepos/HELICS/build-osx/
↪swig/java/com/java/helics/:." HelloWorld
```

You should see the output that is similar to the following.

```
HELICS Version: x.x.x (XX-XX-XX)
```

creating a jar file.

```
jar cfv helics.jar com/java/helics/*.java
```

2.4 MATLAB

2.4.1 Prerequisites

- Install SWIG with MATLAB
- `./configure --prefix=/Users/$USER/local/swig_install; make; make install;`
- Ensure that SWIG and MATLAB are in the PATH

2.4.2 Building HELICS with MATLAB extension

HELICS can be built with the MATLAB extension by enabling the `BUILD_MATLAB_INTERFACE` option in cmake HELICS will also need to know the location of swig with MATLAB that was built.

It can also be built without that version of swig using existing files in the repo, but this will not work if there are any library changes. After installing the mex file will be placed in the matlab folder of the install directory.

2.4.3 Build SWIG MATLAB source

```
cd ~/GitRepos/GMLC-TDC/HELICS/swig/
mkdir matlab
swig -I../src/helics/shared_api_library -outdir ./matlab -matlab ./helicsMATLAB.i
mv helics_wrap.cxx matlab/helicsMEX.cxx
```

2.4.4 Compile MATLAB extension

```
cd ~/GitRepos/GMLC-TDC/HELICS/swig/
mex -I../src/helics/shared_api_library ./matlab/helics_wrap.cxx -lhelicsSharedLib -L/
↪path/to/helics_install/lib/helics/
mv helicsMEX.* matlab/
```

2.4.5 Test HELICS MATLAB extension

Run the following in two separate windows.

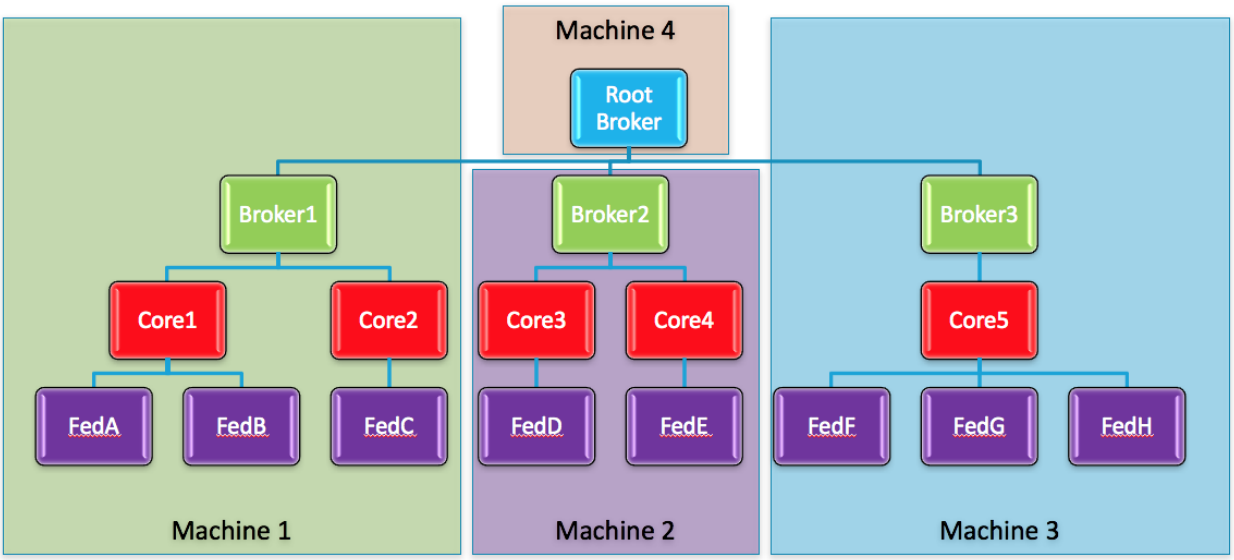
```
matlab -nodesktop -nosplash
cd ~/GitRepos/GMLC-TDC/HELICS-examples/matlab/pi-exchange
pisender
```

The pisender starts a broker so it may work slightly better to start that process first.

```
matlab -nodesktop -nosplash
cd ~/GitRepos/GMLC-TDC/HELICS-examples/matlab/pi-exchange
pireceiver
```

2.5 Terminology

1. Federate - An individual simulator that is computing something interesting and communicating with other simulators
2. Core - An object managing the interactions of one or more Federates
3. Broker - An object coordinating multiple cores or brokers
    - There can be several layers of brokers
4. Root Broker – the top broker on the hierarchy
    - Last chance router
    - Responsible for determining when to enter initialization mode for the federation
5. Federation – the set of all Federates executing together in a single co-simulation
6. Interface - a structure by which a federate can communicate with other federates. Includes Endpoints, Publications, Filters, and Inputs



Example federate hierarchy

2.6 Types of Federates

1. Value Federates
    - Direct Fixed Connections to other Federates
    - Physical values being sent back and forth
    - Associated Units
2. Message Federates
    - Packets of data
    - No fixed connections
    - For things such as Events, Communication packets, triggers

3. Combination Federate (Value+Message Federate)
4. While not a separate type Filters are supported on all federate types and could be created on a simple federate that doesn't otherwise support value or Message interfaces.

2.7 Value vs Message

Publication/Input Values	Endpoint
Fixed routes at initialization	Routes at transmission time
1 to N relationship (publications) N to 1 relationship for Inputs	All endpoints are routable - Unless otherwise specified
Values exist until updated	Destination specified
Default values	Rerouting/modification through filters
Associated units	Data exists as singular blobs - No records kept
No direct request mechanism	May define a message time - Act as events

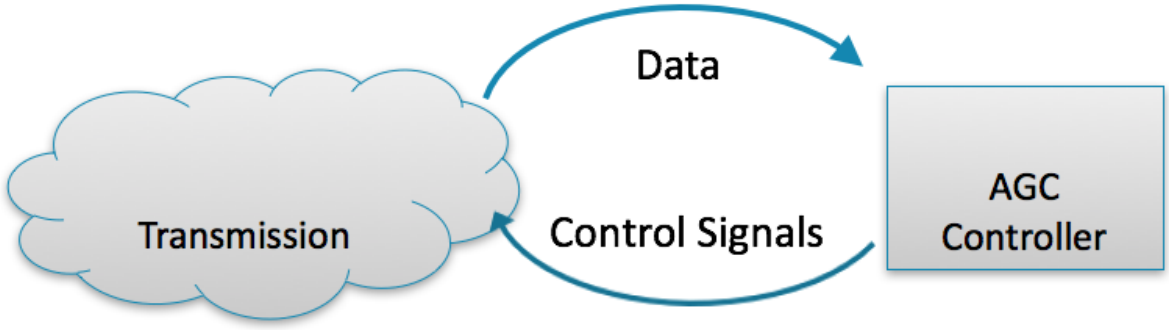
Other Notes:

- Endpoints can subscribe to publications to get a message for each data point
- Both can be nameless to be non-routable from outside the defining federate

2.8 Filters

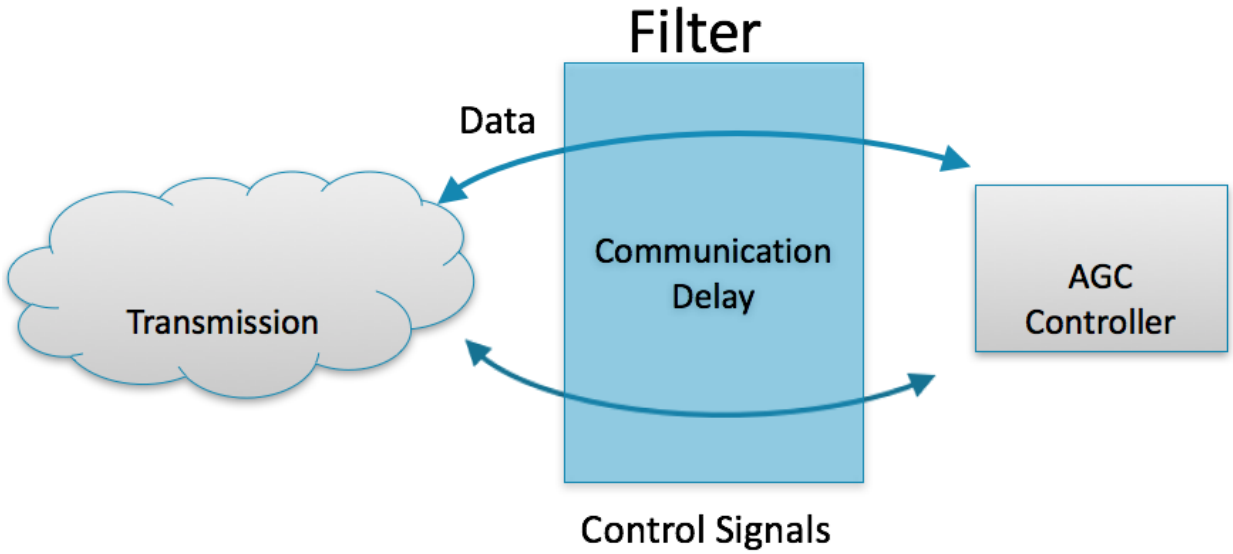
1. Inline operations that can alter a message or events
    - Time Delay (Random or Fixed)
    - Packet Translation
    - Random Dropping
    - Message Cloning / Replication
    - Rerouting
    - Firewall
    - Custom
2. Filters are part of the Core, and the effect of a filter is not limited to the endpoints of local objects
3. Filters can have multiple target endpoints, and trigger off either messages sent from an endpoint (source target) or messages received by an endpoint (destination targets)
4. Filters can be cloning or non-cloning filters. Cloning filters will operate on a copy of the message and in the simple form just deliver a copy to the specified destination locations. The original message gets delivered as it would have without the filter.

2.8.1 Federation



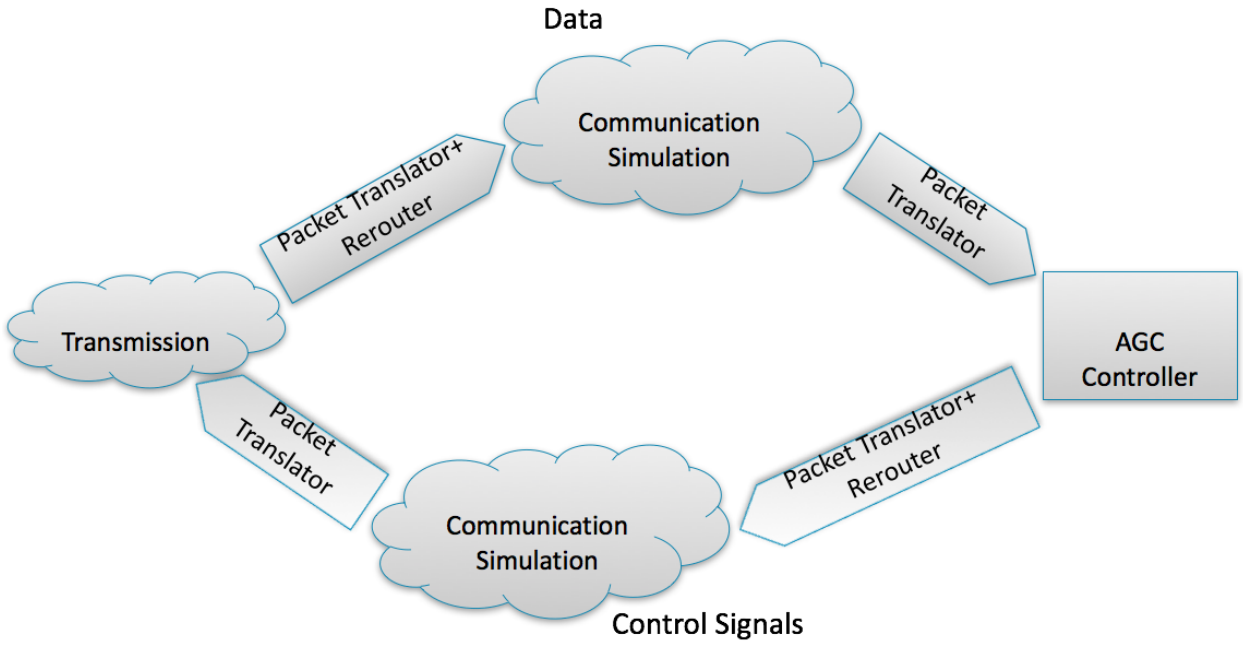
Federate communication

2.8.2 Example with delay



Federate communication with a delay filter

2.8.3 Example with communication system



Federate communication with a communication simulator

2.9 Interfacing with HELICS

The HELICS framework allows other tool developers to add HELICS components to enable co-simulation. Interfacing with HELICS can be done using the C++ Application API or the C Shared Library API. Developers familiar with HELICS and the HELICS Application API can interface with HELICS using the C++ Application API. HELICS also provides an additional well defined C API built as a shared library that can be easily interfaced with from other tools. This C interface also supports easier integration with tools developed in other languages. Developers can easily integrate HELICS into their application using the C Shared Library API using FFI.

In addition, the team has leveraged a development tool called SWIG to build interfaces in the high level programming languages listed below.

- Python (3 and 2)
- MATLAB
- Java
- Octave
- C#

SWIG allows cross platform support, i.e. extensions will work for Windows, Mac and Linux. The user does not necessarily need to install SWIG, the extension code is generated by the developers of HELICS and is only required to be built on the user end. The CMake build process includes targets for the Python, MATLAB and Java extensions. Other languages can be supported in the future. If you have a language requirement, please contact the developers.

2.10 Endpoints

Endpoints are interfaces used to pass packetized data blocks (messages) to another federate. These messages can represent communication packets, or events.

Endpoints can have a type which is a user defined string. HELICS currently does not recognize any predefined types, though some conventions may be developed in the future. An endpoint can send data to any other endpoint in the system. The data consists of raw binary data and optionally a send time.

In the Application API, endpoints can subscribe to publications to receive a message every time the publication publishes a value. They can also define destinations to send data to if no other destination is given.

Messages are delivered first by time order, then federate id number, then handle id, then by order of arrival.

2.11 Inputs

Inputs are part of a matching pair with Publications. They are the input side of a federate for data exchange.

The definition consists of a name, a type, and a unit.

The name of an Input is the identifier used for publishers to send data to it. The name can be left empty to generate a nameless subscription that is not addressable from outside the creating federate. There are some wrapper functions that generate a subsciption. This is simply a wrapper for a nameless input and an `addTarget` call to link to a publication.

Named Inputs can be global or local. The only difference is that local Inputs have the name prepended by the federate name so the global name would be in the form "federateName/inputName" whereas a global input would just have "inputName"

The type of input is represented as an open string but the Application API recognizes several well supported types, including:

- int64
- double
- string
- vector (a series of doubles)
- complex (a pair of doubles)
- vector_complex (a series of complex numbers)
- char
- bool
- time (a HELICS time value)
- named point (a value with a string and a double)

Inputs that just target a single publication (or any for that matter) can leave the type empty and it will take on the type of the publication. These are all convertible and known to the Application API. The data can be retrieved as any of these types though some are lossy. There are API functions to query the type of the input and the type of the publication that sends it data.

Inputs can add a target which is a Publication. An input can be targeted by multiple publications though the interface for dealing with this is not well developed and will be undergoing development in the coming revisions, currently the latest update in time from any publication is used as the value. Other options will be available in the future.

2.12 Publications

Publications are interfaces that send data out of a federate. They are defined through a Value Federate in the Application API.

The definition consists of a name, a type, and a unit.

The name of a publication is the identifier used for subscribers to link to it. The name can be left empty to generate a nameless publication that is not accessible from outside the creating federate. It can still send data out but all links must be specified from inside the creating federate.

Publications can be global or local. The only difference is that local Publications have the name prepended by the federate name so the global name would be like "federateName/publicationName" whereas a global publication would just have "publicationName"

The type of publication is represented as an open string but the Application API recognizes several well supported types, including:

- int64
- double
- string
- vector (a series of doubles)
- complex (a pair of doubles)
- vector_complex (a series of complex numbers)
- char
- bool
- time (a HELICS time value)
- named point (a value with a string and a double)

These are all convertible and known to the Application API but some conversions are lossy, so there is a flag to allow only string matching if that is required.

Publication can add a target which is an Input. A publication can target as many inputs as desired.

CHAPTER 3

User Guide

Co-simulation is a powerful analysis technique that allows simulators of different domains to interact through the course of the simulation, typically by dynamically exchanging values that define boundary conditions for other simulators. HELICS is a co-simulation platform that has been designed to allow integration of these simulators across a variety of computation platforms and languages. HELICS has been designed with power system simulation in mind (GridLAB-D, GridDyn, MATPOWER, OpenDSS, PSLF, InterPSS, FESTIV) but is general enough to support a wide variety of simulators and co-simulation tasks. Support for other domains is anticipated to increase over time.

3.1 Who Is This User Guide For?

There are a number of classes of HELICS users:

- New users that have little to no experience with HELICS and co-simulation in general
- Intermediate users that have run co-simulations with HELICS using simulators in which somebody else implemented the HELICS support
- Experienced users that are incorporating a new simulator and need to know how to use specific features in the HELICS API
- Developers of HELICS who are improving HELICS functionality and contributing to the code base

3.2 User Guide Overview

- **Co-Simulation Overview** - A more detailed discussion of what co-simulation is and how it is used
- **HELICS Key Concepts** - Key terms and concepts to understand before running co-simulations with HELICS
- **HELICS Co-Simulation Walk-through** - A notional walk-through of a simple transmission and distribution HELICS co-simulation to show the basic steps the software runs through
- **Federates** - Discussion of the different types of federates in HELICS (value federates and message federates) and how configure them

- **Message Filters** - How HELICS message filters can be implemented natively in HELICS or as stand-alone federates
- **Co-Simulation Timing** - How HELICS coordinates the simulation time of all the federates in the federation
- **Running HELICS co-simulations via `helics_cli`(forthcoming)** - The HELICS team has developed `helics_cli` as a standardized means of running HELICS co-simulations.
- **Cores (forthcoming)** - Discussion of the different types of message-passing buses and their implementation as HELICS cores
- **Broker Hierarchies (forthcoming)** - Advantages and disadvantages of implementing hierarchies of brokers and how that is accomplished in HELICS
- **Reiteration (forthcoming)** - Discussion of why reiteration is used and how to successfully execute it in HELICS
- **Queries** - How queries can be used to get information on HELICS brokers, federates, and cores
- **Logging** - Discussion of logging within HELICS and how to control it.
- **Getting Information from a running simulation** - Getting live information from a running co-simulation through a webserver.
- **Integrating a New Simulator** - General overview of the process by which a simulator is integrated with HELICS including usage of the common APIs
- **Trouble-Shooting HELICS Co-Simulations (forthcoming)** - What to do when the co-simulations don't seem to be working correctly.
- **Simultaneous co-simulations** - Options for running multiple independent co-simulations on a single system
- **Connecting Multiple Core Types** - What to do when one type of communication isn't sufficient.
- **N to 1 input connections** - Handling multiple publications to a single input
- **Large Co-Simulations in HELICS (forthcoming)** - How to run HELICS co-simulations with a large (100+) number of federates
- **Debugging** - Capabilities to help with debugging

3.3 Additional Resources

- HELICS API - Doxygen of the current API. If you need to know the details of the APIs and function calls, this is the place.
- HELICS federate configuration - Details on how the federates can be configured
- Installation - Instructions on how to install HELICS
- C API
- Developer's Guide - Details on how the software is assembled and some of the underlying components.
- *Existing Tools* - List of the existing tools using HELICS and some under development.
- Youtube Channel - Throughout the development of HELICS, developers and users have given mini-tutorials providing overviews of the work they have been doing. Due to its nature, many of the specifics of the content are out of date but many of the general concepts of HELICS haven't changed. A good, broad overview of the project as a whole.

CHAPTER 4

Tools with HELICS Support

The following list of tools is a list of tools that have worked with HELICS at some level either on current projects or in the past, or in some cases funded projects that will be working with certain tools. These tools are in various levels of development. Check the corresponding links for more information.

4.1 Power Systems Tools

4.1.1 Electric Distribution System Simulation

- GridLAB-D, an open-source tool for distribution power-flow, DER models, basic house thermal and end-use load models, and more. HELICS support currently (8/15/2018) provided in the `develop` branch which you have to build yourself as described here. Or a CMake based branch maintained as part of the GMLC-TDC organization.
- OpenDSS, an open-source tool for distribution powerflow, DER models, harmonics, and other capabilities traditionally found in commercial distribution analysis tools. There are two primary interfaces with HELICS support:
  - OpenDSSDirect.py which provides a "direct" interface to interact with the OpenDSS engine enabling support for non-Windows (Linux, OSX) systems.
  - PyDSS which builds on OpenDSSDirect to provide enhanced advanced inverter models and significantly more robust convergence with high-penetration DER controls along with flexible support for user-defined controls and visualization.
- CYME has been used in connection with a python wrapper interface and through FMI wrapper.

4.1.2 Electric Transmission System Simulation

- GridDyn, an open-source transmission power flow and dynamics simulator. HELICS support provided through the `cmake_updates` branch.
- PSST, an open-source python-based unit-commitment and dispatch market simulator. HELICS examples are included in the HELICS-Tutorial.

- MATPOWER, an open-source Matlab based power flow and optimal power flow tool. HELICS support under development.
- InterPSS, a Java-based power systems simulator. HELICS support under development. Use case instructions can be found here.
- PSLF has some level of support using the experimental python interface.
- PSS/E
- PowerWorld Simulator is an interactive power system simulation package designed to simulate high voltage power system operation on a time frame ranging from several minutes to several days.
- PyPower does not have a standard HELICS integration but it has been used on various projects. PYPOWER is a power flow and Optimal Power Flow (OPF) solver. It is a port of MATPOWER to the Python programming language. Current features include:
  - DC and AC (Newton's method & Fast Decoupled) power flow and
  - DC and AC optimal power flow (OPF)

4.1.3 Real time simulators

- OpalRT A few projects are using HELICS to allow connections between Opal RT and other simulations
- RTDS Some planning or testing for RTDS linkages to HELICS is underway and will be required for some known projects

4.1.4 Electric Power Market simulation

- FESTIV, the Flexible Energy Scheduling Tool for Integrating Variable Generation, provides multi-timescale steady-state power system operations simulations that aims to replicate the full time spectrum of scheduling and reserve processes (multi-step commitment and dispatch plus simplified AGC) to meet energy and reliability needs of the bulk power system.
- PLEXOS, a commercial production cost simulator. Support via OpenPLEXOS is under development.
- MATPOWER (described above) also includes basic optimal powerflow support.
- PyPower (described above) also includes basic AC and DC optimal powerflow solvers.

4.1.5 Contingency Analysis tools

- CAPE protection system modeling.
- DCAT Dynamic contingency analysis tool.

4.2 Communication Tools

- HELICS provides built-in support for simple communications manipulations such as delays, lossy channels, etc. through its built-in filters.
- ns-3, a discrete-event communication network simulator. Supported via the HELICS ns-3 module.

- OMNet++ is a public-source, component-based, modular and open-architecture simulation environment with strong GUI support and an embeddable simulation kernel. Its primary application area is the simulation of communication networks, but it has been successfully used in other areas like the simulation of IT systems, queueing networks, hardware architectures and business processes as well. Early stage development with OMNET++ and HELICS is underway and a prototype example is available in HELICS-omnetpp.

4.3 Gas Pipeline Modeling

- NGFAST.
- GasModels.jl.

4.4 Optimization packages

- GAMS.
- JuMP support is provided through the HELICS Julia interface.

4.5 Transportation modeling

- BEAM.
- POLARIS.

4.6 Buildings

- Energy Plus.

CHAPTER 5

Federate Configuration

Federates have a number of options for configuration. This includes a number of timing options controlling how the federate advances in time and interacts with other federates Other options to specify how the federate communicates with HELICS. There are also a variety of options for specifying the interfaces of a federate and the communication points by which a federate interacts with others. This can be through file configurations (JSON) or through API calls.

5.1 Federate Configuration

General federate configuration consists of setting up the name, connectivity information, and timing information. This is generally done through a `FederateInfo` object and passing that into the construction function for a federate.

5.1.1 Federate Name

A federate name is a string that contains the name of the federate, it should be unique in the federation, if not included a random UID is automatically generated. This name is prepended for any local interfaces.

5.1.2 Core information

Certain information is used by the federate to establish linkages to a core object this.

Core name

the corename identifies a potentially preexisting core in the same process that can be used or just names the created core.

Core type

See Core Types for more details on the specific types of cores which are available and their purposes, but in general the core type defines the communication method used in the federation.

Coreinitstring

The core init string is a string used by any created core to establish connectivity with a broker. This includes port numbers, addresses, and the minimum number of federates. This is usually entered as a string containing command line arguments such as `"--timeout=2s --broker 192.168.2.1"`

5.1.3 Timing information

There are a number of parameters related to timing information in HELICS. These determine what times the request time returns and how a federate handles interruptions and interacts with other federates. For a detailed description of the timing parameters see Timing in Helics.

5.1.4 Interface configuration

The interfaces (Publications, Subscriptions, Endpoints, and a Filters) are how a federate interacts with the larger federation These can be set up through API calls or through Configuration Files Json files can also contain information for the FederateInfo structure including timing and connectivity information

The specific different kinds of Federates define the patterns for different elements. ValueFederates define the interfaces for publications and Input mechanisms. MessageFederates define interfaces for endpoints and the basic Federate contains API's for interacting with Filters

Filters can be configured via files the following is an example of a JSON file. TOML configuration files are also supported. You can find examples here

```
"filters":[
{
    "name":"filtername",  //filters can have names (optional)
    "sourcetargets":"ept1", // source target for the filter
    //"inputType":"genmessage",  //can trigger some warnings if there is mismatches␣
↪for custom filters only used if operation is "custom"
    //"outputType":"genmessage",  //this could be useful if the filter actually␣
↪translates the data and can be used to automatically order filters
    "operation":"delay", //currently valid operations are "delay","clone","cloning",
↪"timedelay","randomdelay","randomdrop","reroute","redirect","custom"
    "info":"this is an information string for use by the application",
    "properties":  //additional properties for filters are specified in a property␣
↪array or object if there is just a single one
    {
        "name":"delay",  //A delay filter just has a single property
        "value":0.2    //times default to seconds though units can also be specified
↪"200 ms" or similar
    }
},
{
    "name":"filtername2",  //filters can have names (optional)
    "sourcetargets":["filterFed/ept2"],  //this is a key field specifying the source␣
↪targets can be an array
    //"dest":["dest targets"],  // field specifying destination targets
    "operation":"reroute", //currently valid operations are "delay","clone","cloning",
↪"timedelay","randomdelay","randomdrop","reroute","redirect","custom"
    "properties":  //additional properties for filters are specified in a property␣
↪array or object if there is just a single one
    {
```

(continues on next page)

(continued from previous page)

```
        "name":"newdestination",  //A reroute filter takes a new destination
        "value":"ept1"    //the value here is the endpoint that should be the new␣
↪destination
    }
},
{
    "name":"filterClone",  //filters can have names (optional)
    "delivery":"ept2",  //cloning filters can have a delivery field
    "cloning":true,  //specify that this is cloning filter
    "properties":  //additional properties for filters are specified in a property␣
↪array or object if there is just a single one
    [{
        "name":"destination",  //destination adds a cloning filter for all messages␣
↪delivered to a particular
        "value":"ept1"    //the value here the endpoint that will have its messages␣
↪cloned
    },
    {

        "name":"source",  //source adds a cloning filter for all messages send from a␣
↪particular endpoint
        "value":"ept1"    //the value here the endpoint that will have its messages␣
↪cloned
    }
    ]  //this pair of properties clone all messages to or from "ept1"  this could␣
↪also be done in one property with "endpoint" but this seemed more instructive in␣
↪this file
}
]

}
```

Notes

The properties of a filter vary depending on the exact filter specified.

Valid modes are “source”, “dest”, “clone”

for source and dest filters valid operations include “delay”, “reroute”, “randdelay”, “randomdrop”, “clone”, “custom”

for clone filters an operation of “clone” is assumed other specification result in errors on configuration.

“custom” filter operations usually require setting of a custom callback otherwise the filter won’t do anything.

5.2 Message Federates

Message Federates provide the interfaces for registering endpoints and sending and receiving messages through those endpoints. Endpoints can be configured through API calls or through file configurations

5.2.1 API calls

Endpoints can be declared through MessageFederate methods or through Endpoint objects. These are defined in MessageFederate.hpp and Endpoints.hpp. For the MessageFederate api the register calls return and `endpoint_id_t`

value that must be used whenever the endpoint is referenced. The Endpoint object api contains those calls in a separate object.

file configuration

File based configuration looks primarily at an "endpoints" JSON array

```
//this should be a valid JSON file (except comments are not recognized in standard␣
↪JSON)
{
  //example JSON configuration file for a message federate all arguments are optional
  "name": "messageFed", // the name of the federate
  //possible flags
  "observer": false, // indicator that the federate does not send anything
  "rollback": false, // indicator that the federate can use rollback -NOTE: not used␣
↪at present
  "only_update_on_change": false, //indicator that the federate should only indicate␣
↪updated values on change
  "only_transmit_on_change": false, //indicator that the federate should only publish␣
↪if the value changed
  "source_only": false, //indicator that the federate is only a source and is not␣
↪expected to receive anything
  "uninterruptible": false, //indicator that the federate should only return␣
↪requested times
  "coreType": "test", //the type of the core "test","zmq","udp","ipc","tcp","mpi"
  "coreName": "the name of the core", //this matters most for ipc and test cores, can␣
↪be empty
  "coreInit": "1", // the initialization string for the core in the form of a command␣
↪line arguments
  "maxIterations": 10, //the maximum number of iterations for a time step
  "period": 1.0, //the period with which federate may return time
  "offset": 0.0, // the offset shift in the period
  "timeDelta": 0.0, // the minimum time between subsequent return times
  "outputDelay": 0, //the propagation delay for federates to send data
  "inputDelay": 0, //the input delay for external data to propagate to federates

  //endpoints used in the federate
  "endpoints": [
    {
      "name": "ept1", // the name of the publication
      "type": "genmessage", // the type associated with a endpoint endpoint types␣
↪have limited usefulness at present (optional)
      "global": true //set to true to make the key global (default is false in which␣
↪case the publication is prepended with the federate name)
    },
    {
      "name": "ept2", // the name of the publication
      "type": "message2", // the type associated with a endpoint (optional)
      //the fact that there is no global value creates a local endpoint with global␣
↪name messageFed/ept2
      "knownDestinations": "ept1", //this value can be an array of strings or just a␣
↪single one it names key paths
      //knownDestinations can be used to optimize the communication pathways inside␣
↪of HELICS
      "subscriptions": "fed2/sub1" //subscribe an endpoint to a particular␣
↪publication  this means that an endpoint will get a message whenever anything is␣
↪published to that particular key
```

(continues on next page)

(continued from previous page)

```
      //the message will be raw data so it would have to be translated to be useful.␣
↪this can also be a JSON array to subscribe to multiple publications
    }
  ]
}
```

See the comments in the file for more information. Endpoints can subscribe to publications in which case a message is delivered for every value published.

5.3 Value Federates

Value Federates provide the API for direct data transfer interfaces This includes Publications and Inputs. Publications are outgoing data, and inputs are incoming data. Value federates provide the API to generate and interact with those types of interfaces. This can be done with configuration files or through API calls.

5.3.1 API calls

Publications and Inputs can be declared through ValueFederate methods. These are defined in ValueFederate.hpp and Publications.hpp and Inputs.hpp. For the ValueFederate api the register calls return a Publication or Input reference. These objects can be used through ValueFederate calls or as independent object with their own methods.

File configuration

File based configuration looks primarily at an "publications" or "subscriptions" JSON array.

```
//this should be a valid json file (except comments are not recognized in standard␣
↪JSON)
{
  //example json configuration file for a value federate all arguments are optional
  "name": "valueFed", // the name of the federate
  //possible flags
  "observer": false, // indicator that the federate does not send anything
  "rollback": false, // indicator that the federate can use rollback -NOTE: not used␣
↪at present
  "only_update_on_change": false, //indicator that the federate should only indicate␣
↪updated values on change
  "only_transmit_on_change": false, //indicator that the federate should only publish␣
↪if the value changed
  "source_only": false, //indicator that the federate is only a source and is not␣
↪expected to receive anything
  "uninterruptible": false, //indicator that the federate should only return␣
↪requested times
  "coretype": "test", //the type of the core "test","zmq","udp","ipc","tcp","mpi"
  "corename": "the name of the core", //this matters most for ipc and test cores, can␣
↪be empty
  "coreinit": "--autobroker", // the initialization string for the core in the form␣
↪of a command line arguments
  "max_iterations": 10, //the maximum number of iterations for a time step
  "period": 1.0, //the period with which federate may return time
  "offset": 0.0, // the offset shift in the period
  "time_delta": 0.0, // the minimum time between subsequent return times
```

(continues on next page)

(continued from previous page)

```
  "output_delay": 0, //the propagation delay for federates to send data
  "input_delay": 0, //the input delay for external data to propagate to federates

  //Publications used in the federate
  "publications": [
    {
      "key": "pub1", // the name of the publication
      "type": "double", // the type associated with a publication (optional)
      "unit": "m", // the units associated with a publication (optional)
      "global": true, //set to true to make the key global (default is false in which␣
↪case the publication is prepended with the federate name)
      "info": "this is an information string for use by the application"
    },
    {
      "key": "pub2", // the name of the publication
      "type": "double" // the type associated with a publication (optional)
      //no global:true implies this will have the federate name prepended like␣
↪valueFed/pub2
    }
  ],
  //subscriptions used in the federate
  "subscriptions": [
    {
      "key": "pub1", // the key of the publication
      "required": true //set to true to make helics issue a warning if the␣
↪publication is not found
    },
    {
      "key": "fedName/pub2", // the name of the publication to subscribe to
      "shortcut": "pubshortcut", //a naming shortcut for the publication for later␣
↪retrieval
      "info": "this is an information string for use by the application"
    }
  ],

  "inputs": [
    { "key": "ipt2", "type": "double", "required": true, "target": "pub1" }
    //specify an input with a target multiple targets could be specified like "targets
↪":["pub1","pub2","pub3"]
  ],

  "globals": [
    ["global1", "this is a global1 value"],
    ["global2", "this is another global value"]
  ]
}
```

Notes

Shortcuts just provide a shortcut name for later reference instead of having to use a potentially longer key, the shortcut is only relevant inside a single federate.

5.4 Filters

Filters are interfaces which can modify messages including routes, destinations, times, existence, and payloads. This is useful for inserting communication modules into a data stream as an optional component they can also be used to clone messages and randomly drop them.

5.4.1 Filter Creation

Filters are registered with the core or through the application API. There are also Filter object that hide some of the API calls in a slightly nicer interface. Generally a filter will define a target endpoint as either a source filter or destination filter. Source filters can be chained, as in there can be more than one of them. At present there can only be a single non-cloning destination filter attached to an endpoint.

Non-cloning filters can modify the message in some way, cloning filters just copy the message and may send it to multiple destinations.

On creation, filters have a target endpoint and an optional name. Custom filters may have input and output types associated with them. This is used for chaining and automatic ordering of filters. Filters do not have to be defined on the same core as the endpoint, and in fact can be anywhere in the federation, any messages will be automatically routed appropriately.

5.4.2 predefined filters

Several predefined filters are available, these are parameterized so they can be tailored to suite the simulation needs

reroute

This filter reroutes a message to a new destination. it also has an optional filtering mechanism that will only reroute if some patterns are matching the patterns should be specified by “condition” in the set string the conditions are regular expression pattern matching strings

delay

This filter will delay a message by a certain amount

randomdelay

This filter will randomly delay a message according to specified random distribution available options include distribution selection, and 2 parameters for the distribution some distributions only take one parameter in which case the second is ignored. The distributions available are based on those available in the C++ <random> library

- constant - param1=”value” this just generates a constant value
- uniform param1=”min”, param2=”max”
- bernoulli param1=”prob”, param2=”value” the bernoulli distribution will return param2 if the bernoulli trial returns true, 0.0 otherwise. Param1 is the probability of returning param2
- binomial param1=t (cast to int) param2=”p”
- geometric param 1=”prob” the output is param2*geom(param1) so multiplies the integer output of the geometric distribution by param2 to get discrete chunks

- poisson param1="mean"
- exponential param1="lambda"
- gamma param1="alpha" param2="beta"
- weibull param1="a" param2="b"
- extreme_value param1="a" param2="b"
- normal param1="mean", param2="stddev"
- lognormal param1="mean", param2="stddev"
- chi_squared param1="n"
- cauchy param1="a" param2="b"
- fisher_f param1="m" param2="n"
- student_t param1="n"

randomdrop

This filter will randomly drop a message, the drop probability is specified, and is modeled as a uniform distribution.

clone

this message will copy a message and send it to the original destination plus a new one.

firewall

The firewall filter will eventually be able to execute firewall like rules on messages and perform certain actions on them, that can set flags, or drop or reroute the message. The nature of this is still in development and will be available at a later release.

custom filters

Custom filters are allowed as well, these require a callback operator that can be called from any thread and modify the message in some way.

5.5 Federate Timing

Time control in a federation is handled via `timeController` objects in each Federate and Core. This allows Federation timing to be handled in a distributed fashion and each federate can tune the timing in a way that is appropriate for the Federate.

The parameters associated with the time control are in `FederateInfo`. They include `inputDelay`, `outputDelay`, `period`, `minTimeDelta`, and `offset`. These parameters along with the `timeRequest` functions determine how time advances in a federate.

5.5.1 Timing Parameters

Thse parameters take a time specification

Period and Offset

The period and offset of a Federate determine the allowable times which a federate may grant. All granted times for a federate will be in accordance with the following:

```
T=n*Period+offset
```

With the exception that all federates are granted time=0 when entering execution mode. `n` can be 0 so if the offset is greater than 0 then the first granted time will T=offset. The default values for both period and offset are 0. Offset can be set to a value bigger than the period if a federate wishes to skip ahead and ignore transients or other updates going on in the first part of a co-simulation.

minTimeDelta

The minimum time delta of federate determines how close two granted times may be to each other. The default value is set to the system epsilon which is the minimum time resolution of the Time class used in HELICS. This can be used to achieve similar effects as the period, but it has a different meaning. If the period is set to be smaller than the `minTimeDelta`, then when granted the federate will skip ahead a couple time steps.

With these parameters many different patterns are possible.

Input Delay

The input delay can be thought of as the propagation delay for signals going into a federate. Basically all values and signals are only acknowledged in the timing calculations after the prescribed delay.

Output Delay

The output delay is symmetrical to the input delay. Except it applies to all outgoing messages. Basically once a time is granted the federate cannot effect other federates until `T+outputDelay`.

rt_lag

real time tolerance - the maximum time grants can lag real time before HELICS automatically acts default=0.2 given this operates on a computer clock, time <0.005 are not going be very accurate or followed that closely unless the OS is specifically setup for that sort of timing level

rt_lead

real time tolerance - the maximum time grants can lead real time before HELICS forces an additional delay

5.5.2 Timing Flags

uninterruptible

If set to true the federate can only return time expressly requested(or the next valid time after the requested time)

source_only

Indicator that the federate is only used for signal generation and doesn't depend on any other federate for timing. Having subscriptions or receiving messages is still possible but the timing of them non-deterministic.

observer

If the observer flag is set to true, the federate is intended to be receive only and will not impact timing of any other federate sending messages from an observer federate is undefined.

rollback (not used)

Should be set to true for federates that support rollback

only_update_on_change

If set to true a federate will only trigger a value update if the value has actually changed on a granted time. Change is defined as binary equivalence, Subscription objects can be used for numerical limits and other change detection.

only_transmit_on_change

If set to true a federate will only transmit publishes if the value has changed. Change is defined as binary equivalence. If numerical deltas and ranges are desired use Publication objects for finer grained control. This flag applies federate wide.

wait_for_current_time_update

If set to true a federate will wait on the requested time until all other federates have completed at least 1 iteration of the current time or have moved past it. If it is known that 1 federate depends on others in a non-cyclic fashion, this can be used to optimize the order of execution without iterating.

realtime

If set to true the federate uses `rt_lag` and `rt_lead` to match the time grants of a federate to the computer wall clock. If the federate is running faster than real time this will insert additional delays. If the federate is running slower than real time this will cause a force grant, which can lead to non-deterministic behavior. `rt_lag` can be set to maxVal to disable force grant

restrictive-time-policy

Using the option `restrictive-time-policy` forces HELICS to use a fully conservative mode in granting time. This can be useful in situations beyond the current reach of the distributed time algorithms. It is generally used in cases where it is known that some federate is executing and will trigger someone else, but most federates won't know who that might be. This prevents extra messages from being sent and a potential for time skips. It is not needed if some federates are periodic and execute every time step. It is currently only used in few benchmarks using peculiar configurations. The flag can be used for federates and for brokers and cores to force very conservative timing.

5.6 Federate info

The FederateInfo structure contains information that can be passed to a federation upon construction. Some information can be updated continuously other can be only be changed before initializationMode is entered.

separator [char]

the separator character between federateName and endpoint or publications that are not declared global. the default is '/'

coreName [string]

The name of the core to connect with, can be left blank to either find an available core or generate one automatically.

coreInitString [string]

Command line arguments that are passed to the core when starting it. Some examples are:

- "-f2" to specify 2 federates will connect
- "-f1 –broker=192.168.2.3:23444" to specify a single federate and to connect to a broker at ipaddress 192.168.2.3 port 23444

coreType [enum]

Specify which type of core to use. See core types for more details

They can be generated from a string using the

```
core_type coreTypeFromString (std::string type) noexcept
```

function call. The function

```
bool isCoreTypeAvailable (core_type type) noexcept;
```

will check if the specified core type is available in the current build of the library on a specific platform.

broker [string]

specify the broker to connect to, can be an ipaddress, or a name of the broker depending on the core type and federation configuration.

localport [string]

The local ip port to use for incoming connections. This is usually a number but depending on the system some ports can be named.

properties [bool]

Federate info structures accept properties as either Time values, integers, or flag values (bool). These are entered through the `setProperty` calls or the `setFlagOption` call. The function calls take a propertyID and a value. For a description of the available options see Timing and helics_enums and helics_definitions

5.6.1 Timing control variables

see timing for more details.

timeDelta[time]

the minimum time advance allowed by the federate default timeEpsilon

outputDelay[time]

The amount of time values and messages take to propagate to be available to external federates. default= 0

inputDelay[time]

the time it takes values and messages to propagate to be accessible to the Federate default=0

period[time]

a period value, all granted times must be on this period n*Period+offset default=0

offset[time]

offset to the time period default=0

rt_lag[time]

real time tolerance - the maximum time grants can lag real time before HELICS automatically acts default=0.2 given this operates on a computer clock, time <0.005 are not going be very accurate or followed that closely unless the OS is specifically setup for that sort of timing level

rt_lead[time]

real time tolerance - the maximum time grants can lead real time before HELICS forces an additional delay default 0.1

5.6.2 Timing flags

- `observer` = false flag indicating that the federate is an observer
- `uninterruptible` =false flag indicating that the federate should never return a time other than requested
- `source_only` = false; flag indicating that the federate does not receive or do anything with received information.
- `only_transmit_on_change` =false flag indicating that values should only updated if the number has actually changes
- `only_update_on_change` = false flag indicating values should be discarded if they are not changed from previous values
- `wait_for_current_time_updates` = false flag indicating that the federate should only grant when no more messages can be received at the current time
- realtime = false flag indicating that the federate is required to operate in real time. the federate must have a non-zero period
- `slow_responding` = false flag indicating that the federate might be slow to respond to internal pings or take a long time between steps

5.6.3 Other Controls

maxIterations[int16]

the maximum number of iterations allowed for the federate default=50

logLevel[int32]

the logging level above which not to log to file default 1(WARNING)

5.7 Federate flags

There are a number of flags which control how a federate acts with respect to timing and its interfaces. The Timing flags and controls are described here. There are also a number of other flags which control some aspects of the interfaces, and a few other flags which can be applied to specific interfaces.

5.7.1 single_thread_federate

If specified in the federateInfo on creation this tells the core that this federate will only execute in a single thread and only a single federate is interacting with the connected core.

NOTE: This option is not fully enabled and won't be fully available until HELICS 3.0 is released.

This disables the asynchronous functions in the federate and turns off a number of protection mechanisms for handling federate interaction across multiple threads. This can be used for performance reasons and can interact with the single_thread core types that are in development.

5.7.2 ignore_time_mismatch_warnings

If certain timing options are used this can cause the granted time to be greater than the requested time. For example with the `period`, or `minTimeDelta` specified. This situation would normally generate a warning message, but if this option is enabled those warnings are silenced.

5.7.3 connections_required

When an interface requests a target it tries to find a match in the federation. If it cannot find a match at the time the federation is initialized, then the default is to generate a warning. This will not halt the federation but will display a log message. If the connections required flag is set on a federate all subsequent `addTarget` calls on any interface will generate an error if the target is not available. If the addTarget is made after the initialization point, the error is immediate.

5.7.4 connections_optional

When an interface requests a target it tries to find a match in the federation. If it cannot find a match at the time the federation is initialized, then the default is to generate a warning. This will not halt the federation but will display a log message. If the `connections_optional` flag is set on a federate all subsequent `addTarget` calls on any interface will not generate any message if the target is not available.

5.7.5 strict_input_type_checking

This applies to Input interface. If enabled this flag tells the inputs to check that the type matches.

5.7.6 slow_responding

If specified on a federate it indicates the federate may be slow in responding, and to not disconnect the federate if things are slow. If applied to a core or broker, it is indicative that the broker doesn't respond to internal pings quickly so they cannot be used as a mechanism for timeout. For federates this option doesn't do much but its role will likely be expanded as more robust timeout and coordination mechanics are developed.

5.7.7 debugging

If a program is being debugged and may halt execution the `--debugging` flag may be used to turn off some timeouts and keep everything working a little more smoothly. This flag is the equivalent of "–slow_responding" for a federate and "–slow_responding –disable_timer‘ for a broker/core.

5.7.8 terminate on error

If the `terminate_on_error` flag is set then a federate encountering an internal error will trigger a global error and cause the entire federation to abort. If the flag is not set then errors will only be local. Errors of this nature are typically the result of configuration errors. For example having a required publication that is not used or incompatible units or types on publications and subscriptions.

5.8 Core Types

There are several different core/broker types available in HELICS These can be used in different circumstances depending on the platform and system desires

5.8.1 Test

The Test core functions in a single process, and works through inter-thread communications. Its primary purpose is to test communication patterns and algorithms. However, in situations where all federates can be run in a single process it is probably the fastest and easiest to setup, and it is fully operational.

5.8.2 Interprocess

The Interprocess core uses memory mapped files to transfer data, In some circumstances it can be faster than the other cores It can only be used inside a single shared memory platform. It also has some limitations on Message sizes. It does not support multi tiered brokers.

5.8.3 ZMQ

The ZMQ is the primary core to use for multi-machine systems. It uses the ZMQ mechanisms. Internally it makes use of the REQ/REP mechanics for priority communications and PUSH/PULL for non-priority communication messages.

5.8.4 ZMQ_SS

The ZMQ_SS core was developed to minimize the number of sockets in use to support very high federate counts on a single machine. It uses the DEALER/ROUTER mechanics instead of PUSH/PULL

5.8.5 UDP

UDP communications sends IP messages. UDP communication is not guaranteed or ordered, but may be faster in cases with highly reliable networking Its primary use is for performance testing. The UDP core uses asio for networking

5.8.6 TCP

TCP communications is an alternative to ZMQ on platforms where ZMQ is not available, performance comparisons have not been done, so it is unclear as to the relative performance differences between TCP, UDP, and ZMQ. It uses the asio library for networking

5.8.7 TCP_SS

The TCP_SS core is targeted at firewall applications to allow the outgoing connections to be made from the cores or brokers and have only a single external socket exposed

5.8.8 MPI

MPI communications is often used in HPC systems. It uses the message passing interface to communicate between nodes in an HPC system. It is still in testing and over time there is expected to be a few different levels of the MPI core used in different platforms depending on MPI versions available and federation needs.

CHAPTER 6

Apps

Included with HELICS are a number of apps that provide useful utilities and test programs for getting started and running with HELICS

6.1 Recorder

The Recorder application is one of the HELICS apps available with the library Its purpose is to provide a easy way to capture data from a federation It acts as a federate that can "capture" values or messages from specific publications or direct endpoints or cloned endpoints which exist elsewhere

6.1.1 Command line arguments

```
allowed options:

command line only:
  -? [ --help ]          produce help message
  -v [ --version ]       display a version string
  --config-file arg      specify a configuration file to use

configuration:
  --local                specify otherwise unspecified endpoints and
                         publications as local( i.e.the keys will be prepended
                         with the player name
  --stop arg             the time to stop the player
  --quiet                turn off most display output

allowed options:

configuration:
  -b [ --broker ] arg    address of the broker to connect
  -n [ --name ] arg      name of the player federate
```

(continues on next page)

(continued from previous page)

```
  --corename arg          the name of the core to create or find
  -c [ --core ] arg       type of the core to connect to
  --offset arg            the offset of the time steps
  --period arg            the period of the federate
  --timedelta arg         the time delta of the federate
  --rttolerance arg       the time tolerance of the real time mode
  -i [ --coreinit ] arg   the core initialization string
  --separator arg         separator character for local federates
  --inputdelay arg        the input delay on incoming communication of the
                          federate
  --outputdelay arg       the output delay for outgoing communication of the
                          federate
  -f [ --flags ] arg      named flag for the federate

allowed options:

configuration:
  --tags arg              tags to record, this argument may be specified any
                          number of times
  --endpoints arg         endpoints to capture, this argument may be specified
                          multiple time
  --sourceclone arg       existing endpoints to capture generated packets from,
                          this argument may be specified multiple time
  --destclone arg         existing endpoints to capture all packets with the
                          specified endpoint as a destination, this argument may
                          be specified multiple time
  --clone arg             existing endpoints to clone all packets to and from
  --capture arg           capture all the publications of a particular federate
                          capture="fed1;fed2"  supports multiple arguments or a
                          semicolon/comma separated list
  -o [ --output ] arg     the output file for recording the data
  --allow_iteration       allow iteration on values
  --verbose               print all value results to the screen
  --marker arg            print a statement indicating time advancement     every <arg>
↪ seconds of the simulation
                          is the period of the marker
  --mapfile arg           write progress to a map file for concurrent progress
                          monitoring
```

also permissible are all arguments allowed for federates and any specific broker specified:

Command line reference

the player executable also takes an untagged argument of a file name for example

```
helics_recorder record_file.txt --stop 5
```

Recorders support both delimited text files and json files some examples can be found in

Player configuration examples

6.1.2 config file detail

subscriptions

a simple example of a recorder file specifying some subscriptions

```
#FederateName topic1

sub pub1
subscription pub2
```

signifies a comment

if only a single column is specified it is assumed to be a subscription

for two column rows the second is the identifier arguments with spaces should be enclosed in quotes

interface	description
s, sub, subscription	subscribe to a particular publication
endpoint, ept, e	generate an endpoint to capture all targeted packets
source, sourceclone,src	capture all messages coming from a particular endpoint
dest, destination, destclone	capture all message going to a particular endpoint
capture	capture all data coming from a particular federate
clone	capture all message going from or to a particular endpoint

for 3 column rows the first must be either clone or capture for clone the second can be either source or destination and the third the endpoint name [for capture it can be either "endpoints" or "subscriptions"] NOTE:not fully working yet for capture

JSON configuration

recorders can also be specified via JSON files

here are two examples of the text format and equivalent JSON

```
#list publications and endpoints for a recorder

pub1
pub2
e src1
```

JSON example

```
{
  "subscriptions": [
    {
      "key": "pub1",
      "type": "double"
    },
    {
      "key": "pub2",
      "type": "double"
    }
  ],
  "endpoints": [
    {
      "name": "src1",
      "global": true
    }
  ]
}
```

some configuration can also be done through JSON through elements of “stop”,”local”,”separator”,”timeunits” and file elements can be used to load up additional files

output

Recorders capture files in a format the Player can read see Player the `--verbose` option will also print the values to the screen.

Map file output

the recorder can generate a live file that can be used in process to see the progress of the Federation This is occasionally useful, though for many uses the Tracer will be more useful when it is completed

6.2 Player

The player application is one of the HELICS apps available with the library Its purpose is to provide a easy way to generate data into a federation It acts as a federate that can “play” values or messages at specific times It exists as a standalone executable but also as library object so could be integrated into other components

6.2.1 Command line arguments

```
command line only:
  -? [ --help ]          produce help message
  -v [ --version ]       display a version string
  --config-file arg      specify a configuration file to use

configuration:
  --local                specify otherwise unspecified endpoints and
                         publications as local( i.e.the keys will be prepended
                         with the player name
  --stop arg             the time to stop the player
  --quiet                turn off most display output

configuration:
  -b [ --broker ] arg    address of the broker to connect
  -n [ --name ] arg      name of the player federate
  --corename arg         the name of the core to create or find
  -c [ --core ] arg      type of the core to connect to
  --offset arg           the offset of the time steps
  --period arg           the period of the federate
  --timedelta arg        the time delta of the federate
  --rttolerance arg      the time tolerance of the real time mode
  -i [ --coreinit ] arg  the core initialization string
  --separator arg        separator character for local federates
  --inputdelay arg       the input delay on incoming communication of the
                         federate
  --outputdelay arg      the output delay for outgoing communication of the
                         federate
  -f [ --flags ] arg     named flag for the federate

allowed options:
```

(continues on next page)

(continued from previous page)

```
configuration:
  --datatype arg          type of the publication data type to use
  --marker arg            print a statement indicating time advancement every  arg␣
↪seconds
                          is the period of the marker
  --time_units arg         the default units on the timestamps used in file based
                          input
```

also permissible are all arguments allowed for federates and any specific broker specified:

Command line reference

the player executable also takes an untagged argument of a file name for example

```
helics_player player_file.txt --stop 5
```

Players support both delimited text files and JSON files some examples can be found in

Player configuration examples

6.2.2 Config File Detail

publications

a simple example of a player file publishing values

```
#second    topic                type(opt)                    value
-1.0, pub1, d, 0.3
1, pub1, 0.5
3, pub1 0.8
2, pub1 0.7
# pub 2
1, pub2, d, 0.4
2, pub2, 0.6
3, pub2, 0.9
4, 0.7  # this statement is assumed to refer to pub 2
```

`#` signifies a comment the first column is time in seconds unless otherwise specified via the `--time_units` flag or other configuration means the second column is publication name the final column is the value the optional third column specifies a type valid types are

time specifications are typically numerical with optional units `5` or `"500 ms"` or `23.7us` if there is a space between the number and units it must be enclosed in quotes if no units are specified the time defaults to units specified via `--time_units` or seconds if none were specified valid units are “s”, “ms”, “us”, “min”, “day”, “hr”, “ns”, “ps” the default precision in HELICS is ns so time specified in ps is not guaranteed to be precise

identifier	type	Example
d,f, double	double	45.1
s,string	string	“this is a test”
i, i64, int	integer	456
c, complex	complex	23+2j, -23.1j, 1+3i
v, vector	vector of doubles	[23.1,34,17.2,-5.6]
cv, complex_vector	vector of complex numbers	[23+2j, -23.1j, 1+3i]

capitalization does not matter

values with times <0 are sent during the initialization phase values with time==0 are sent immediately after entering execution phase

Messages

messages are specified in one of two forms

```
m <time> <source> <dest>  <data>
```

or

```
m <sendtime> <deliverytime> <source> <dest> <time> <data>
```

the second option allows sending events at a different time than they are triggered the data portion of messages can be encoded in base64 by marking as b64[] or base64[X] all data between the brackets will be converted to raw binary. A ‘]’ must be last

JSON configuration

player values can also be specified via JSON files

here are two examples of the text format and equivalent JSON

```
#example player file
mess 1.0 src dest "this is a test message"
mess 1.0 2.0 src dest "this is test message2"
M 2.0 3.0 src dest "this is message 3"
```

JSON example

```
 {
    "messages": [{
            "source": "src",
            "dest": "dest",
            "time": 1.0,
            "data":"this is a test message"
        }, {
              "source": "src",
              "dest": "dest",
              "time": 1.0,
        "encoding":"base64"
              "data
↪":AAECAwQFBgcICQoLDA0ODxAREhMUFRYXGBkaGxwdHh8gISIjJCUmJygpKissLS4vMDEyMzQ1Njc4OTo7PD0+P0BBQkNERUZH
↪wMHCw8TFxsfIycrLzM3Oz9DR0tPU1dbX2Nna29zd3t/g4eLj5OXm5+jp6uvs7e7v8PHy8/T19vf4+fr7/P3+/wABAgME
↪P3+/w=="
          },{
            "source": "src",
            "dest": "dest",
            "time": 2.0,
            "data":"this is test message2"
        }, {
            "source": "src",
            "dest": "dest",
            "time": 3.0,
```

(continues on next page)

(continued from previous page)

```
            "data":"this is message 3"
        }
    ]
}
```

```
#second    topic                type(opt)                    value
-1 pub1 d 0.3
1 pub1 d 0.5
2 pub1 d 0.7
3 pub1 d 0.8
1 pub2 d 0.4
2 pub2 d 0.6
3 pub2 d 0.9
```

Example JSON

```
{
  "points": [
    {
      "key": "pub1",
      "type": "double",
      "value": 0.3,
      "time": -1
    },
    {
      "key": "pub2",
      "type": "double",
      "value": 0.4,
      "time": 1.0
    },
    {
      "key": "pub1",
      "value": 0.5,
      "time": 1.0
    },
    {
      "key": "pub1",
      "value": 0.8,
      "time": 3.0
    },
    {
      "key": "pub1",
      "value": 0.7,
      "time": 2.0
    },
    {
      "key": "pub2",
      "value": 0.6,
      "time": 2.0
    },
    {
      "key": "pub2",
      "value": 0.9,
      "time": 3.0
    }
  ]
}
```

some configuration can also be done through JSON through elements of “stop”,”local”,”separator”,”time_units” and file elements can be used to load up additional files

6.3 Source

The Source app generates signals for other federates, it functions similarly to the player but doesn't take a prescribed file instead it generates signals according to some mathematical function, like sine, ramp, pulse, or random walk. This can be useful for sending probing signals or just testing responses of the federate to various stimuli.

6.3.1 Command line arguments

```
allowed options:

command line only:
  -? [ --help ]          produce help message
  -v [ --version ]       display a version string
  --config-file arg      specify a configuration file to use

configuration:
  --datatype arg         type of the publication data type to use
  --local                specify otherwise unspecified endpoints and
                         publications as local( i.e.the keys will be prepended
                         with the player name
  --separator arg        specify the separator for local publications and
                         endpoints
  --time_units arg        the default units on the timestamps used in file based
                         input
  --stop arg             the time to stop the player

federate configuration
  -b [ --broker ] arg    address of the broker to connect
  -n [ --name ] arg      name of the player federate
  --corename arg         the name of the core to create or find
  -c [ --core ] arg      type of the core to connect to
  --offset arg           the offset of the time steps
  --period arg           the period of the federate
  --timedelta arg        the time delta of the federate
  -i [ --coreinit ] arg  the core initialization string
  --inputdelay arg       the input delay on incoming communication of the
                         federate
  --outputdelay arg      the output delay for outgoing communication of the
                         federate
  -f [ --flags ] arg     named flags for the federate
```

also permissible are all arguments allowed for federates and any specific broker specified:

Command line reference

the player executable also takes an untagged argument of a file name for example

```
helics_player player_file.txt --stop 5
```

Players support both delimited text files and JSON files some examples can be found in

Player configuration examples

6.3.2 Config File Detail

publications

a simple example of a player file publishing values

```
#second    topic                type(opt)                    value
-1.0, pub1, d, 0.3
1, pub1, 0.5
3, pub1 0.8
2, pub1 0.7
# pub 2
1, pub2, d, 0.4
2, pub2, 0.6
3, pub2, 0.9
```

`#` signifies a comment the first column is time in seconds unless otherwise specified via the `--time_units` flag or other configuration means the second column is publication name the final column is the value the optional third column specifies a type valid types are

time specifications are typically numerical with optional units `5` or `"500 ms"` or `23.7us` if there is a space between the number and units it must be enclosed in quotes if no units are specified the time defaults to units specified via `--time_units` or seconds if none were specified valid units are "s", "ms", "us", "min", "day", "hr", "ns", "ps" the default precision in HELICS is ns so time specified in ps is not guaranteed to be precise

identifier	type	Example
d,f, double	double	45.1
s,string	string	"this is a test"
i, i64, int	integer	456
c, complex	complex	23+2j, -23.1j, 1+3i
v, vector	vector of doubles	[23.1,34,17.2,-5.6]
cv, complex_vector	vector of complex numbers	[23+2j, -23.1j, 1+3i]

capitalization does not matter

values with times <0 are sent during the initialization phase values with time==0 are sent immediately after entering execution phase

Messages

messages are specified in one of two forms

```
m <time> <source> <dest>  <data>
```

or

```
m <sendtime> <deliverytime> <source> <dest> <time> <data>
```

the second option allows sending events at a different time than they are triggered

JSON configuration

player values can also be specified via JSON files

here are two examples of the text format and equivalent JSON

```
#example player file
mess 1.0 src dest "this is a test message"
mess 1.0 2.0 src dest "this is test message2"
M 2.0 3.0 src dest "this is message 3"
```

JSON example

```
{
  "messages": [
    {
      "source": "src",
      "dest": "dest",
      "time": 1.0,
      "data": "this is a test message"
    },
    {
      "source": "src",
      "dest": "dest",
      "time": 2.0,
      "data": "this is test message2"
    },
    {
      "source": "src",
      "dest": "dest",
      "time": 3.0,
      "data": "this is message 3"
    }
  ]
}
```

```
#second    topic                type(opt)                    value
-1 pub1 d 0.3
1 pub1 d 0.5
2 pub1 d 0.7
3 pub1 d 0.8
1 pub2 d 0.4
2 pub2 d 0.6
3 pub2 d 0.9
```

Example JSON

```
{
  "points": [
    {
      "key": "pub1",
      "type": "double",
      "value": 0.3,
      "time": -1
    },
    {
      "key": "pub2",
      "type": "double",
      "value": 0.4,
      "time": 1.0
    },
    {
      "key": "pub1",
```

(continues on next page)

(continued from previous page)

```
      "value": 0.5,
      "time": 1.0
    },
    {
      "key": "pub1",
      "value": 0.8,
      "time": 3.0
    },
    {
      "key": "pub1",
      "value": 0.7,
      "time": 2.0
    },
    {
      "key": "pub2",
      "value": 0.6,
      "time": 2.0
    },
    {
      "key": "pub2",
      "value": 0.9,
      "time": 3.0
    }
  ]
}
```

some configuration can also be done through JSON through elements of "stop","local","separator","time_units" and file elements can be used to load up additional files

6.4 helics_app

The HELICS apps executable is one of the HELICS apps available with the library Its purpose is to provide a common executable for running any of the other as

typical syntax is as follows

```
helics-app.exe <app> <app arguments ...>
```

possible apps are

6.4.1 Echo

The Echo app is a responsive app that will echo any message sent to its endpoints back to the original source with a specified delay

This is useful for testing communication pathways and in combination with filters can be used to create some interesting situations

6.4.2 Player

The player app will generate signals through specified interfaces from prescribed data This is used for generating test signals into a federate

6.4.3 Recorder

The Recorder app captures signals and data on specified interfaces and can record then to various file formats including text files and JSON files The files saved can then be used by the Player app at a later time

6.4.4 Tracer

The Tracer app functions much like the recorder when run as a standalone app with the exception that it displays information to a text window and doesn't capture to a file The additional purpose is used as a library object as the basis for additional display purposes and interfaces

6.4.5 Source

The Source app is a signal generator like the player except that is can generate signals from defined patterns including some random signals in value and timing, and other patterns like sine, square wave, ramps and others. Used much like the player in situations some test signals are needed.

6.4.6 Broker

The Broker executes a broker like the stand alone Broker app, it does not include the broker terminal application.

6.4.7 Clone

The Clone has the ability to copy another federate and record it to a file that can be used by a Player. It will duplicate all publications and subscriptions of a federate.

6.4.8 MultiBroker

The Multibroker is an in progress development of a broker that can interact with multiple communication modes. Such as a single broker that can act as a bridge between MPI and ZeroMQ or other network protocols. More documentation will be available as the multibroker is developed

6.5 Command Line Arguments

```
allowed options:

command line only:
  -? [ --help ]          produce help message
  -v [ --version ]       display a version string
  --config-file arg      specify a configuration file to use

configuration:
  -n [ --name ] arg      name of the broker
  -t [ --type ] arg      type of the broker ("(zmq)", "ipc", "test", "mpi",
                         "test", "tcp", "udp")

 Help for Zero MQ Broker:
```

(continues on next page)

(continued from previous page)

```
configuration:
  --interface arg        the local interface to use for the receive ports
  -b [ --broker ] arg    identifier for the broker
  --broker_address arg   location of the broker i.e network address
  --brokerport arg       port number for the broker priority port
  --localport arg        port number for the local receive port
  --port arg             port number for the broker's port
  --portstart arg        starting port for automatic port definitions

 Help for Interprocess Broker:

configuration:
  --queueloc arg         the named location of the shared queue
  -b [ --broker ] arg    identifier for the broker
  --broker_address arg   location of the broker i.e network address
  --brokerinit arg       the initialization string for the broker

 Help for Test Broker:

configuration:
  --brokername arg       identifier for the broker-same as broker
  -b [ --broker ] arg    identifier for the broker
  --broker_address arg   location of the broker i.e network address
  --brokerinit arg       the initialization string for the broker

 Help for UDP Broker:

configuration:
  --interface arg        the local interface to use for the receive ports
  -b [ --broker ] arg    identifier for the broker
  --broker_address arg   location of the broker i.e network address
  --brokerport arg       port number for the broker priority port
  --localport arg        port number for the local receive port
  --port arg             port number for the broker's port
  --portstart arg        starting port for automatic port definitions
```

6.6 Echo

The Echo application is one of the HELICS apps available with the library Its purpose is to provide a easy way to generate an echo response to a message Mainly for testing and demos

6.6.1 Command line arguments

```
allowed options:

command line only:
  -? [ --help ]          produce help message
  -v [ --version ]       display a version string
  --config-file arg      specify a configuration file to use

configuration:
  --local                specify otherwise unspecified endpoints and
```

(continues on next page)

(continued from previous page)

```
                        publications as local( i.e.the keys will be prepended
                        with the echo name
  --stop arg            the time to stop the app

configuration:
  -b [ --broker ] arg   address of the broker to connect
  -n [ --name ] arg     name of the player federate
  --corename arg        the name of the core to create or find
  -c [ --core ] arg     type of the core to connect to
  --offset arg          the offset of the time steps
  --period arg          the period of the federate
  --timedelta arg       the time delta of the federate
  -i [ --coreinit ] arg the core initialization string
  --separator arg       separator character for local federates
  --inputdelay arg      the input delay on incoming communication of the
                        federate
  --outputdelay arg     the output delay for outgoing communication of the
                        federate
  -f [ --flags ] arg    named flag for the federate

configuration:
  --delay arg           the delay with which the echo app will echo message
```

also permissible are all arguments allowed for federates and any specific broker specified:

Command line reference

the echo executable also takes an untagged argument of a file name for example

```
helics_app echo echo_file.txt --stop 5
```

The Echo app supports JSON files some examples can be found in

Echo configuration examples

the main property of the echo app is the delay time which messages are echoed.

6.7 Tracer

The Tracer application is one of the HELICS apps available with the library Its purpose is to provide a easy way to display data from a federation It acts as a federate that can "capture" values or messages from specific publications or direct endpoints or cloned endpoints which exist elsewhere and either trigger callbacks or display it to a screen The main use is a simple visual indicator and a monitoring app

6.7.1 Command line arguments

```
allowed options:

command line only:
  -? [ --help ]         produce help message
  -v [ --version ]      display a version string
  --config-file arg     specify a configuration file to use

configuration:
```

(continues on next page)

(continued from previous page)

```
  --stop arg              the time to stop recording
  --tags arg              tags to record, this argument may be specified any
                          number of times
  --endpoints arg         endpoints to capture, this argument may be specified
                          multiple time
  --sourceclone arg       existing endpoints to capture generated packets from,
                          this argument may be specified multiple time
  --destclone arg         existing endpoints to capture all packets with the
                          specified endpoint as a destination, this argument may
                          be specified multiple time
  --clone arg             existing endpoints to clone all packets to and from
  --capture arg           capture all the publications of a particular federate
                          capture="fed1;fed2"  supports multiple arguments or a
                          semicolon/comma separated list
  -o [ --output ] arg     the output file for recording the data
  --mapfile arg           write progress to a memory mapped file

federate configuration
  -b [ --broker ] arg     address of the broker to connect
  -n [ --name ] arg       name of the player federate
  --corename arg          the name of the core to create or find
  -c [ --core ] arg       type of the core to connect to
  --offset arg            the offset of the time steps
  --period arg            the period of the federate
  --timedelta arg         the time delta of the federate
  -i [ --coreinit ] arg   the core initialization string
  --inputdelay arg        the input delay on incoming communication of the
                          federate
  --outputdelay arg       the output delay for outgoing communication of the
                          federate
  -f [ --flags ] arg      named flags for the federate
```

also permissible are all arguments allowed for federates and any specific broker specified:

Command line reference

the tracer executable also takes an untagged argument of a file name for example

```
helics_app tracer tracer_file.txt --stop 5
```

Tracers support both delimited text files and JSON files some examples can be found in, they are otherwise the same as options for recorders.

Tracer configuration examples

6.7.2 Config File Detail

subscriptions

a simple example of a recorder file specifying some subscriptions

```
#FederateName topic1

sub pub1
subscription pub2
```

signifies a comment

if only a single column is specified it is assumed to be a subscription

for two column rows the second is the identifier arguments with spaces should be enclosed in quotes

interface	description
s, sub, subscription	subscribe to a particular publication
endpoint, ept, e	generate an endpoint to capture all targeted packets
source, sourceclone,src	capture all messages coming from a particular endpoint
dest, destination, destclone	capture all message going to a particular endpoint
capture	capture all data coming from a particular federate
clone	capture all message going from or to a particular endpoint

for 3 column rows the first must be either clone or capture for clone the second can be either source or destination and the third the endpoint name [for capture it can be either "endpoints" or "subscriptions"]

JSON configuration

Tracers can also be specified via JSON files

here are two examples of the text format and equivalent JSON

```
#list publications and endpoints for a recorder

pub1
pub2
e src1
```

JSON example

```
{
  "subscriptions": [
    {
      "key": "pub1",
      "type": "double"
    },
    {
      "key": "pub2",
      "type": "double"
    }
  ],
  "endpoints": [
    {
      "name": "src1",
      "global": true
    }
  ]
}
```

some configuration can also be done through JSON through elements of "stop","local","separator","timeunits" and file elements can be used to load up additional files

6.8 Broker

Brokers function as intermediaries or roots in the HELICS hierarchy The Broker can be run through the helics_broker or via helics-app

6.8.1 Command line arguments

```
helics_broker term <broker args...> will start a broker and open a terminal control␣
↪window for the broker run help in a terminal for more commands
helics_broker --autorestart <broker args ...> will start a continually regenerating␣
↪broker there is a 3 second countdown on broker completion to halt the program via␣
↪ctrl-C
helics_broker <broker args ..> just starts a broker with the given args and waits for␣
↪it to complete
allowed options:

command line only:
  -? [ --help ]          produce help message
  -v [ --version ]       display a version string
  --config-file arg      specify a configuration file to use

configuration:
  -n [ --name ] arg      name of the broker
  -t [ --type ] arg      type of the broker ("(zmq)", "ipc", "test", "mpi","test",
↪"tcp", "udp")

 Help for Zero MQ Broker:
allowed options:

configuration:
  --interface arg        the local interface to use for the receive ports
  -b [ --broker ] arg    identifier for the broker
  --broker_address arg   location of the broker i.e network address
  --brokername arg       the name of the broker
  --local                use local interface(default)
  --ipv4                 use external ipv4 addresses
  --ipv6                 use external ipv6 addresses
  --external             use all external interfaces
  --brokerport arg       port number for the broker priority port
  --localport arg        port number for the local receive port
  --port arg             port number for the broker's port
  --portstart arg        starting port for automatic port definitions

 Help for Interprocess Broker:
allowed options:

configuration:
  --queueloc arg         the named location of the shared queue
  -b [ --broker ] arg    identifier for the broker
  --broker_address arg   location of the broker i.e network address
  --brokerinit arg       the initialization string for the broker

 Help for Test Broker:
allowed options:
```

(continues on next page)

(continued from previous page)

```
configuration:
  --brokername arg       identifier for the broker-same as broker
  -b [ --broker ] arg    identifier for the broker
  --broker_address arg   location of the broker i.e network address
  --brokerinit arg       the initialization string for the broker

 Help for TCP Broker:
allowed options:

configuration:
  --interface arg        the local interface to use for the receive ports
  -b [ --broker ] arg    identifier for the broker
  --broker_address arg   location of the broker i.e network address
  --brokername arg       the name of the broker
  --local                use local interface(default)
  --ipv4                 use external ipv4 addresses
  --ipv6                 use external ipv6 addresses
  --external             use all external interfaces
  --brokerport arg       port number for the broker priority port
  --localport arg        port number for the local receive port
  --port arg             port number for the broker's port
  --portstart arg        starting port for automatic port definitions

 Help for UDP Broker:
allowed options:

configuration:
  --interface arg        the local interface to use for the receive ports
  -b [ --broker ] arg    identifier for the broker
  --broker_address arg   location of the broker i.e network address
  --brokername arg       the name of the broker
  --local                use local interface(default)
  --ipv4                 use external ipv4 addresses
  --ipv6                 use external ipv6 addresses
  --external             use all external interfaces
  --brokerport arg       port number for the broker priority port
  --localport arg        port number for the local receive port
  --port arg             port number for the broker's port
  --portstart arg        starting port for automatic port definitions

Broker Specific options:

configuration:
  --root                 specify whether the broker is a root

configuration:
  -n [ --name ] arg      name of the broker/core
  --federates arg        the minimum number of federates that will be
                         connecting
  --minfed arg           the minimum number of federates that will be
                         connecting
  --maxiter arg          maximum number of iterations
  --logfile arg          the file to log message to
  --loglevel arg         the level which to log the higher this is set to the
                         more gets logs (-1) for no logging
  --fileloglevel arg     the level at which messages get sent to the file
  --consoleloglevel arg  the level at which message get sent to the console
```

(continues on next page)

(continued from previous page)

```
  --minbrokers arg       the minimum number of core/brokers that need to be
                         connected (ignored in cores)
  --identifier arg       name of the core/broker
  --tick arg             number of milliseconds per tick counter if there is no
                         broker communication for 2 ticks then secondary actions
                         are taken (can also be entered as a time like '10s' or '45ms
↪')
  --dumplog              capture a record of all messages and dump a complete log to␣
↪file or console on termination
  --terminate_on_error   Specify that the co-simulation should terminate if any error␣
↪occurs
  --timeout arg          milliseconds to wait for a broker connection (can also
                         be entered as a time like '10s' or '45ms')

  --error_timeout arg    milliseconds to wait before disconnecting after an error
                         (can also be entered as a time like '10s' or '45ms')
```

If the Broker is started with `term` as the first option, a terminal is opened for user entry of commands all command line arguments following term are passed to the broker.

```
starting broker
helics>>help
`quit` -> close the terminal application and wait for broker to finish
`terminate` -> force the broker to stop
`terminate*` -> force the broker to stop and exit application
`help`,`?` -> this help display
`restart` -> restart a completed broker
`status` -> will display the current status of the broker
`info` -> will display info about the broker
`force restart` -> will force terminate a broker and restart it
`query` <queryString> -> will query a broker for <queryString>
`query` <queryTarget> <queryString> -> will query <queryTarget> for <queryString>
helics>>
```

`status` will print out current status of the brokers including counts of federates, brokers, and handles

```
helics>>status
Broker (643204-ibrVd-14EWH-unKfh-hExUP) is connected and is accepting new federates
{"brokers":0,
"federates":0,
"handles":0}
helics>>
```

info prints out name, connection status, and connection information

```
helics>>info
Broker (643204-ibrVd-14EWH-unKfh-hExUP) is connected and is accepting new federates
address=tcp://127.0.0.1:23404
```

The `query` command allows any query to be executed from the command line, `query counts` displays the same count numbers as `status`.

Other available queries are described in Queries.

various restart options are also available, `terminate`, `restart`, `force restart`. And finally `quit` will exit the terminal and wait for the broker to complete. enter `terminate` before quit or `terminate*` to terminate and quit.

6.9 Broker Server

Brokers function as intermediaries or roots in the HELICS hierarchy The broker server is an executable that can be used to automatically generate brokers on an as needed basis and coordinate their control and management. It is considered experimental as version 2.2 only works with the ZMQ core type. Future versions will expand this significantly.

Future plans include expanding to all networking core types (ZMQ, ZMQSS, TCP, TCPSS, UDP, and MPI), expanding the abilities of a terminal program and making a Restful interface to the server and underlying brokers.

6.9.1 Command line arguments

```
The Broker server is a helics broker coordinator that can generate brokers on request
Usage:helics_broker_server [OPTIONS] [config]

Positionals:
  config TEXT                 load a config file for the broker server

Options:
  -h,-?,--help                Print this help message and exit
  -v,--version
  -z,--zmq                    start a broker-server for the zmq comms in helics
  --zmqss                     start a broker-server for the zmq single socket comms␣
↪in helics
  -t,--tcp                    start a broker-server for the tcp comms in helics
  -u,--udp                    start a broker-server for the udp comms in helics
  --mpi                       start a broker-server for the mpi comms in helics
[Option Group: quiet]
  Options:
    --quiet                     silence most print output

helics broker server command line
helics_broker_server [OPTIONS] [SUBCOMMAND]

Options:
  -h,-?,--help                Print this help message and exit
  -v,--version
  --duration TIME=30 minutes  specify the length of time the server should run
[Option Group: quiet]
  Options:
    --quiet                     silence most print output

Subcommands:
  term                        helics_broker_server term will start a broker server␣
↪and open a terminal control window for the broker server, run help in a terminal␣
↪for more commands

helics_broker_server server types starts a broker with the given args and waits for␣
↪it to complete
```

If the Broker_server is started with `term` as the first option, a terminal is opened for user entry of commands all command line arguments following term are passed to the broker.

```
starting broker Server
servers started
helics-broker-server>>help
```

(continues on next page)

(continued from previous page)

```
`quit` -> close the terminal application and wait for broker to finish
`terminate` -> force the broker server to stop
`terminate*` -> force the broker server to stop and all existing brokers to terminate
`help`,`?` -> this help display

helics-broker-server>>
```

more commands will be added in future releases

6.10 Clone

The Clone application is one of the HELICS apps available with the library Its purpose is to provide a easy way to clone a federate for later playback It acts as a federate that can "capture" values or messages from a single federate It also captures the interfaces and subscriptions of a federate and will store those in a configuration file that can be used by the Player. The clone app will try to match the federate being cloned as close as possible in timing of messages and publications and subscriptions. At present it does not match nameless publications or filters.

6.10.1 Command line arguments

```
Helics Clone App
Usage: helics_app clone [OPTIONS]

Command line options for the Clone App
Usage: [OPTIONS] [capture]

Positionals:
  capture TEXT                name of the federate to clone

Options:
  --allow_iteration           allow iteration on values
  -o,--output TEXT=clone.json the output file for recording the data

Options:
  -h,-?,--help                Print this help message and exit
```

also permissible are all arguments allowed for federates and any specific broker specified:

Command line reference

the clone app is accessible through the helics_app

```
helics_app clone fed1 -o fed1.json -stop 10
```

output

The Clone app captures output and configuration in a JSON format the Player can read. All publications of a federate are created as global with the name of the original federate, so a player could be named something else if desired and not impact the transmission.

CHAPTER 7

C API Reference

7.1 Enum

0]a default core type that will default to something available

helics_core_type_http = 12
a core type using http for communication

helics_core_type_inproc = 18
an in process core type for handling communications in shared memory it is pretty similar to the test core but stripped from the "test" components

helics_core_type_interprocess = 4
interprocess uses memory mapped files to transfer data (for use when all federates are on the same machine

helics_core_type_ipc = 5
interprocess uses memory mapped files to transfer data (for use when all federates are on the same machine ipc is the same as /ref helics_core_type_interprocess

helics_core_type_mpi = 2
use MPI for operation on a parallel cluster

helics_core_type_nng = 9
for using the nanomsg communications

helics_core_type_null = 66
an explicit core type that is recognized but explicitly doesn't exist, for testing and a few other assorted reasons

helics_core_type_tcp = 6
use a generic TCP protocol message stream to send messages
11]a single socket version of the TCP core for more easily handling firewalls

helics_core_type_test = 3
use the Test core if all federates are in the same process

helics_core_type_udp = 7
use UDP packets to send the data

helics_core_type_websocket = 14
a core using websockets for communication

helics_core_type_zmq = 1
use the Zero MQ networking protocol
10]single socket version of ZMQ core usually for high fed count on the same system

helics_data_type_any = 25262
open type that can be anything

helics_data_type_boolean = 7
a boolean data type

helics_data_type_complex = 3
a pair of doubles representing a complex number

helics_data_type_complex_vector = 5
a complex vector object

helics_data_type_double = 1
a double precision floating point number

helics_data_type_int = 2
a 64 bit integer

helics_data_type_named_point = 6
a named point consisting of a string and a double

helics_data_type_raw = 25
raw data type

helics_data_type_string = 0
a sequence of characters

helics_data_type_time = 8
time data type

helics_data_type_vector = 4
an array of doubles

helics_error_connection_failure = -2
the operation to connect has failed

helics_error_discard = -5
the input was discarded and not used for some reason

helics_error_execution_failure = -14
the function execution has failed

helics_error_external_type = -203
an unknown non-helics error was produced

helics_error_fatal = -404
global fatal error for federation
-18]insufficient space is available to store requested data
-4]the parameter passed was invalid and unable to be used
-10]the call made was invalid in the present state of the calling object

helics_error_invalid_object = -3
indicator that the object used was not a valid object
-9]error issued when an invalid state transition occurred

helics_error_other = -101
the function produced a helics error of some other type

helics_error_registration_failure = -1
registration has failed
-6]the federate has terminated unexpectedly and the call cannot be completed

helics_filter_type_clone = 5
a filter type that duplicates a message and sends the copy to a different destination

helics_filter_type_custom = 0
a custom filter type that executes a user defined callback

helics_filter_type_delay = 1
a filter type that executes a fixed delay on a message

helics_filter_type_firewall = 6
a customizable filter type that can perform different actions on a message based on firewall like rules

helics_filter_type_random_delay = 2
a filter type that executes a random delay on the messages

helics_filter_type_random_drop = 3
a filter type that randomly drops messages

helics_filter_type_reroute = 4
a filter type that reroutes a message to a different destination than originally specified

helics_flag_delay_init_entry = 45
used to delay a core from entering initialization mode even if it would otherwise be ready

helics_flag_enable_init_entry = 47
used to clear the HELICS_DELAY_INIT_ENTRY flag in cores

helics_flag_forward_compute = 14
flag indicating that a federate performs forward computation and does internal rollback

helics_flag_ignore_time_mismatch_warnings = 67
used to not display warnings on mismatched requested times

helics_flag_interruptible = 2
flag indicating that a federate can be interrupted

helics_flag_observer = 0
flag indicating that a federate is observe only

helics_flag_only_transmit_on_change = 6
flag indicating a federate/interface should only transmit values if they have changed(binary equivalence)

helics_flag_only_update_on_change = 8
flag indicating a federate/interface should only trigger an update if a value has changed (binary equivalence)

helics_flag_realtime = 16
flag indicating that a federate needs to run in real time

helics_flag_restrictive_time_policy = 11
flag indicating a federate should operate on a restrictive time policy, which disallows some 2nd order time evaluation and can be useful for certain types of dependency cycles and update patterns, but generally shouldn't be used as it can lead to some very slow update conditions

helics_flag_rollback = 12
flag indicating that a federate has rollback capability

helics_flag_single_thread_federate = 27
flag indicating that the federate will only interact on a single thread

helics_flag_slow_responding = 29
flag specifying that a federate, core, or broker may be slow to respond to pings If the federate goes offline there is no good way to detect it so use with caution

helics_flag_source_only = 4
flag indicating that a federate/interface is a signal generator only

helics_flag_terminate_on_error = 72
specify that a federate error should terminate the federation

helics_flag_uninterruptible = 1
flag indicating that a federate can only return requested times

helics_flag_wait_for_current_time_update = 10
flag indicating a federate should only grant time if all other federates have already passed the requested time

helics_handle_option_buffer_data = 411
specify that the last data should be buffered and sent on subscriptions after init

helics_handle_option_connection_optional = 402
specify that a connection is NOT required for an interface and will only be made if available no warning will be issues if not available

helics_handle_option_connection_required = 397
specify that a connection is required for an interface and will generate an error if not available

helics_handle_option_ignore_interrupts = 475
specify that an interface does not participate in determining time interrupts

helics_handle_option_ignore_unit_mismatch = 447
specify that the mismatching units should be ignored

helics_handle_option_multiple_connections_allowed = 409
specify that multiple connections are allowed for an interface

helics_handle_option_only_transmit_on_change = 452
specify that an interface will only transmit on change(only applicable to publications)

helics_handle_option_only_update_on_change = 454
specify that an interface will only update if the value has actually changed

helics_handle_option_single_connection_only = 407
specify that only a single connection is allowed for an interface

helics_handle_option_strict_type_checking = 414
specify that the types should be checked strictly for pub/sub and filters

helics_iteration_request_force_iteration
force iteration return when able

helics_iteration_request_iterate_if_needed
only return an iteration if necessary

helics_iteration_request_no_iteration
no iteration is requested

helics_iteration_result_error
there was an error

helics_iteration_result_halted
the federation has halted

helics_iteration_result_iterating
the federate is iterating at current time

helics_iteration_result_next_step
the iterations have progressed to the next time

helics_log_level_connections = 3
summary+ notices about federate and broker connections +messages about network connections

helics_log_level_data = 6
timing+ data transfer notices

helics_log_level_error = 0
only print error level indicators

helics_log_level_interfaces = 4
connections+ interface definitions

helics_log_level_no_print = -1
don't print anything except a few catastrophic errors

helics_log_level_summary = 2
warning errors and summary level information

helics_log_level_timing = 5
interfaces + timing message

helics_log_level_trace = 7
all internal messages

helics_log_level_warning = 1
only print warnings and errors

helics_ok = 0
the function executed successfully

helics_property_int_console_log_level = 274
integer property controlling the log level for file logging in a federate see helics_log_levels

helics_property_int_file_log_level = 272
integer property controlling the log level for file logging in a federate see helics_log_levels

helics_property_int_log_level = 271
integer property controlling the log level in a federate see helics_log_levels

helics_property_int_max_iterations = 259
integer property controlling the maximum number of iterations in a federate

helics_property_time_delta = 137
the property controlling the minimum time delta for a federate

helics_property_time_input_delay = 148
the property controlling input delay for a federate

helics_property_time_offset = 141
the property controlling time offset for the period of federate

helics_property_time_output_delay = 150
the property controlling output delay for a federate

helics_property_time_period = 140
the property controlling the period for a federate

helics_property_time_rt_lag = 143
the property controlling real time lag for a federate the max time a federate can lag real time

helics_property_time_rt_lead = 144
the property controlling real time lead for a federate the max time a federate can be ahead of real time

helics_property_time_rt_tolerance = 145
the property controlling real time tolerance for a federate sets both rt_lag and rt_lead

helics_state_error
error state no core communication is possible but values can be retrieved

helics_state_execution
entered after the enterExectuationState call has returned

helics_state_finalize
the federate has finished executing normally final values may be retrieved

helics_state_initialization
entered after the enterInitializingMode call has returned

helics_state_pending_exec
state pending EnterExecution State

helics_state_pending_finalize
state that the federate is pending a finalize request

helics_state_pending_init
indicator that the federate is pending entry to initialization state

helics_state_pending_iterative_time
state that the federate is pending an iterative time request

helics_state_pending_time
state that the federate is pending a timeRequest

helics_state_startup = 0
when created the federate is in startup state

helics_warning = -8
the function issued a warning of some kind

7.2 Functions

1. `Broker`
2. `Core`
3. `Endpoint`
4. `FederateInfo`
5. `Federate`
6. `Filter`
7. `Input`
8. `Message`
9. `Publication`
10. `Query`

7.2.1 Broker

HELICS_EXPORT void helicsBrokerAddDestinationFilterToEndpoint(helics_broker broker, const c
Link a named filter to a destination endpoint.

Parameters

- `broker`: The broker to generate the connection from.
- `filter`: The name of the filter (cannot be NULL).
- `endpoint`: The name of the endpoint to filter the data going to (cannot be NULL).
- `err`: A helics_error object, can be NULL if the errors are to be ignored.

HELICS_EXPORT void helicsBrokerAddSourceFilterToEndpoint(helics_broker broker, const char
Link a named filter to a source endpoint.

Parameters

- `broker`: The broker to generate the connection from.
- `filter`: The name of the filter (cannot be NULL).
- `endpoint`: The name of the endpoint to filter the data from (cannot be NULL).
- `err`: A helics_error object, can be NULL if the errors are to be ignored.

HELICS_EXPORT helics_broker helicsBrokerClone(helics_broker broker, helics_error * err)
Create a new reference to an existing broker.

This will create a new broker object that references the existing broker it must be freed as well.

Return A new reference to the same broker.

Parameters

- `broker`: An existing helics_broker.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsBrokerDataLink(helics_broker broker, const char * source, const ch
Link a named publication and named input using a broker.

Parameters

- `broker`: The broker to generate the connection from.
- `source`: The name of the publication (cannot be NULL).
- `target`: The name of the target to send the publication data (cannot be NULL).
- `err`: A helics_error object, can be NULL if the errors are to be ignored.

HELICS_EXPORT void helicsBrokerDestroy(helics_broker broker)
Disconnect and free a broker.

HELICS_EXPORT void helicsBrokerDisconnect(helics_broker broker, helics_error * err)
Disconnect a broker.

Parameters

- `broker`: The broker to disconnect.

- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsBrokerFree(helics_broker broker)
Release the memory associated with a broker.

HELICS_EXPORT const char* helicsBrokerGetAddress(helics_broker broker)
Get the network address associated with a broker.

Return A string with the network address of the broker.

Parameters

- `broker`: The broker to query.

HELICS_EXPORT const char* helicsBrokerGetIdentifier(helics_broker broker)
Get an identifier for the broker.

Return A string containing the identifier for the broker.

Parameters

- `broker`: The broker to query.

HELICS_EXPORT helics_bool helicsBrokerIsConnected(helics_broker broker)
Check if a broker is connected.

A connected broker implies it is attached to cores or cores could reach out to communicate.

Return helics_false if not connected.

HELICS_EXPORT helics_bool helicsBrokerIsValid(helics_broker broker)
Check if a broker object is a valid object.

Parameters

- `broker`: The helics_broker object to test.

HELICS_EXPORT void helicsBrokerMakeConnections(helics_broker broker, const char * file, hel
Load a file containing connection information.

Parameters

- `broker`: The broker to generate the connections from.
- `file`: A JSON or TOML file containing connection information.
- `err`: A helics_error object, can be NULL if the errors are to be ignored.

HELICS_EXPORT void helicsBrokerSetGlobal(helics_broker broker, const char * valueName, cons
Set a federation global value.

This overwrites any previous value for this name.

Parameters

- `broker`: The broker to set the global through.
- `valueName`: The name of the global to set.
- `value`: The value of the global.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsBrokerSetLogFile(helics_broker broker, const char * logFileName, l
Set the log file on a broker.

Parameters

- broker: The broker to set the log file for.
- logFileName: The name of the file to log to.
- err: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT helics_bool helicsBrokerWaitForDisconnect(helics_broker broker, int msToWait,
Wait for the broker to disconnect.

Return helics_true if the disconnect was successful, helics_false if there was a timeout.

Parameters

- broker: The broker to wait for.
- msToWait: The time out in millisecond (<0 for infinite timeout).
- err: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsBrokerSetLoggingCallback(helics_broker broker, void(*logger)(int l
Set the logging callback to a broker.

Add a logging callback function to a broker. The logging callback will be called when a message flows into a broker from the core or from a broker.

Parameters

- broker: The broker object in which to set the callback.
- logger: A callback with signature void(int, const char *, const char *, void *); the function arguments are loglevel, an identifier, a message string, and a pointer to user data.
- userdata: A pointer to user data that is passed to the function when executing.
- err: A pointer to an error object for catching errors.

7.2.2 Core

**HELICS_EXPORT void helicsCoreAddDestinationFilterToEndpoint(helics_core core, const char *
**
Link a named filter to a destination endpoint.

Parameters

- core: The core to generate the connection from.
- filter: The name of the filter (cannot be NULL).
- endpoint: The name of the endpoint to filter the data going to (cannot be NULL).
- err: A helics_error object, can be NULL if the errors are to be ignored.

HELICS_EXPORT void helicsCoreAddSourceFilterToEndpoint(helics_core core, const char * filte
Link a named filter to a source endpoint.

Parameters

- `core`: The core to generate the connection from.
- `filter`: The name of the filter (cannot be NULL).
- `endpoint`: The name of the endpoint to filter the data from (cannot be NULL).
- `err`: A helics_error object, can be NULL if the errors are to be ignored.

HELICS_EXPORT helics_core helicsCoreClone(helics_core core, helics_error * err)
Create a new reference to an existing core.

This will create a new broker object that references the existing broker. The new broker object must be freed as well.

Return A new reference to the same broker.

Parameters

- `core`: An existing helics_core.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT helics_bool helicsCoreConnect(helics_core core, helics_error * err)
Connect a core to the federate based on current configuration.

Return helics_false if not connected, helics_true if it is connected.

Parameters

- `core`: The core to connect.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsCoreDataLink(helics_core core, const char * source, const char * t
Link a named publication and named input using a core.

Parameters

- `core`: The core to generate the connection from.
- `source`: The name of the publication (cannot be NULL).
- `target`: The name of the target to send the publication data (cannot be NULL).
- `err`: A helics_error object, can be NULL if the errors are to be ignored.

HELICS_EXPORT void helicsCoreDestroy(helics_core core)
Disconnect and free a core.

HELICS_EXPORT void helicsCoreDisconnect(helics_core core, helics_error * err)
Disconnect a core from the federation.

Parameters

- `core`: The core to query.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsCoreFree(helics_core core)
Release the memory associated with a core.

HELICS_EXPORT const char* helicsCoreGetAddress(helics_core core)
Get the network address associated with a core.

Return A string with the network address of the broker.

Parameters

- `core`: The core to query.

HELICS_EXPORT const char* helicsCoreGetIdentifier(helics_core core)
Get an identifier for the core.

Return A string with the identifier of the core.

Parameters

- `core`: The core to query.

HELICS_EXPORT helics_bool helicsCoreIsConnected(helics_core core)
Check if a core is connected.

A connected core implies it is attached to federates or federates could be attached to it

Return helics_false if not connected, helics_true if it is connected.

HELICS_EXPORT helics_bool helicsCoreIsValid(helics_core core)
Check if a core object is a valid object.

Parameters

- `core`: The helics_core object to test.

HELICS_EXPORT void helicsCoreMakeConnections(helics_core core, const char * file, helics_er
Load a file containing connection information.

Parameters

- `core`: The core to generate the connections from.
- `file`: A JSON or TOML file containing connection information.
- `err`: A helics_error object, can be NULL if the errors are to be ignored.

HELICS_EXPORT helics_filter helicsCoreRegisterCloningFilter(helics_core core, const char *
Create a cloning Filter on the specified core.

Cloning filters copy a message and send it to multiple locations, source and destination can be added through other functions.

Return A helics_filter object.

Parameters

- `core`: The core to register through.
- `name`: The name of the filter (can be NULL).
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT helics_filter helicsCoreRegisterFilter(helics_core core, helics_filter_type t
Create a source Filter on the specified core.

Filters can be created through a federate or a core, linking through a federate allows a few extra features of name matching to function on the federate interface but otherwise equivalent behavior.

Return A helics_filter object.

Parameters

- `core`: The core to register through.
- `type`: The type of filter to create /ref helics_filter_type.
- `name`: The name of the filter (can be NULL).
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT void helicsCoreSetGlobal(helics_core core, const char * valueName, const cha

Set a global value in a core.

This overwrites any previous value for this name.

Parameters

- `core`: The core to set the global through.
- `valueName`: The name of the global to set.
- `value`: The value of the global.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsCoreSetLogFile(helics_core core, const char * logFileName, helics_

Set the log file on a core.

Parameters

- `core`: The core to set the log file for.
- `logFileName`: The name of the file to log to.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsCoreSetReadyToInit(helics_core core, helics_error * err)

Set the core to ready for init.

This function is used for cores that have filters but no federates so there needs to be a direct signal to the core to trigger the federation initialization.

Parameters

- `core`: The core object to enable init values for.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT helics_bool helicsCoreWaitForDisconnect(helics_core core, int msToWait, helic

Wait for the core to disconnect.

Return helics_true if the disconnect was successful, helics_false if there was a timeout.

Parameters

- `core`: The core to wait for.
- `msToWait`: The time out in millisecond (<0 for infinite timeout).
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsCoreSetLoggingCallback(helics_core core, void(*logger)(int logleve
Set the logging callback for a core.

Add a logging callback function to a core. The logging callback will be called when a message flows into a core from the core or from a broker.

Parameters

- `core`: The core object in which to set the callback.
- `logger`: A callback with signature void(int, const char *, const char *, void *); The function arguments are loglevel, an identifier, a message string, and a pointer to user data.
- `userdata`: A pointer to user data that is passed to the function when executing.
- `err`: A pointer to an error object for catching errors.

7.2.3 Endpoint

HELICS_DEPRECATED_EXPORT void helicsEndpointClearMessages(helics_endpoint endpoint)
Clear all message from an endpoint.

Parameters

- `endpoint`: The endpoint object to operate on.

HELICS_EXPORT helics_message_object helicsEndpointCreateMessageObject(helics_endpoint endpo
Create a new empty message object.

The message is empty and isValid will return false since there is no data associated with the message yet.

Return A new helics_message_object.

Parameters

- `endpoint`: The endpoint object to associate the message with.
- `err`: An error object to fill out in case of an error.

HELICS_EXPORT const char* helicsEndpointGetDefaultDestination(helics_endpoint endpoint)
Get the default destination for an endpoint.

Return A string with the default destination.

Parameters

- `endpoint`: The endpoint to set the destination for.

HELICS_EXPORT const char* helicsEndpointGetInfo(helics_endpoint end)
Get the data in the info field of a filter.

Return A string with the info field string.

Parameters

- `end`: The filter to query.

HELICS_DEPRECATED_EXPORT helics_message helicsEndpointGetMessage(helics_endpoint endpoint)
Receive a packet from a particular endpoint.

Return A message object.

Parameters

- `endpoint`: The identifier for the endpoint.

HELICS_EXPORT helics_message_object helicsEndpointGetMessageObject(helics_endpoint endpoint
Receive a packet from a particular endpoint.

Return A message object.

Parameters

- `endpoint`: The identifier for the endpoint.

HELICS_EXPORT const char* helicsEndpointGetName(helics_endpoint endpoint)
Get the name of an endpoint.

Return The name of the endpoint.

Parameters

- `endpoint`: The endpoint object in question.

HELICS_EXPORT int helicsEndpointGetOption(helics_endpoint endpoint, int option)
Set a handle option on an endpoint.

Return the value of the option, for boolean options will be 0 or 1

Parameters

- `end`: The endpoint to modify.
- `option`: Integer code for the option to set /ref helics_handle_options.

HELICS_EXPORT const char* helicsEndpointGetType(helics_endpoint endpoint)
Get the type specified for an endpoint.

Return The defined type of the endpoint.

Parameters

- `endpoint`: The endpoint object in question.

HELICS_EXPORT helics_bool helicsEndpointHasMessage(helics_endpoint endpoint)
Check if a given endpoint has any unread messages.

Return helics_true if the endpoint has a message, helics_false otherwise.

Parameters

- `endpoint`: The endpoint to check.

HELICS_EXPORT helics_bool helicsEndpointIsValid(helics_endpoint endpoint)
Check if an endpoint is valid.

Return helics_true if the Endpoint object represents a valid endpoint.

Parameters

- `endpoint`: The endpoint object to check.

HELICS_EXPORT int helicsEndpointPendingMessages(helics_endpoint endpoint)
Returns the number of pending receives for all endpoints of a particular federate.

Parameters

- `endpoint`: The endpoint to query.

HELICS_EXPORT void helicsEndpointSendEventRaw(helics_endpoint endpoint, const char * dst, c
Send a message at a specific time to the specified destination.

Parameters

- `endpoint`: The endpoint to send the data from.
- `dst`: The target destination. nullptr to use the default destination. "" to use the default destination.
- `data`: The data to send.
- `inputDataLength`: The length of the data to send.
- `time`: The time the message should be sent.
- `err`: A pointer to an error object for catching errors.

HELICS_DEPRECATED_EXPORT void helicsEndpointSendMessage(helics_endpoint endpoint, helics_me
Send a message object from a specific endpoint.

Parameters

- `endpoint`: The endpoint to send the data from.
- `message`: The actual message to send.
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT void helicsEndpointSendMessageObject(helics_endpoint endpoint, helics_message
Send a message object from a specific endpoint.

Parameters

- `endpoint`: The endpoint to send the data from.
- `message`: The actual message to send which will be copied.
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT void helicsEndpointSendMessageObjectZeroCopy(helics_endpoint endpoint, helics
Send a message object from a specific endpoint, the message will not be copied and the message object will no longer be valid after the call.

Parameters

- `endpoint`: The endpoint to send the data from.
- `message`: The actual message to send which will be copied.
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT void helicsEndpointSendMessageRaw(helics_endpoint endpoint, const char * dst,
Send a message to the specified destination.

Parameters

- `endpoint`: The endpoint to send the data from.
- `dst`: The target destination. nullptr to use the default destination. "" to use the default destination.
- `data`: The data to send.
- `inputDataLength`: The length of the data to send.
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT void helicsEndpointSetDefaultDestination(helics_endpoint endpoint, const char
Set the default destination for an endpoint if no other endpoint is given.

Parameters

- `endpoint`: The endpoint to set the destination for.
- `dst`: A string naming the desired default endpoint.
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT void helicsEndpointSetInfo(helics_endpoint endpoint, const char * info, helic
Set the data in the info field for a filter.

Parameters

- `end`: The endpoint to query.
- `info`: The string to set.
- `err`: An error object to fill out in case of an error.

HELICS_EXPORT void helicsEndpointSetOption(helics_endpoint endpoint, int option, int value,
Set a handle option on an endpoint.

Parameters

- `end`: The endpoint to modify.
- `option`: Integer code for the option to set /ref helics_handle_options.
- `value`: The value to set the option to.
- `err`: An error object to fill out in case of an error.

HELICS_EXPORT void helicsEndpointSubscribe(helics_endpoint endpoint, const char * key, heli
Subscribe an endpoint to a publication.

Parameters

- `endpoint`: The endpoint to use.
- `key`: The name of the publication.
- `err`: A pointer to an error object for catching errors.

7.2.4 FederateInfo

HELICS_EXPORT helics_federate_info helicsFederateInfoClone(helics_federate_info fi, helics_
Create a federate info object from an existing one and clone the information.

Return A helics_federate_info object which is a reference to the created object.

Parameters

- `fi`: A federateInfo object to duplicate.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsFederateInfoFree(helics_federate_info fi)
Delete the memory associated with a federate info object.

HELICS_EXPORT void helicsFederateInfoLoadFromArgs(helics_federate_info fi, int argc, const
Load federate info from command line arguments.

Parameters

- `fi`: A federateInfo object.
- `argc`: The number of command line arguments.
- `argv`: An array of strings from the command line.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsFederateInfoSetBroker(helics_federate_info fi, const char * broke
Set the name or connection information for a broker.

This is only used if the core is automatically created, the broker information will be transferred to the core for connection.

Parameters

- `fi`: The federate info object to alter.
- `broker`: A string which defines the connection information for a broker either a name or an address.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsFederateInfoSetBrokerInitString(helics_federate_info fi, const cha
Set the initialization string that a core will pass to a generated broker usually in the form of command line arguments.

Parameters

- `fi`: The federate info object to alter.
- `brokerInit`: A string with command line arguments for a generated broker.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsFederateInfoSetBrokerKey(helics_federate_info fi, const char * bro
Set the key for a broker connection.

This is only used if the core is automatically created, the broker information will be transferred to the core for connection.

Parameters

- `fi`: The federate info object to alter.
- `brokerkey`: A string containing a key for the broker to connect.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsFederateInfoSetBrokerPort(helics_federate_info fi, int brokerPort,
Set the port to use for the broker.

This is only used if the core is automatically created, the broker information will be transferred to the core for connection. This will only be useful for network broker connections.

Parameters

- `fi`: The federate info object to alter.

- brokerPort: The integer port number to use for connection with a broker.
- err: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsFederateInfoSetCoreInitString(helics_federate_info fi, const char

Set the initialization string for the core usually in the form of command line arguments.

Parameters

- fi: The federate info object to alter.
- coreInit: A string containing command line arguments to be passed to the core.
- err: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsFederateInfoSetCoreName(helics_federate_info fi, const char * core

Set the name of the core to link to for a federate.

Parameters

- fi: The federate info object to alter.
- corename: The identifier for a core to link to.
- err: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsFederateInfoSetCoreType(helics_federate_info fi, int coretype, hel

Set the core type by integer code.

Valid values available by definitions in api-data.h.

Parameters

- fi: The federate info object to alter.
- coretype: An numerical code for a core type see /ref helics_core_type.
- err: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsFederateInfoSetCoreTypeFromString(helics_federate_info fi, const c

Set the core type from a string.

Parameters

- fi: The federate info object to alter.
- coretype: A string naming a core type.
- err: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsFederateInfoSetFlagOption(helics_federate_info fi, int flag, helic

Set a flag in the info structure.

Valid flags are available /ref helics_federate_flags.

Parameters

- fi: The federate info object to alter.
- flag: A numerical index for a flag.

- `value`: The desired value of the flag helics_true or helics_false.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsFederateInfoSetIntegerProperty(helics_federate_info fi, int intPro

Set an integer property for a federate.

Set known properties.

Parameters

- `fi`: The federateInfo object to alter.
- `intProperty`: An int identifying the property.
- `propertyValue`: The value to set the property to.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsFederateInfoSetLocalPort(helics_federate_info fi, const char * loc

Set the local port to use.

This is only used if the core is automatically created, the port information will be transferred to the core for connection.

Parameters

- `fi`: The federate info object to alter.
- `localPort`: A string with the port information to use as the local server port can be a number or "auto" or "os_local".
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsFederateInfoSetSeparator(helics_federate_info fi, char separator,

Set the separator character in the info structure.

The separator character is the separation character for local publications/endpoints in creating their global name. For example if the separator character is '/' then a local endpoint would have a globally reachable name of fedName/localName.

Parameters

- `fi`: The federate info object to alter.
- `separator`: The character to use as a separator.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsFederateInfoSetTimeProperty(helics_federate_info fi, int timeProp

Set the output delay for a federate.

Parameters

- `fi`: The federate info object to alter.
- `timeProperty`: An integer representation of the time based property to set see /ref helics_properties.
- `propertyValue`: The value of the property to set the timeProperty to.

- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

7.2.5 Federate

HELICS_EXPORT void helicsFederateAddDependency(helics_federate fed, const char * fedName, h

Add a time dependency for a federate. The federate will depend on the given named federate for time synchronization.

Parameters

- `fed`: The federate to add the dependency for.
- `fedName`: The name of the federate to depend on.
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT void helicsFederateClearMessages(helics_federate fed)

Clear all stored messages from a federate.

This clears messages retrieved through helicsFederateGetMessage or helicsFederateGetMessageObject

Parameters

- `fed`: The federate to clear the message for.

HELICS_EXPORT void helicsFederateClearUpdates(helics_federate fed)

Clear all the update flags from a federates inputs.

Parameters

- `fed`: The value federate object for which to clear update flags.

HELICS_EXPORT helics_federate helicsFederateClone(helics_federate fed, helics_error * err)

Create a new reference to an existing federate.

This will create a new helics_federate object that references the existing federate. The new object must be freed as well.

Return A new reference to the same federate.

Parameters

- `fed`: An existing helics_federate.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT helics_message_object helicsFederateCreateMessageObject(helics_federate fed,

Create a new empty message object.

The message is empty and isValid will return false since there is no data associated with the message yet.

Return A helics_message_object containing the message data.

Parameters

- `fed`: the federate object to associate the message with
- `err`: An error object to fill out in case of an error.

HELICS_EXPORT void helicsFederateDestroy(helics_federate fed)

Disconnect and free a federate.

HELICS_EXPORT void helicsFederateEnterExecutingMode(helics_federate fed, helics_error * er

Request that the federate enter the Execution mode.

This call is blocking until granted entry by the core object. On return from this call the federate will be at time 0. For an asynchronous alternative call see /ref helicsFederateEnterExecutingModeAsync.

Parameters

- `fed`: A federate to change modes.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsFederateEnterExecutingModeAsync(helics_federate fed, helics_error

Request that the federate enter the Execution mode.

This call is non-blocking and will return immediately. Call /ref helicsFederateEnterExecutingModeComplete to finish the call sequence.

Parameters

- `fed`: The federate object to complete the call.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsFederateEnterExecutingModeComplete(helics_federate fed, helics_er

Complete the call to /ref helicsFederateEnterExecutingModeAsync.

Parameters

- `fed`: The federate object to complete the call.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT helics_iteration_result helicsFederateEnterExecutingModeIterative(helics_fede

Request an iterative time.

This call allows for finer grain control of the iterative process than /ref helicsFederateRequestTime. It takes a time and iteration request, and returns a time and iteration status.

Return An iteration structure with field containing the time and iteration status.

Parameters

- `fed`: The federate to make the request of.
- `iterate`: The requested iteration mode.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsFederateEnterExecutingModeIterativeAsync(helics_federate fed, heli

Request an iterative entry to the execution mode.

This call allows for finer grain control of the iterative process than /ref helicsFederateRequestTime. It takes a time and iteration request, and returns a time and iteration status

Parameters

- `fed`: The federate to make the request of.
- `iterate`: The requested iteration mode.

- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT helics_iteration_result helicsFederateEnterExecutingModeIterativeComplete(hel
Complete the asynchronous iterative call into ExecutionMode.

Return An iteration object containing the iteration time and iteration_status.

Parameters

- `fed`: The federate to make the request of.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsFederateEnterInitializingMode(helics_federate fed, helics_error
Enter the initialization state of a federate.

The initialization state allows initial values to be set and received if the iteration is requested on entry to the execution state. This is a blocking call and will block until the core allows it to proceed.

Parameters

- `fed`: The federate to operate on.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsFederateEnterInitializingModeAsync(helics_federate fed, helics_er
Non blocking alternative to *helicsFederateEnterInitializingMode*.

The function helicsFederateEnterInitializationModeFinalize must be called to finish the operation.

Parameters

- `fed`: The federate to operate on.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsFederateEnterInitializingModeComplete(helics_federate fed, helics_
Finalize the entry to initialize mode that was initiated with /ref heliceEnterInitializingModeAsync.

Parameters

- `fed`: The federate desiring to complete the initialization step.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsFederateFinalize(helics_federate fed, helics_error * err)
Finalize the federate. This function halts all communication in the federate and disconnects it from the core.

HELICS_EXPORT void helicsFederateFinalizeAsync(helics_federate fed, helics_error * err)
Finalize the federate in an async call.

HELICS_EXPORT void helicsFederateFinalizeComplete(helics_federate fed, helics_error * err)
Complete the asynchronous finalize call.

HELICS_EXPORT void helicsFederateFree(helics_federate fed)
Release the memory associated with a federate.

HELICS_EXPORT helics_core helicsFederateGetCoreObject(helics_federate fed, helics_error *
Get the core object associated with a federate.

Return A core object, nullptr if invalid.

Parameters

- `fed`: A federate object.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT helics_time helicsFederateGetCurrentTime(helics_federate fed, helics_error
Get the current time of the federate.

Return The current time of the federate.

Parameters

- `fed`: The federate object to query.
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT helics_endpoint helicsFederateGetEndpoint(helics_federate fed, const char * n
Get an endpoint object from a name.

Return A helics_endpoint object. The object will not be valid and err will contain an error code if no endpoint with the specified name exists.

Parameters

- `fed`: The message federate object to use to get the endpoint.
- `name`: The name of the endpoint.
- `err`: The error object to complete if there is an error.

HELICS_EXPORT helics_endpoint helicsFederateGetEndpointByIndex(helics_federate fed, int ind
Get an endpoint by its index, typically already created via registerInterfaces file or something of that nature.

Return A helics_endpoint. It will be NULL if given an invalid index.

Parameters

- `fed`: The federate object in which to create a publication.
- `index`: The index of the publication to get.
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT int helicsFederateGetEndpointCount(helics_federate fed)
Get the number of endpoints in a federate.

Return (-1) if fed was not a valid federate, otherwise returns the number of endpoints.

Parameters

- `fed`: The message federate to query.

HELICS_EXPORT helics_filter helicsFederateGetFilter(helics_federate fed, const char * name,
Get a filter by its name, typically already created via registerInterfaces file or something of that nature.

Return A helics_filter object, the object will not be valid and err will contain an error code if no filter with the specified name exists.

Parameters

- `fed`: The federate object to use to get the filter.

- name: The name of the filter.
- err: The error object to complete if there is an error.

HELICS_EXPORT helics_filter helicsFederateGetFilterByIndex(helics_federate fed, int index,
Get a filter by its index, typically already created via registerInterfaces file or something of that nature.

Return A helics_filter, which will be NULL if an invalid index is given.

Parameters

- fed: The federate object in which to create a publication.
- index: The index of the publication to get.
- err: A pointer to an error object for catching errors.

HELICS_EXPORT int helicsFederateGetFilterCount(helics_federate fed)
Get the number of filters registered through a federate.

Return A count of the number of filters registered through a federate.

Parameters

- fed: The federate object to use to get the filter.

HELICS_EXPORT helics_bool helicsFederateGetFlagOption(helics_federate fed, int flag, helics
Get a flag value for a federate.

Return The value of the flag.

Parameters

- fed: The federate to get the flag for.
- flag: The flag to query.
- err: A pointer to an error object for catching errors.

HELICS_EXPORT helics_input helicsFederateGetInput(helics_federate fed, const char * key, he
Get an input object from a key.

Return A helics_input object, the object will not be valid and err will contain an error code if no input with the specified key exists.

Parameters

- fed: The value federate object to use to get the publication.
- key: The name of the input.
- err: The error object to complete if there is an error.

HELICS_EXPORT helics_input helicsFederateGetInputByIndex(helics_federate fed, int index, he
Get an input by its index, typically already created via registerInterfaces file or something of that nature.

Return A helics_input, which will be NULL if an invalid index.

Parameters

- fed: The federate object in which to create a publication.
- index: The index of the publication to get.
- err: A pointer to an error object for catching errors.

HELICS_EXPORT int helicsFederateGetInputCount(helics_federate fed)
Get the number of subscriptions in a federate.

Return (-1) if fed was not a valid federate otherwise returns the number of subscriptions.

HELICS_EXPORT int helicsFederateGetIntegerProperty(helics_federate fed, int intProperty, he
Get the current value of an integer property (such as a logging level).

Return The value of the property.

Parameters

- `fed`: The federate to get the flag for.
- `intProperty`: A code for the property to set /ref helics_handle_options.
- `err`: A pointer to an error object for catching errors.

HELICS_DEPRECATED_EXPORT helics_message helicsFederateGetMessage(helics_federate fed)
Receive a communication message for any endpoint in the federate.

The return order will be in order of endpoint creation. So all messages that are available for the first endpoint, then all for the second, and so on. Within a single endpoint, the messages are ordered by time, then source_id, then order of arrival.

Return A unique_ptr to a Message object containing the message data.

HELICS_EXPORT helics_message_object helicsFederateGetMessageObject(helics_federate fed)
Receive a communication message for any endpoint in the federate.

The return order will be in order of endpoint creation. So all messages that are available for the first endpoint, then all for the second, and so on. Within a single endpoint, the messages are ordered by time, then source_id, then order of arrival.

Return A helics_message_object which references the data in the message.

HELICS_EXPORT const char* helicsFederateGetName(helics_federate fed)
Get the name of the federate.

Return A pointer to a string with the name.

Parameters

- `fed`: The federate object to query.

HELICS_EXPORT helics_publication helicsFederateGetPublication(helics_federate fed, const ch
Get a publication object from a key.

Return A helics_publication object, the object will not be valid and err will contain an error code if no publication with the specified key exists.

Parameters

- `fed`: The value federate object to use to get the publication.
- `key`: The name of the publication.
- `err`: The error object to complete if there is an error.

HELICS_EXPORT helics_publication helicsFederateGetPublicationByIndex(helics_federate fed, i
Get a publication by its index, typically already created via registerInterfaces file or something of that nature.

Return A helics_publication.

Parameters

- `fed`: The federate object in which to create a publication.
- `index`: The index of the publication to get.
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT int helicsFederateGetPublicationCount(helics_federate fed)
Get the number of publications in a federate.

Return (-1) if fed was not a valid federate otherwise returns the number of publications.

HELICS_EXPORT helics_federate_state helicsFederateGetState(helics_federate fed, helics_erro
Get the current state of a federate.

Return State the resulting state if void return helics_ok.

Parameters

- `fed`: The federate to query.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT helics_input helicsFederateGetSubscription(helics_federate fed, const char *
Get an input object from a subscription target.

Return A helics_input object, the object will not be valid and err will contain an error code if no input with the specified key exists.

Parameters

- `fed`: The value federate object to use to get the publication.
- `key`: The name of the publication that a subscription is targeting.
- `err`: The error object to complete if there is an error.

HELICS_EXPORT helics_time helicsFederateGetTimeProperty(helics_federate fed, int timePrope
Get the current value of a time based property in a federate.

Parameters

- `fed`: The federate query.
- `timeProperty`: The property to query.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsFederateGlobalError(helics_federate fed, int error_code, const cha
Generate a global error from a federate.

A global error halts the co-simulation completely.

Parameters

- `fed`: The federate to create an error in.
- `error_code`: The integer code for the error.
- `error_string`: A string describing the error.

HELICS_EXPORT helics_bool helicsFederateHasMessage(helics_federate fed)
Check if the federate has any outstanding messages.

Return helics_true if the federate has a message waiting, helics_false otherwise.

Parameters

- `fed`: The federate to check.

HELICS_EXPORT helics_bool helicsFederateIsAsyncOperationCompleted(helics_federate fed, heli
Check if the current Asynchronous operation has completed.

Return helics_false if not completed, helics_true if completed.

Parameters

- `fed`: The federate to operate on.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT helics_bool helicsFederateIsValid(helics_federate fed)
Check if a federate_object is valid.

Return helics_true if the federate is a valid active federate, helics_false otherwise

HELICS_EXPORT void helicsFederateLocalError(helics_federate fed, int error_code, const char
Generate a local error in a federate.

This will propagate through the co-simulation but not necessarily halt the co-simulation, it has a similar effect to finalize but does allow some interaction with a core for a brief time.

Parameters

- `fed`: The federate to create an error in.
- `error_code`: The integer code for the error.
- `error_string`: A string describing the error.

HELICS_EXPORT void helicsFederateLogDebugMessage(helics_federate fed, const char * logmessa
Log a debug message through a federate.

Parameters

- `fed`: The federate to log the debug message through.
- `logmessage`: The message to put in the log.
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT void helicsFederateLogErrorMessage(helics_federate fed, const char * logmessa
Log an error message through a federate.

Parameters

- `fed`: The federate to log the error message through.
- `logmessage`: The message to put in the log.
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT void helicsFederateLogInfoMessage(helics_federate fed, const char * logmessag
Log an info message through a federate.

Parameters

- `fed`: The federate to log the info message through.
- `logmessage`: The message to put in the log.
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT void helicsFederateLogLevelMessage(helics_federate fed, int loglevel, const c
Log a message through a federate.

Parameters

- `fed`: The federate to log the message through.
- `loglevel`: The level of the message to log see /ref helics_log_levels.
- `logmessage`: The message to put in the log.
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT void helicsFederateLogWarningMessage(helics_federate fed, const char * logmes
Log a warning message through a federate.

Parameters

- `fed`: The federate to log the warning message through.
- `logmessage`: The message to put in the log.
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT int helicsFederatePendingMessages(helics_federate fed)
Returns the number of pending receives for the specified destination endpoint.

Parameters

- `fed`: The federate to get the number of waiting messages from.

HELICS_EXPORT void helicsFederatePublishJSON(helics_federate fed, const char * json, helics
Publish data contained in a JSON file or string.

Parameters

- `fed`: The value federate object through which to publish the data.
- `json`: The publication file name or literal JSON data string.
- `err`: The error object to complete if there is an error.

HELICS_EXPORT helics_filter helicsFederateRegisterCloningFilter(helics_federate fed, const
Create a cloning Filter on the specified federate.

Cloning filters copy a message and send it to multiple locations, source and destination can be added through other functions.

Return A helics_filter object.

Parameters

- `fed`: The federate to register through.
- `name`: The name of the filter (can be NULL).
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT helics_endpoint helicsFederateRegisterEndpoint(helics_federate fed, const cha
Create an endpoint.

The endpoint becomes part of the federate and is destroyed when the federate is freed so there are no separate free functions for endpoints.

Return An object containing the endpoint. nullptr on failure.

Parameters

- `fed`: The federate object in which to create an endpoint must have been created with helicsCreateMessageFederate or helicsCreateCombinationFederate.
- `name`: The identifier for the endpoint. This will be prepended with the federate name for the global identifier.
- `type`: A string describing the expected type of the publication (may be NULL).
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT helics_filter helicsFederateRegisterFilter(helics_federate fed, helics_filte
Create a source Filter on the specified federate.

Filters can be created through a federate or a core, linking through a federate allows a few extra features of name matching to function on the federate interface but otherwise equivalent behavior

Return A helics_filter object.

Parameters

- `fed`: The federate to register through.
- `type`: The type of filter to create /ref helics_filter_type.
- `name`: The name of the filter (can be NULL).
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT void helicsFederateRegisterFromPublicationJSON(helics_federate fed, const cha
Register the publications via JSON publication string.

This would be the same JSON that would be used to publish data.

Parameters

- `fed`: The value federate object to use to register the publications.
- `json`: The JSON publication string.
- `err`: The error object to complete if there is an error.

HELICS_EXPORT helics_filter helicsFederateRegisterGlobalCloningFilter(helics_federate fed,
Create a global cloning Filter on the specified federate.

Cloning filters copy a message and send it to multiple locations, source and destination can be added through other functions.

Return A helics_filter object.

Parameters

- `fed`: The federate to register through.
- `name`: The name of the filter (can be NULL).
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT helics_endpoint helicsFederateRegisterGlobalEndpoint(helics_federate fed, co

Create an endpoint.

The endpoint becomes part of the federate and is destroyed when the federate is freed so there are no separate free functions for endpoints.

Return An object containing the endpoint. nullptr on failure.

Parameters

- `fed`: The federate object in which to create an endpoint must have been created with helicsCreateMessageFederate or helicsCreateCombinationFederate.
- `name`: The identifier for the endpoint, the given name is the global identifier.
- `type`: A string describing the expected type of the publication (may be NULL).
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT helics_filter helicsFederateRegisterGlobalFilter(helics_federate fed, helics_

Create a global source filter through a federate.

Filters can be created through a federate or a core, linking through a federate allows a few extra features of name matching to function on the federate interface but otherwise equivalent behavior.

Return A helics_filter object.

Parameters

- `fed`: The federate to register through.
- `type`: The type of filter to create /ref helics_filter_type.
- `name`: The name of the filter (can be NULL).
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT helics_publication helicsFederateRegisterGlobalInput(helics_federate fed, co

Register a global named input.

The publication becomes part of the federate and is destroyed when the federate is freed so there are no separate free functions for subscriptions and publications.

Return An object containing the publication.

Parameters

- `fed`: The federate object in which to create a publication.
- `key`: The identifier for the publication.
- `type`: A code identifying the type of the input see /ref helics_data_type for available options.
- `units`: A string listing the units of the subscription maybe NULL.
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT helics_publication helicsFederateRegisterGlobalPublication(helics_federate fe

Register a global named publication with an arbitrary type.

The publication becomes part of the federate and is destroyed when the federate is freed so there are no separate free functions for subscriptions and publications.

Return An object containing the publication.

Parameters

- `fed`: The federate object in which to create a publication.
- `key`: The identifier for the publication.
- `type`: A code identifying the type of the input see /ref helics_data_type for available options.
- `units`: A string listing the units of the subscription (may be NULL).
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT helics_publication helicsFederateRegisterGlobalTypeInput(helics_federate fed,
Register a global publication with an arbitrary type.

The publication becomes part of the federate and is destroyed when the federate is freed so there are no separate free functions for subscriptions and publications.

Return An object containing the publication.

Parameters

- `fed`: The federate object in which to create a publication.
- `key`: The identifier for the publication.
- `type`: A string defining the type of the input.
- `units`: A string listing the units of the subscription maybe NULL.
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT helics_publication helicsFederateRegisterGlobalTypePublication(helics_federat
Register a global publication with a defined type.

The publication becomes part of the federate and is destroyed when the federate is freed so there are no separate free functions for subscriptions and publications.

Return An object containing the publication.

Parameters

- `fed`: The federate object in which to create a publication.
- `key`: The identifier for the publication.
- `type`: A string describing the expected type of the publication.
- `units`: A string listing the units of the subscription (may be NULL).
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT helics_input helicsFederateRegisterInput(helics_federate fed, const char * ke
Register a named input.

The input becomes part of the federate and is destroyed when the federate is freed so there are no separate free functions for subscriptions, inputs, and publications.

Return An object containing the input.

Parameters

- `fed`: The federate object in which to create an input.
- `key`: The identifier for the publication the global input key will be prepended with the federate name.
- `type`: A code identifying the type of the input see /ref helics_data_type for available options.
- `units`: A string listing the units of the input (may be NULL).

- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT void helicsFederateRegisterInterfaces(helics_federate fed, const char * file,
Load interfaces from a file.

Parameters

- `fed`: The federate to which to load interfaces.
- `file`: The name of a file to load the interfaces from either JSON, or TOML.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT helics_publication helicsFederateRegisterPublication(helics_federate fed, cor
Register a publication with a known type.

The publication becomes part of the federate and is destroyed when the federate is freed so there are no separate free functions for subscriptions and publications.

Return An object containing the publication.

Parameters

- `fed`: The federate object in which to create a publication.
- `key`: The identifier for the publication the global publication key will be prepended with the federate name.
- `type`: A code identifying the type of the input see /ref helics_data_type for available options.
- `units`: A string listing the units of the subscription (may be NULL).
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT helics_input helicsFederateRegisterSubscription(helics_federate fed, const ch
sub/pub registration Create a subscription.

The subscription becomes part of the federate and is destroyed when the federate is freed so there are no separate free functions for subscriptions and publications.

Return An object containing the subscription.

Parameters

- `fed`: The federate object in which to create a subscription, must have been created with /ref helicsCreateValueFederate or /ref helicsCreateCombinationFederate.
- `key`: The identifier matching a publication to get a subscription for.
- `units`: A string listing the units of the subscription (may be NULL).
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT helics_input helicsFederateRegisterTypeInput(helics_federate fed, const char
Register an input with a defined type.

The input becomes part of the federate and is destroyed when the federate is freed so there are no separate free functions for subscriptions, inputs, and publications.

Return An object containing the publication.

Parameters

- `fed`: The federate object in which to create an input.

- `key`: The identifier for the input.
- `type`: A string describing the expected type of the input.
- `units`: A string listing the units of the input maybe NULL.
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT helics_publication helicsFederateRegisterTypePublication(helics_federate fed,
Register a publication with a defined type.

The publication becomes part of the federate and is destroyed when the federate is freed so there are no separate free functions for subscriptions and publications.

Return An object containing the publication.

Parameters

- `fed`: The federate object in which to create a publication.
- `key`: The identifier for the publication.
- `type`: A string labeling the type of the publication.
- `units`: A string listing the units of the subscription (may be NULL).
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT helics_time helicsFederateRequestNextStep(helics_federate fed, helics_error
Request the next time step for federate execution.

Feds should have setup the period or minDelta for this to work well but it will request the next time step which is the current time plus the minimum time step.

Return The time granted to the federate, will return helics_time_maxtime if the simulation has terminated or is invalid

Parameters

- `fed`: The federate to make the request of.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT helics_time helicsFederateRequestTime(helics_federate fed, helics_time reques
Request the next time for federate execution.

Return The time granted to the federate, will return helics_time_maxtime if the simulation has terminated or is invalid.

Parameters

- `fed`: The federate to make the request of.
- `requestTime`: The next requested time.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT helics_time helicsFederateRequestTimeAdvance(helics_federate fed, helics_time
Request the next time for federate execution.

Return The time granted to the federate, will return helics_time_maxtime if the simulation has terminated or is invalid

Parameters

- `fed`: The federate to make the request of.
- `timeDelta`: The requested amount of time to advance.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsFederateRequestTimeAsync(helics_federate fed, helics_time requestT

Request the next time for federate execution in an asynchronous call.

Call /ref helicsFederateRequestTimeComplete to finish the call.

Parameters

- `fed`: The federate to make the request of.
- `requestTime`: The next requested time.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT helics_time helicsFederateRequestTimeComplete(helics_federate fed, helics_err

Complete an asynchronous requestTime call.

Return The time granted to the federate, will return helics_time_maxtime if the simulation has terminated.

Parameters

- `fed`: The federate to make the request of.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT helics_time helicsFederateRequestTimeIterative(helics_federate fed, helics_ti

Request an iterative time.

This call allows for finer grain control of the iterative process than /ref helicsFederateRequestTime. It takes a time and and iteration request, and returns a time and iteration status.

Return The granted time, will return helics_time_maxtime if the simulation has terminated along with the appropriate iteration result. This function also returns the iteration specification of the result.

Parameters

- `fed`: The federate to make the request of.
- `requestTime`: The next desired time.
- `iterate`: The requested iteration mode.
- `outIteration`: The iteration specification of the result.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsFederateRequestTimeIterativeAsync(helics_federate fed, helics_time

Request an iterative time through an asynchronous call.

This call allows for finer grain control of the iterative process than /ref helicsFederateRequestTime. It takes a time and iteration request, and returns a time and iteration status. Call /ref helicsFederateRequestTimeIterative-Complete to finish the process.

Parameters

- `fed`: The federate to make the request of.
- `requestTime`: The next desired time.
- `iterate`: The requested iteration mode.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT helics_time helicsFederateRequestTimeIterativeComplete(helics_federate fed, h

Complete an iterative time request asynchronous call.

Return The granted time, will return helics_time_maxtime if the simulation has terminated. This function also returns the iteration specification of the result.

Parameters

- `fed`: The federate to make the request of.
- `outIterate`: The iteration specification of the result.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsFederateSetFlagOption(helics_federate fed, int flag, helics_bool

Set a flag for the federate.

Parameters

- `fed`: The federate to alter a flag for.
- `flag`: The flag to change.
- `flagValue`: The new value of the flag. 0 for false, !=0 for true.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsFederateSetGlobal(helics_federate fed, const char * valueName, co

Set a federation global value through a federate.

This overwrites any previous value for this name.

Parameters

- `fed`: The federate to set the global through.
- `valueName`: The name of the global to set.
- `value`: The value of the global.
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT void helicsFederateSetIntegerProperty(helics_federate fed, int intProperty,

Set an integer based property of a federate.

Parameters

- `fed`: The federate to change the property for.
- `intProperty`: The property to set.
- `propertyVal`: The value of the property.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsFederateSetLogFile(helics_federate fed, const char * logFile, heli

Set the logging file for a federate (actually on the core associated with a federate).

Parameters

- `fed`: The federate to set the log file for.
- `logFile`: The name of the log file.
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT void helicsFederateSetSeparator(helics_federate fed, char separator, helics_e

Set the separator character in a federate.

The separator character is the separation character for local publications/endpoints in creating their global name. For example if the separator character is '/' then a local endpoint would have a globally reachable name of fedName/localName.

Parameters

- `fed`: The federate info object to alter.
- `separator`: The character to use as a separator.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsFederateSetTimeProperty(helics_federate fed, int timeProperty, hel

Set a time based property for a federate.

Parameters

- `fed`: The federate object to set the property for.
- `timeProperty`: A integer code for a time property.
- `time`: The requested value of the property.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

Warning: doxygenfunction: Cannot find function "helicsFederateEnterInitializingModeCompleted" in doxygen xml output for project "helics" from directory: /home/docs/checkouts/readthedocs.org/user_builds/helics/checkouts/latest/build-doxygen/docs/xml

HELICS_EXPORT void helicsFederateSetLoggingCallback(helics_federate fed, void(*logger)(int

Set the logging callback for a federate.

Add a logging callback function to a federate. The logging callback will be called when a message flows into a federate from the core or from a federate.

Parameters

- `fed`: The federate object in which to create a subscription must have been created with helicsCreateValueFederate or helicsCreateCombinationFederate.
- `logger`: A callback with signature void(int, const char *, const char *, void *); The function arguments are loglevel, an identifier string, a message string, and a pointer to user data.
- `userdata`: A pointer to user data that is passed to the function when executing.

- `err`: A pointer to an error object for catching errors.

7.2.6 Filter

HELICS_EXPORT void helicsFilterAddDeliveryEndpoint(helics_filter filt, const char * delive
Add a delivery endpoint to a cloning filter.

All cloned messages are sent to the delivery address(es).

Parameters

- `filt`: The given filter.
- `deliveryEndpoint`: The name of the endpoint to deliver messages to.
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT void helicsFilterAddDestinationTarget(helics_filter filt, const char * dst, h
Add a destination target to a filter.

All messages going to a destination are copied to the delivery address(es).

Parameters

- `filt`: The given filter to add a destination target to.
- `dst`: The name of the endpoint to add as a destination target.
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT void helicsFilterAddSourceTarget(helics_filter filt, const char * source, hel
Add a source target to a filter.

All messages coming from a source are copied to the delivery address(es).

Parameters

- `filt`: The given filter.
- `source`: The name of the endpoint to add as a source target. .
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT const char* helicsFilterGetInfo(helics_filter filt)
Get the data in the info field of a filter.

Return A string with the info field string.

Parameters

- `filt`: The given filter.

HELICS_EXPORT const char* helicsFilterGetName(helics_filter filt)
Get the name of the filter and store in the given string.

get the name of the filter

Return A string with the name of the filter.

Parameters

- `filt`: The given filter.

HELICS_EXPORT int helicsFilterGetOption(helics_filter filt, int option)
Get a handle option for the filter.

Parameters

- `filt`: The given filter to query.
- `option`: The option to query /ref helics_handle_options.

HELICS_EXPORT helics_bool helicsFilterIsValid(helics_filter filt)
Check if a filter is valid.

Return helics_true if the Filter object represents a valid filter.

Parameters

- `filt`: The filter object to check.

HELICS_EXPORT void helicsFilterRemoveDeliveryEndpoint(helics_filter filt, const char * deli
Remove a delivery destination from a cloning filter.

Parameters

- `filt`: The given filter (must be a cloning filter).
- `deliveryEndpoint`: A string with the delivery endpoint to remove.
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT void helicsFilterRemoveTarget(helics_filter filt, const char * target, helics
Remove a destination target from a filter.

Parameters

- `filt`: The given filter.
- `target`: The named endpoint to remove as a target.
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT void helicsFilterSet(helics_filter filt, const char * prop, double val, helic
Set a property on a filter.

Parameters

- `filt`: The filter to modify.
- `prop`: A string containing the property to set.
- `val`: A numerical value for the property.
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT void helicsFilterSetInfo(helics_filter filt, const char * info, helics_error
Set the data in the info field for a filter.

Parameters

- `filt`: The given filter.
- `info`: The string to set.
- `err`: An error object to fill out in case of an error.

HELICS_EXPORT void helicsFilterSetOption(helics_filter filt, int option, int value, helics_
Set the data in the info field for a filter.

Parameters

- `filt`: The given filter.
- `option`: The option to set /ref helics_handle_options.
- `value`: The value of the option commonly 0 for false 1 for true.
- `err`: An error object to fill out in case of an error.

HELICS_EXPORT void helicsFilterSetString(helics_filter filt, const char * prop, const char
Set a string property on a filter.

Parameters

- `filt`: The filter to modify.
- `prop`: A string containing the property to set.
- `val`: A string containing the new value.
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT void helicsFilterSetCustomCallback(helics_filter filter, void(*filtCall)(hel
Set a general callback for a custom filter.

Add a custom filter callback for creating a custom filter operation in the C shared library.

Parameters

- `filter`: The filter object to set the callback for.
- `filtCall`: A callback with signature helics_message_object(helics_message_object, void *); The function arguments are the message to filter and a pointer to user data. The filter should return a new message.
- `userdata`: A pointer to user data that is passed to the function when executing.
- `err`: A pointer to an error object for catching errors.

7.2.7 Input

HELICS_EXPORT void helicsInputAddTarget(helics_input ipt, const char * target, helics_erro
Add a publication to the list of data that an input subscribes to.

Parameters

- `ipt`: The named input to modify.
- `target`: The name of a publication that an input should subscribe to.
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT void helicsInputClearUpdate(helics_input ipt)
Clear the updated flag from an input.

HELICS_EXPORT helics_bool helicsInputGetBoolean(helics_input ipt, helics_error * err)
Get a boolean value from a subscription.

Return A boolean value of current input value.

Parameters

- `ipt`: The input to get the data for.
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT char helicsInputGetChar(helics_input ipt, helics_error * err)
Get a single character value from an input.

Return The resulting character value. NAK (negative acknowledgment) symbol returned on error

Parameters

- `ipt`: The input to get the data for.
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT void helicsInputGetComplex(helics_input ipt, double * real, double * imag, he
Get a pair of double forming a complex number from a subscriptions.

Return a pair of floating point values that represent the real and imag values

Parameters

- `ipt`: The input to get the data for.
- `real`: Memory location to place the real part of a value.
- `imag`: Memory location to place the imaginary part of a value.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function. On error the values will not be altered.

**HELICS_EXPORT helics_complex helicsInputGetComplexObject(helics_input ipt, helics_error *
Get a complex object from an input object.

Return A helics_complex structure with the value.

Parameters

- `ipt`: The input to get the data for.
- `err`: A helics error object, if the object is not empty the function is bypassed otherwise it is filled in if there is an error.

HELICS_EXPORT double helicsInputGetDouble(helics_input ipt, helics_error * err)
Get a double value from a subscription.

Return The double value of the input.

Parameters

- `ipt`: The input to get the data for.
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT const char* helicsInputGetExtractionUnits(helics_input ipt)
Get the units of an input.

The same as helicsInputGetUnits.

Return A void enumeration, helics_ok if everything worked.

Parameters

- `ipt`: The input to query.

HELICS_EXPORT const char* helicsInputGetInfo(helics_input inp)
Get the data in the info field of an input.

Return A string with the info field string.

Parameters

- `inp`: The input to query.

HELICS_EXPORT const char* helicsInputGetInjectionUnits(helics_input ipt)
Get the units of the publication that an input is linked to.

Return A void enumeration, helics_ok if everything worked.

Parameters

- `ipt`: The input to query.

HELICS_EXPORT int64_t helicsInputGetInteger(helics_input ipt, helics_error * err)
Get an integer value from a subscription.

Return An int64_t value with the current value of the input.

Parameters

- `ipt`: The input to get the data for.
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT const char* helicsInputGetKey(helics_input ipt)
Get the key of an input.

Return A void enumeration, helics_ok if everything worked.

Parameters

- `ipt`: The input to query.

HELICS_EXPORT void helicsInputGetNamedPoint(helics_input ipt, char * outputString, int maxS
Get a named point from a subscription.

Return a string and a double value for the named point

Parameters

- `ipt`: The input to get the result for.
- `outputString`: Storage for copying a null terminated string.
- `maxStringLength`: The maximum size of information that str can hold.
- `actualLength`: The actual length of the string
- `val`: The double value for the named point.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT int helicsInputGetOption(helics_input inp, int option)
Get the current value of an input handle option

Return An integer value with the current value of the given option.

Parameters

- `inp`: The input to query.
- `option`: Integer representation of the option in question see /ref helics_handle_options.

HELICS_EXPORT const char* helicsInputGetPublicationType(helics_input ipt)
Get the type the publisher to an input is sending.

Return A const char * with the type name.

Parameters

- `ipt`: The input to query.

HELICS_EXPORT void helicsInputGetRawValue(helics_input ipt, void * data, int maxDataLength,
Get the raw data for the latest value of a subscription.

Return Raw string data.

Parameters

- `ipt`: The input to get the data for.
- `data`: The memory location of the data
- `maxDataLength`: The maximum size of information that data can hold.
- `actualSize`: The actual length of data copied to data.
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT int helicsInputGetRawValueSize(helics_input ipt)
Get the size of the raw value for subscription.

Return The size of the raw data/string in bytes.

HELICS_EXPORT void helicsInputGetString(helics_input ipt, char * outputString, int maxStrin
Get a string value from a subscription.

Return A string data

Parameters

- `ipt`: The input to get the data for.
- `outputString`: Storage for copying a null terminated string.
- `maxStringLength`: The maximum size of information that str can hold.
- `actualLength`: The actual length of the string.
- `err`: Error term for capturing errors.

HELICS_EXPORT int helicsInputGetStringSize(helics_input ipt)
Get the size of a value for subscription assuming return as a string.

Return The size of the string.

HELICS_EXPORT helics_time helicsInputGetTime(helics_input ipt, helics_error * err)
Get a time value from a subscription.

Return The resulting time value.

Parameters

- `ipt`: The input to get the data for.
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT const char* helicsInputGetType(helics_input ipt)
Get the type of an input.

Return A void enumeration, helics_ok if everything worked.

Parameters

- `ipt`: The input to query.

HELICS_EXPORT const char* helicsInputGetUnits(helics_input ipt)
Get the units of an input.

Return A void enumeration, helics_ok if everything worked.

Parameters

- `ipt`: The input to query.

HELICS_EXPORT void helicsInputGetVector(helics_input ipt, double data[], int maxLength, int
Get a vector from a subscription.

Return a list of floating point values

Parameters

- `ipt`: The input to get the result for.
- `data`: The location to store the data.
- `maxLength`: The maximum size of the vector.
- `actualSize`: Location to place the actual length of the resulting vector.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT int helicsInputGetVectorSize(helics_input ipt)
Get the size of a value for subscription assuming return as an array of doubles.

Return The number of doubles in a returned vector.

HELICS_EXPORT helics_bool helicsInputIsUpdated(helics_input ipt)
Check if a particular subscription was updated.

Return helics_true if it has been updated since the last value retrieval.

HELICS_EXPORT helics_bool helicsInputIsValid(helics_input ipt)
Check if an input is valid.

Return helics_true if the Input object represents a valid input.

Parameters

- `ipt`: The input to check.

HELICS_EXPORT helics_time helicsInputLastUpdateTime(helics_input ipt)
Get the last time a subscription was updated.

HELICS_EXPORT void helicsInputSetDefaultBoolean(helics_input ipt, helics_bool val, helics_e
Set the default as a boolean.

Parameters

- `ipt`: The input to set the default for.
- `val`: The default boolean value.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsInputSetDefaultChar(helics_input ipt, char val, helics_error * e

Set the default as a char.

Parameters

- ipt: The input to set the default for.
- val: The default char value.
- err: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsInputSetDefaultComplex(helics_input ipt, double real, double imag,

Set the default as a complex number.

Parameters

- ipt: The input to set the default for.
- real: The default real value.
- imag: The default imaginary value.
- err: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsInputSetDefaultDouble(helics_input ipt, double val, helics_error

Set the default as a double.

Parameters

- ipt: The input to set the default for.
- val: The default double value.
- err: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsInputSetDefaultInteger(helics_input ipt, int64_t val, helics_erro

Set the default as an integer.

Parameters

- ipt: The input to set the default for.
- val: The default integer.
- err: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsInputSetDefaultNamedPoint(helics_input ipt, const char * str, doub

Set the default as a NamedPoint.

Parameters

- ipt: The input to set the default for.
- str: A pointer to a string representing the name.
- val: A double value for the value of the named point.
- err: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsInputSetDefaultRaw(helics_input ipt, const void * data, int inputL
Set the default as a raw data array.

Parameters

- `ipt`: The input to set the default for.
- `data`: A pointer to the raw data to use for the default.
- `inputDataLength`: The size of the raw data.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsInputSetDefaultString(helics_input ipt, const char * str, helics_e
Set the default as a string.

Parameters

- `ipt`: The input to set the default for.
- `str`: A pointer to the default string.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsInputSetDefaultTime(helics_input ipt, helics_time val, helics_erro
Set the default as a time.

Parameters

- `ipt`: The input to set the default for.
- `val`: The default time value.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsInputSetDefaultVector(helics_input ipt, const double * vectorInput
Set the default as a vector of doubles.

Parameters

- `ipt`: The input to set the default for.
- `vectorInput`: A pointer to an array of double data.
- `vectorLength`: The number of points to publish.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsInputSetInfo(helics_input inp, const char * info, helics_error *e
Set the data in the info field for an input.

Parameters

- `inp`: The input to query.
- `info`: The string to set.
- `err`: An error object to fill out in case of an error.

HELICS_EXPORT void helicsInputSetMinimumChange(helics_input inp, double tolerance, helics_e
Set the minimum change detection tolerance.

Parameters

- `inp`: The input to modify.
- `tolerance`: The tolerance level for registering an update, values changing less than this value will not show as being updated.
- `err`: An error object to fill out in case of an error.

HELICS_EXPORT void helicsInputSetOption(helics_input inp, int option, int value, helics_er
Set an option on an input

Parameters

- `inp`: The input to query.
- `option`: The option to set for the input /ref helics_handle_options.
- `value`: The value to set the option to.
- `err`: An error object to fill out in case of an error.

7.2.8 Message

HELICS_EXPORT void helicsMessageAppendData(helics_message_object message, const void * data
Append data to the payload.

Parameters

- `message`: The message object in question.
- `data`: A string containing the message data to append.
- `inputDataLength`: The length of the data to input.
- `err`: An error object to fill out in case of an error.

HELICS_EXPORT helics_bool helicsMessageCheckFlag(helics_message_object message, int flag)
Check if a flag is set on a message.

Return The flags associated with a message.

Parameters

- `message`: The message object in question.
- `flag`: The flag to check should be between [0,15].

HELICS_EXPORT void helicsMessageClearFlags(helics_message_object message)
Clear the flags of a message.

Parameters

- `message`: The message object in question

HELICS_EXPORT helics_message_object helicsMessageClone(helics_message_object message, helic
Clone a message object.

Parameters

- `message`: The message object to copy from.
- `err`: An error object to fill out in case of an error.

HELICS_EXPORT void helicsMessageCopy(helics_message_object src_message, helics_message_obje
Copy a message object.

Parameters

- `src_message`: The message object to copy from.
- `dst_message`: The message object to copy to.
- `err`: An error object to fill out in case of an error.

HELICS_EXPORT void helicsMessageFree(helics_message_object message)
Free a message object from memory memory for message is managed so not using this function does not create memory leaks, this is an indication to the system that the memory for this message is done being used and can be reused for a new message. *helicsFederateClearMessages()* can also be used to clear up all stored messages at once

Parameters

- `message`: The message object to copy from.

HELICS_EXPORT const char* helicsMessageGetDestination(helics_message_object message)
Get the destination endpoint of a message.

Return A string with the destination endpoint.

Parameters

- `message`: The message object in question.

HELICS_EXPORT int helicsMessageGetMessageID(helics_message_object message)
Get the messageID of a message.

Return The messageID.

Parameters

- `message`: The message object in question.

HELICS_EXPORT const char* helicsMessageGetOriginalDestination(helics_message_object message
Get the original destination endpoint of a message, the destination may have been modified by filters or other actions.

Return A string with the original destination of a message.

Parameters

- `message`: The message object in question.

HELICS_EXPORT const char* helicsMessageGetOriginalSource(helics_message_object message)
Get the original source endpoint of a message, the source may have been modified by filters or other actions.

Return A string with the source of a message.

Parameters

- `message`: The message object in question.

HELICS_EXPORT void helicsMessageGetRawData(helics_message_object message, void * data, int
Get the raw data for a message object.

Return Raw string data.

Parameters

- message: A message object to get the data for.
- data: The memory location of the data.
- maxMessageLength: The maximum size of information that data can hold.
- actualSize: The actual length of data copied to data.
- err: A pointer to an error object for catching errors.

HELICS_EXPORT int helicsMessageGetRawDataSize(helics_message_object message)
Get the size of the data payload in bytes.

Return The size of the data payload.

Parameters

- message: The message object in question.

HELICS_EXPORT const char* helicsMessageGetSource(helics_message_object message)
Get the source endpoint of a message.

Return A string with the source endpoint.

Parameters

- message: The message object in question.

HELICS_EXPORT const char* helicsMessageGetString(helics_message_object message)
Get the payload of a message as a string.

Return A string representing the payload of a message.

Parameters

- message: The message object in question.

HELICS_EXPORT helics_time helicsMessageGetTime(helics_message_object message)
Get the helics time associated with a message.

Return The time associated with a message.

Parameters

- message: The message object in question.

HELICS_EXPORT helics_bool helicsMessageIsValid(helics_message_object message)
A check if the message contains a valid payload.

Return helics_true if the message contains a payload.

Parameters

- message: The message object in question.

HELICS_EXPORT void helicsMessageReserve(helics_message_object message, int reserveSize, hel
Reserve space in a buffer but don't actually resize.

The message data buffer will be reserved but not resized.

Parameters

- message: The message object in question.
- reserveSize: The number of bytes to reserve in the message object.

- err: An error object to fill out in case of an error.

HELICS_EXPORT void helicsMessageSetData(helics_message_object message, const void * data, i

Set the data payload of a message as raw data.

Parameters

- message: The message object in question.
- data: A string containing the message data.
- inputDataLength: The length of the data to input.
- err: An error object to fill out in case of an error.

HELICS_EXPORT void helicsMessageSetDestination(helics_message_object message, const char *

Set the destination of a message.

Parameters

- message: The message object in question.
- dst: A string containing the new destination.
- err: An error object to fill out in case of an error.

HELICS_EXPORT void helicsMessageSetFlagOption(helics_message_object message, int flag, heli

Set a flag on a message.

Parameters

- message: The message object in question.
- flag: An index of a flag to set on the message.
- flagValue: The desired value of the flag.
- err: An error object to fill out in case of an error.

HELICS_EXPORT void helicsMessageSetMessageID(helics_message_object message, int32_t message

Set the message ID for the message.

Normally this is not needed and the core of HELICS will adjust as needed.

Parameters

- message: The message object in question.
- messageID: A new message ID.
- err: An error object to fill out in case of an error.

HELICS_EXPORT void helicsMessageSetOriginalDestination(helics_message_object message, const

Set the original destination of a message.

Parameters

- message: The message object in question.
- dst: A string containing the new original source.
- err: An error object to fill out in case of an error.

HELICS_EXPORT void helicsMessageSetOriginalSource(helics_message_object message, const char

Set the original source of a message.

Parameters

- `message`: The message object in question.
- `src`: A string containing the new original source.
- `err`: An error object to fill out in case of an error.

HELICS_EXPORT void helicsMessageSetSource(helics_message_object message, const char * src,
Set the source of a message.

Parameters

- `message`: The message object in question.
- `src`: A string containing the source.
- `err`: An error object to fill out in case of an error.

HELICS_EXPORT void helicsMessageSetString(helics_message_object message, const char * str,
Set the data payload of a message as a string.

Parameters

- `message`: The message object in question.
- `str`: A string containing the message data.
- `err`: An error object to fill out in case of an error.

HELICS_EXPORT void helicsMessageSetTime(helics_message_object message, helics_time time, he
Set the delivery time for a message.

Parameters

- `message`: The message object in question.
- `time`: The time the message should be delivered.
- `err`: An error object to fill out in case of an error.

HELICS_EXPORT void* helicsMessageGetRawDataPointer(helics_message_object message)
Get a pointer to the raw data of a message.

Return A pointer to the raw data in memory, the pointer may be NULL if the message is not a valid message.

Parameters

- `message`: A message object to get the data for.

Warning: doxygenfunction: Cannot find function "helicsMessageSetOrginalSource" in doxygen xml output for project "helics" from directory: /home/docs/checkouts/readthedocs.org/user_builds/helics/checkouts/latest/build-doxygen/docs/xml

HELICS_EXPORT void helicsMessageResize(helics_message_object message, int newSize, helics_e
Resize the data buffer for a message.

The message data buffer will be resized. There are no guarantees on what is in the buffer in newly allocated space. If the allocated space is not sufficient new allocations will occur.

Parameters

- `message`: The message object in question.
- `newSize`: The new size in bytes of the buffer.
- `err`: An error object to fill out in case of an error.

7.2.9 Publication

HELICS_EXPORT void helicsPublicationAddTarget(helics_publication pub, const char * target,
Add a named input to the list of targets a publication publishes to.

Parameters

- `pub`: The publication to add the target for.
- `target`: The name of an input that the data should be sent to.
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT const char* helicsPublicationGetInfo(helics_publication pub)
Get the data in the info field of an publication.

Return A string with the info field string.

Parameters

- `pub`: The publication to query.

HELICS_EXPORT const char* helicsPublicationGetKey(helics_publication pub)
Get the key of a publication.

This will be the global key used to identify the publication to the federation.

Return A void enumeration, helics_ok if everything worked.

Parameters

- `pub`: The publication to query.

HELICS_EXPORT int helicsPublicationGetOption(helics_publication pub, int option)
Get the value of an option for a publication

Return A string with the info field string.

Parameters

- `pub`: The publication to query.
- `option`: The value to query see /ref helics_handle_options.

HELICS_EXPORT const char* helicsPublicationGetType(helics_publication pub)
Get the type of a publication.

Return A void enumeration, helics_ok if everything worked.

Parameters

- `pub`: The publication to query.

HELICS_EXPORT const char* helicsPublicationGetUnits(helics_publication pub)
Get the units of a publication.

Return A void enumeration, helics_ok if everything worked.

Parameters

- pub: The publication to query.

HELICS_EXPORT helics_bool helicsPublicationIsValid(helics_publication pub)
Check if a publication is valid.

Return helics_true if the publication is a valid publication.

Parameters

- pub: The publication to check.

HELICS_EXPORT void helicsPublicationPublishBoolean(helics_publication pub, helics_bool val,
Publish a Boolean Value.

Parameters

- pub: The publication to publish for.
- val: The boolean value to publish.
- err: A pointer to an error object for catching errors.

HELICS_EXPORT void helicsPublicationPublishChar(helics_publication pub, char val, helics_er
Publish a single character.

Parameters

- pub: The publication to publish for.
- val: The numerical value to publish.
- err: A pointer to an error object for catching errors.

HELICS_EXPORT void helicsPublicationPublishComplex(helics_publication pub, double real, dou
Publish a complex value (or pair of values).

Parameters

- pub: The publication to publish for.
- real: The real part of a complex number to publish.
- imag: The imaginary part of a complex number to publish.
- err: A pointer to an error object for catching errors.

HELICS_EXPORT void helicsPublicationPublishDouble(helics_publication pub, double val, helic
Publish a double floating point value.

Parameters

- pub: The publication to publish for.
- val: The numerical value to publish.
- err: A pointer to an error object for catching errors.

HELICS_EXPORT void helicsPublicationPublishInteger(helics_publication pub, int64_t val, hel
Publish an integer value.

Parameters

- pub: The publication to publish for.

- `val`: The numerical value to publish.
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT void helicsPublicationPublishNamedPoint(helics_publication pub, const char * Publish a named point.

Parameters

- `pub`: The publication to publish for.
- `str`: A string for the name to publish.
- `val`: A double for the value to publish.
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT void helicsPublicationPublishRaw(helics_publication pub, const void * data, Publish raw data from a char * and length.

Parameters

- `pub`: The publication to publish for.
- `data`: A pointer to the raw data.
- `inputDataLength`: The size in bytes of the data to publish.
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT void helicsPublicationPublishString(helics_publication pub, const char * str, Publish a string.

Parameters

- `pub`: The publication to publish for.
- `str`: The string to publish.
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT void helicsPublicationPublishTime(helics_publication pub, helics_time val, he Publish a time value.

Parameters

- `pub`: The publication to publish for.
- `val`: The numerical value to publish.
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT void helicsPublicationPublishVector(helics_publication pub, const double * ve Publish a vector of doubles.

Parameters

- `pub`: The publication to publish for.
- `vectorInput`: A pointer to an array of double data.
- `vectorLength`: The number of points to publish.
- `err`: A pointer to an error object for catching errors.

HELICS_EXPORT void helicsPublicationSetInfo(helics_publication pub, const char * info, heli
Set the data in the info field for a publication.

Parameters

- `pub`: The publication to set the info field for.
- `info`: The string to set.
- `err`: An error object to fill out in case of an error.

HELICS_EXPORT void helicsPublicationSetMinimumChange(helics_publication pub, double toleran
Set the minimum change detection tolerance.

Parameters

- `pub`: The publication to modify.
- `tolerance`: The tolerance level for publication, values changing less than this value will not be published.
- `err`: An error object to fill out in case of an error.

HELICS_EXPORT void helicsPublicationSetOption(helics_publication pub, int option, int val,
Set the value of an option for a publication

Parameters

- `pub`: The publication to query.
- `option`: Integer code for the option to set /ref helics_handle_options.
- `val`: The value to set the option to.
- `err`: An error object to fill out in case of an error.

7.2.10 Query

HELICS_EXPORT const char* helicsQueryBrokerExecute(helics_query query, helics_broker broke
Execute a query directly on a broker.

The call will block until the query finishes which may require communication or other delays.

Return A pointer to a string. The string will remain valid until the query is freed or executed again. The return will be nullptr if broker or query is an invalid object, the return string will be "#invalid" if the query itself was invalid

Parameters

- `query`: The query object to use in the query.
- `broker`: The broker to send the query to.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT const char* helicsQueryCoreExecute(helics_query query, helics_core core, heli
Execute a query directly on a core.

The call will block until the query finishes which may require communication or other delays.

Return A pointer to a string. The string will remain valid until the query is freed or executed again. The return will be nullptr if core or query is an invalid object, the return string will be "#invalid" if the query itself was invalid.

Parameters

- `query`: The query object to use in the query.
- `core`: The core to send the query to.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT const char* helicsQueryExecute(helics_query query, helics_federate fed, helic

Execute a query.

The call will block until the query finishes which may require communication or other delays.

Return A pointer to a string. The string will remain valid until the query is freed or executed again. The return will be nullptr if fed or query is an invalid object, the return string will be "#invalid" if the query itself was invalid.

Parameters

- `query`: The query object to use in the query.
- `fed`: A federate to send the query through.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsQueryExecuteAsync(helics_query query, helics_federate fed, helics_

Execute a query in a non-blocking call.

Parameters

- `query`: The query object to use in the query.
- `fed`: A federate to send the query through.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT const char* helicsQueryExecuteComplete(helics_query query, helics_error * e

Complete the return from a query called with /ref helicsExecuteQueryAsync.

The function will block until the query completes /ref isQueryComplete can be called to determine if a query has completed or not.

Return A pointer to a string. The string will remain valid until the query is freed or executed again. The return will be nullptr if query is an invalid object

Parameters

- `query`: The query object to complete execution of.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsQueryFree(helics_query query)

Free the memory associated with a query object.

HELICS_EXPORT helics_bool helicsQueryIsCompleted(helics_query query)
Check if an asynchronously executed query has completed.

This function should usually be called after a QueryExecuteAsync function has been called.

Return Will return helics_true if an asynchronous query has completed or a regular query call was made with a result, and false if an asynchronous query has not completed or is invalid

Parameters

- `query`: The query object to check if completed.

Warning: doxygenfunction: Cannot find function "helicsQueryExecuteCompleted" in doxygen xml output for project "helics" from directory: /home/docs/checkouts/readthedocs.org/user_builds/helics/checkouts/latest/build-doxygen/docs/xml

HELICS_EXPORT void helicsQuerySetTarget(helics_query query, const char * target, helics_er
Update the target of a query.

Parameters

- `query`: The query object to change the target of.
- `target`: the name of the target to query
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT void helicsQuerySetQueryString(helics_query query, const char * queryString,
Update the queryString of a query.

Parameters

- `query`: The query object to change the target of.
- `queryString`: the new queryString
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

7.2.11 Others

HELICS_EXPORT void helicsCleanupLibrary(void)
Function to do some housekeeping work.

This runs some cleanup routines and tries to close out any residual thread that haven't been shutdown yet.

HELICS_EXPORT void helicsCloseLibrary(void)
Call when done using the helics library. This function will ensure the threads are closed properly. If possible this should be the last call before exiting.

HELICS_EXPORT helics_broker helicsCreateBroker(const char * type, const char * name, const
Create a broker object.

Return A helics_broker object. It will be NULL if there was an error indicated in the err object.

Parameters

- `type`: The type of the broker to create.

- name: The name of the broker. It can be a nullptr or empty string to have a name automatically assigned.
- initString: An initialization string to send to the core-the format is similar to command line arguments. Typical options include a broker address such as broker="XSSAF" if this is a subbroker, or the number of federates, or the address.
- err: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT helics_broker helicsCreateBrokerFromArgs(const char * type, const char * name

Create a core object by passing command line arguments.

Return A helics_core object.

Parameters

- type: The type of the core to create.
- name: The name of the core. It can be a nullptr or empty string to have a name automatically assigned.
- argc: The number of arguments.
- argv: The list of string values from a command line.
- err: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT helics_federate helicsCreateCombinationFederate(const char * fedName, helics_

Create a combination federate from a federate info object.

Combination federates are both value federates and message federates, objects can be used in all functions that take a helics_federate, helics_message_federate or helics_federate object as an argument

Return An opaque value federate object nullptr if the object creation failed.

Parameters

- fedName: A string with the name of the federate, can be NULL or an empty string to pull the default name from fi.
- fi: The federate info object that contains details on the federate.
- err: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT helics_federate helicsCreateCombinationFederateFromConfig(const char * confi

Create a combination federate from a JSON file or JSON string or TOML file.

Combination federates are both value federates and message federates, objects can be used in all functions that take a helics_federate, helics_message_federate or helics_federate object as an argument

Return An opaque combination federate object.

Parameters

- configFile: A JSON file or a JSON string or TOML file that contains setup and configuration information.
- err: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT helics_core helicsCreateCore(const char * type, const char * name, const cha

Create a core object.

Return A helics_core object. If the core is invalid, err will contain the corresponding error message and the returned object will be NULL.

Parameters

- `type`: The type of the core to create.
- `name`: The name of the core. It can be a nullptr or empty string to have a name automatically assigned.
- `initString`: An initialization string to send to the core. The format is similar to command line arguments. Typical options include a broker name, the broker address, the number of federates, etc.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT helics_core helicsCreateCoreFromArgs(const char * type, const char * name, in

Create a core object by passing command line arguments.

Return A helics_core object.

Parameters

- `type`: The type of the core to create.
- `name`: The name of the core. It can be a nullptr or empty string to have a name automatically assigned.
- `argc`: The number of arguments.
- `argv`: The list of string values from a command line.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT helics_federate_info helicsCreateFederateInfo(void)

Create a federate info object for specifying federate information when constructing a federate.

Return A helics_federate_info object which is a reference to the created object.

HELICS_EXPORT helics_federate helicsCreateMessageFederate(const char * fedName, helics_fede

Create a message federate from a federate info object.

helics_message_federate objects can be used in all functions that take a helics_message_federate or helics_federate object as an argument.

Return An opaque message federate object.

Parameters

- `fedName`: The name of the federate to create.
- `fi`: The federate info object that contains details on the federate.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT helics_federate helicsCreateMessageFederateFromConfig(const char * configFile

Create a message federate from a JSON file or JSON string or TOML file.

helics_message_federate objects can be used in all functions that take a helics_message_federate or helics_federate object as an argument.

Return An opaque message federate object.

Parameters

- `configFile`: A Config(JSON,TOML) file or a JSON string that contains setup and configuration information.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT helics_query helicsCreateQuery(const char * target, const char * query)
Create a query object.

A query object consists of a target and query string.

Parameters

- `target`: The name of the target to query.
- `query`: The query to make of the target.

HELICS_EXPORT helics_federate helicsCreateValueFederate(const char * fedName, helics_federa
Create a value federate from a federate info object.

helics_federate objects can be used in all functions that take a helics_federate or helics_federate object as an argument.

Return An opaque value federate object.

Parameters

- `fedName`: The name of the federate to create, can NULL or an empty string to use the default name from fi or an assigned name.
- `fi`: The federate info object that contains details on the federate.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT helics_federate helicsCreateValueFederateFromConfig(const char * configFile,
Create a value federate from a JSON file, JSON string, or TOML file.

helics_federate objects can be used in all functions that take a helics_federate or helics_federate object as an argument.

Return An opaque value federate object.

Parameters

- `configFile`: A JSON file or a JSON string or TOML file that contains setup and configuration information.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT const char* helicsGetBuildFlags(void)
Get the build flags used to compile HELICS.

HELICS_EXPORT const char* helicsGetCompilerVersion(void)
Get the compiler version used to compile HELICS.

HELICS_EXPORT helics_federate helicsGetFederateByName(const char * fedName, helics_error
Get an existing federate object from a core by name.

The federate must have been created by one of the other functions and at least one of the objects referencing the created federate must still be active in the process.

Return NULL if no fed is available by that name otherwise a helics_federate with that name.

Parameters

- `fedName`: The name of the federate to retrieve.
- `err`: An error object that will contain an error code and string if any error occurred during the execution of the function.

HELICS_EXPORT int helicsGetOptionIndex(const char * val)
Get an option index for use in /ref helicsPublicationSetOption, /ref helicsInputSetOption, /ref helicsEndpointSetOption, /ref helicsFilterSetOption, and the corresponding get functions.

Return An int with the option index or (-1) if not a valid property.

Parameters

- `val`: A string with the option name.

HELICS_EXPORT int helicsGetPropertyIndex(const char * val)
Get a property index for use in /ref helicsFederateInfoSetFlagOption, /ref helicsFederateInfoSetTimeProperty, or /ref helicsFederateInfoSetIntegerProperty

Return An int with the property code or (-1) if not a valid property.

Parameters

- `val`: A string with the property name.

HELICS_EXPORT const char* helicsGetVersion(void)
Get a version string for HELICS.

HELICS_EXPORT helics_bool helicsIsCoreTypeAvailable(const char * type)
Returns true if core/broker type specified is available in current compilation.

Options include “zmq”, “udp”, “ipc”, “interprocess”, “tcp”, “default”, “mpi”.

Parameters

- `type`: A string representing a core type.

HELICS_EXPORT const char* helicsSubscriptionGetKey(helics_input ipt)
Get the key of a subscription.

Return A const char with the subscription key.

HELICS_EXPORT helics_error helicsErrorInitialize(void)
Return an initialized error object.

HELICS_EXPORT void helicsErrorClear(helics_error * err)
Clear an error object.

clear an error object

CHAPTER 8

C++ API Reference (Doxygen)

CHAPTER 9

Developer Guide

9.1 Style Guide

The goal of the style guide is to describe in detail naming conventions for developing HELICS. Style conventions are encapsulated in the .clang_format files in the project.

We have an EditorConfig file that has basic formatting rules code editors and IDEs can use. See https://editorconfig.org/ for how to setup support in your preferred editor or IDE.

9.1.1 Naming Conventions

1. All functions should be `camelCase`

```
publication_id_t registerGlobalPublication (const std::string &name, const␣
↪std::string &type, const std::string &units = "");
```

2. All classes should be `PascalCase` except those noted below(namely classes which are wrappers for fundamental types)

```
class ValueFederate : public virtual Federate
{
public:
    ValueFederate (const FederateInfo &fi);
}
```

3. class methods should be `camelCase`

```
Publication &registerGlobalPublication (const std::string &name, const␣
↪std::string &type, const std::string &units = "");
```

   Exceptions: functions that match standard library functions e.g. to_string()

4. All fundamental types and enumerations should be underscore separated words in lower case. Fundamental types are those for which normal usage does not involve calling any methods or are simple aliases for other fundamental types

```
/* Type definitions */
typedef enum {
    helics_ok,
    helics_discard,
    helics_warning,
    helics_error,
} helics_status;

typedef void *helics_subscription;
typedef void *helics_publication;
typedef void *helics_endpoint;
typedef void *helics_source_filter;
typedef void *helics_destination_filter;
typedef void *helics_core;
typedef void *helics_broker;

typedef int helics_bool_t;
```

5. All C++ functions and types should be contained in the helics namespace with subnamespaces used as appropriate

```
namespace helics
{
    ...
} // namespace helics
```

6. C interface functions should begin with helicsXXXX

```
helics_bool helicsBrokerIsConnected (helics_broker broker);
```

7. C interface function should be of the format helics{Class}{Action} or helics{Action} if no class is appropriate

```
helics_bool helicsBrokerIsConnected (helics_broker broker);

const char *helicsGetVersion ();
```

8. All cmake commands (those defined in cmake itself) should be lower case

```
if as opposed to IF
install vs INSTALL
```

9. Public interface functions should be documented consistent with Doxygen style comments non public ones should be documented as well with doxygen but we are a ways from that goal

```
/** get an identifier for the core
    @param core the core to query
    @return a string with the identifier of the core
*/
HELICS_EXPORT const char *helicsCoreGetIdentifier (helics_core core);
```

9.2 Generating SWIG extension

Python

The easiest way to generate the latest C files for the Python extension is to use CMake itself. For example, you can run the following in a POSIX/Unix environment where you have `swig` installed with Python 3.6.

```
git clone https://github.com/GMLC-TDC/HELICS
cd HELICS
mkdir build
cmake -DBUILD_PYTHON_INTERFACE=ON -DPYTHON_INCLUDE_DIR=$(python3-config --prefix)/
↪include/python3.6m/ -DCMAKE_INSTALL_PREFIX=$HOME/local/helics-develop/ .. && make -
↪j 8 && make install
cd swig/python
cp helicsPYTHON_wrap.c ../../../swig/python/helics_wrap.c
cp helics.py ../../../swig/python/helics.py
```

This method verifies that the C file generated from CMake using SWIG compiles correctly into a Python extension.

For building a Python 2 compatible interface, use BUILD_PYTHON2_INTERFACE instead of BUILD_PYTHON_INTERFACE.

MATLAB

For the MATLAB extension, you need a special version of SWIG. Get it here.

```
git clone https://github.com/jaeandersson/swig
cd swig
./configure --prefix=/Users/$(whoami)/local/swig-matlab/ && make -j8 && make install
```

The matlab interface can be built using BUILD_MATLAB_INTERFACE in the cmake build of HELICS. This will use a MATLAB installation to build the interface. See installation

Octave

Octave is a free program that works similarly to MATLAB Building the octave interface requires swig, and currently will work with Octave 4.0 through 4.2. 4.4 is not currently supported by SWIG unless you build from the current master of the swig repo and use that version. The next release of swig will likely support it. It does work on windows, though the actual generation is not fully operational for unknown reasons and will be investigated at some point. A mkhelicsOCTFile.m is generated in the build directory this needs to be executed in octave, then a helics.oct file should be generated, the libHelicsShared.dll needs to be copied along with the libzmq.dll files Once this is done the library can be loaded by calling helics. On linux this build step is done for you with BUILD_OCTAVE_INTERFACE.

C# A C# interface can be generated using swig and enabling BUILD_CSHARP_INTERFACE in the CMake. The support is partial; it builds and can be run but not all the functions are completely usable and it hasn't been fully tested.

Java A JAVA interface can be generated using swig and enabling BUILD_JAVA_INTERFACE in the CMake. This interface is tested regularly as part of the CI test system.

9.3 Run tests

Python

To run the python tests, first install pytest

```
pip install pytest
```

Then run it by doing the following

```
cd ~/GitRepos/HELICS/tests/
pytest -sv python_helics
```

9.4 Generating Documentation

The documentation requires Pandoc to convert from Markdown to RST.

You will need the following Python packages.

```
pip install sphinx
pip install ghp-import
pip install breathe
pip install sphinx_rtd_theme
pip install nbsphinx
pip install sphinxcontrib-pandoc-markdown
```

You will also need doxygen.

You can then type `make doxygen html` to create the documentation locally.

If you don't have Pandoc, you can install it using `conda`.

```
conda install pandoc
```

If you are unable to install `pandoc`, you may be able to generate some of the documentation if you install the following.

```
pip install recommonmark
```

9.5 HELICS Benchmarks

The HELICS repository has a few benchmarks that are intended to test various aspects of the code and record performance over time

9.5.1 Baseline benchmarks

These benchmarks run on a single machine using Google Benchmarks and are intended to test various aspects of HELICS over a range of spaces applicable to a single machine.

ActionMessage

Micro-benchmarks to test some operations concerning the serialization of the underlying message structure in HELICS

Conversion

Micro-benchmarks to test the serialization and deserialization of common data types in HELICS

9.5.2 Simulation Benchmarks

Echo

A set of federates representing a hub and spoke model of communication for value based interfaces

Echo_c

A set of federates representing a hub and spoke model of communication for value based interfaces using the C shared library.

Echo Message

A set of federates representing a hub and spoke model of communication for message based interfaces

Filter

A variant of the Echo message test that add filters to the messages

Ring Benchmark

A ring like structure that passes a value token around a bunch of times

Ring Message Benchmark

A ring like structure that passes a message token around a bunch of times.

Timing Benchmark

Similar to echo but doesn't actually send any data just pure test of the timing messages

9.5.3 Message Benchmarks

Benchmarks testing various aspects of the messaging structure in HELICS

MessageLookup

Benchmarks sends messages to random federates, varying the total number of interfaces and federates.

MessageSend

Sending messages between 2 federates varying the message size and count per timing loop.

9.5.4 Standardized Tests

PHold

A standard PHOLD benchmark varying the number of federates.

9.5.5 Multinode Benchmarks

Some of the benchmarks above have multinode variants. These benchmarks will have a standalone binary for the federate used in the benchmark that can be run on each node. Any multinode benchmark run will require some setup to make it launch in your particular environment and knowing the basics for the job scheduler on your cluster will be very helpful.

Any sbatch files for multinode benchmark runs in the repository are setup for running in the pdebug queue on LC's Quartz cluster. They are unlikely to work as is on other clusters, however they should work as a starting point for other clusters using slurm. The minimum changes required are likely to involve setting the queue/partition correctly and ensuring the right bank/account for charging CPU time is used.

9.6 Description of the different continuous integration test setups running on the CI servers

There are 5 CI servers that are running along with a couple additional checks Travis, Appveyor, Circle-CI, Azure and Drone.

9.6.1 Travis-CI Tests

Travis-ci runs many of the primary checks In 3 different stages

Push Tests

Push tests run on all pushes to any branch in the main repository, there are 4 tests that run regularly

- GCC 6: Test the GCC 6.0 compiler and the CI labeled Tests BOOST 1.61, SWIG, MPI
- Clang 5: Test the clang compiler and run the CI labeled Tests, along with python and Java interface generation and Tests Using C++17
- GCC 4.9: Test the oldest supported compiler in GCC, Test the included interface files(SWIG OFF) for Java and python, and test a packaging build. The main tests are disabled, BOOST 1.61
- XCode 10.2: Test a recent XCode compiler with the Shared API library tests

PR tests and develop branch Tests

Pull request tests run on every pull request to develop or master. In addition to the previous 4 tests 2 additional tests are run.

- Clang 3.6: which is the oldest fully supported clang compiler, with boost 1.58 (Build only)
- XCode 10.2

Daily Builds on develop

On the develop branch a few additional tests are run on a daily basis. These will run an extended set of tests or things like valgrind or clang-sanitizers. The previous tests are run with an extended set of tests and a few additional tests are run

- gcc 6.0 valgrind, interface disabled
- gcc 6.0 Code Coverage, MPI, interfaces disabled
- gcc 6.0 ZMQ subproject cmake 3.11
- Mingw test building on the Mingw platform
- Xcode 9.4 which is the oldest fully supported Xcode version (not for PRs to develop)

9.6.2 Appveyor tests

- MSVC 2015 CMake 3.13, python and JAVA builds

9.6.3 Azure tests

PRs and commits to the master and develop branches that pass the tests on Travis will trigger builds on Azure for several other HELICS related repositories (such as HELICS-Examples). The result of the builds for those repositories will be reported as a comment on the PR (if any) that triggered the build.

On the Primary HELICS repository there are 4 Azure builds:

- MSVC2015 64bit Build and test, chocolatey swig/boost
- MSVC2017 32bit Build and test
- MSVC2017 64bit Build and test with Java
- MSVC2019 64bit Build and test with Java

9.6.4 Circle CI

All PR's and branches trigger a set of builds using Docker images on Circle-CI.

- Octave tests - tests the Octave interface and runs some tests
- Clang-MSAN - runs the clang memory sanitizer
- Clang-ASAN - runs the clang address sanitizer and undefined behavior sanitizer
- Clang-TSAN - runs the clang thread sanitizer
- install1 - build and install and link with the C shared library, C++ shared library, C++98 library and C++ apps library, and run some tests linking to the installed libraries
- install2 - build and install and link with the C shared library, and C++98 library only and run some tests linking with the installed library

Benchmark tests

Circle ci also runs a benchmark test that runs every couple days. Eventually this will form the basis of benchmark regression test.

9.6.5 Drone

- 64 bit and 32 bit builds on ARM processors

9.6.6 Cirrus CI

- FreeBSD 12.1 build

9.6.7 Read the docs

- Build the docs for the website and test on every commit

9.6.8 Codacy

There are some static analysis checks run with Codacy. While it is watched it is not always required to pass.

9.7 Public API

This file defines what is included in what is considered the stable user API for HELICS.

This API will be backwards code compatible through major version numbers, though functions may be marked deprecated between minor version numbers. Functions in any other header will not be considered in versioning decisions. If other headers become commonly used we will take that into consideration at a later time. Anything marked private is subject to change and most things marked protected can change as well though somewhat more consideration will be given in versioning.

The public API includes the following

- Application API headers
  - CombinationFederate.hpp
  - Publications.hpp
  - Subscriptions.hpp
  - Endpoints.hpp
  - Filters.hpp
  - Federate.hpp
  - helicsTypes.hpp
  - data_view.hpp
  - MessageFederate.hpp
  - MessageOperators.hpp
  - ValueConverter.hpp
  - ValueFederate.hpp
  - HelicsPrimaryTypes.hpp
  - queryFunctions.hpp
  - FederateInfo.hpp

  - Inputs.hpp
  - BrokerApp.hpp (New in 2.3 Moved from App library)
  - CoreApp.hpp (New in 2.3)- Operations and some capabilities may be added or tweaked in the next revision
  - timeOperations.hpp (New in 2.3)- previously functions were in helics-time.hpp
  - typeOperations.hpp (New in 2.3)- previously functions were in core-types.hpp
  - Exceptions: Any function or method dealing with Inputs with data from multiple sources is subject to change, the vector subscriptions, and vector inputs are subject to change. The functionality related to PublishJSON is considered experimental and may change in the future. The queries to retrieve JSON may update the format of the returned JSON in the future. A general note on queries. The data returned via queries is subject to change, in general queries will not be removed, but if a need arises the data structure may change at minor revision numbers.

- Core library headers
  - Core.hpp
  - Broker.hpp
  - CoreFactory.hpp (Header is deprecated for public API in 2.3 use CoreApp instead)
  - BrokerFactory.hpp (Header is deprecated for public API in 2.3 use BrokerApp instead)
  - core-exceptions.hpp
  - core-types.hpp (string operation functions moved to typeOperations.hpp in 2.3, though are still available for compatibility reasons in the Public API)
  - core-data.hpp
  - helics-time.hpp (string operation functions moved to timeOperations.hpp in 2.3, though are still available for compatibility reasons in the Public API)
  - CoreFederateInfo.hpp
  - helicsVersion.hpp
  - federate_id.hpp
  - helics_definitions.hpp
  - NOTE: core headers in the public API are headers that need to be available for the Application API public headers. The core api can be used more directly with static linking but applications are generally recommended to use the application API or other higher level API's

- C Shared Library headers (c)
  - api-data.h
  - helics.h
  - helicsCallbacks.h (Not well used and considered experimental yet)
  - MessageFederate.h
  - MessageFilters.h
  - ValueFederate.h

- App Library
  - Player.hpp
  - Recorder.hpp

  - Echo.hpp
  - Source.hpp
  - Tracer.hpp
  - Clone.hpp (new in 2.2)
  - helicsApp.hpp
  - BrokerApp.hpp (aliased to application_api version)
  - CoreApp.hpp (aliased to application_api version)
  - BrokerServer.hpp (removed in 2.3 as not useful for library operations, though still available in the static library)

- Exceptions: Any function dealing with Inputs concerning data from multiple sources is subject to change, the vector subscription Objects, and vector Input objects are subject to change. Also some changes may occur in regard to units on the Application API.
- C++98 Library *All headers are mostly stable. Though we reserve the ability to make changes to make them better match the main C++ API.*

In the installed folder are some additional headers from third party libraries (cereal, C++ compatibility headers, CLI11, utilities), we will try to make sure these are compatible in the features used in the HELICS API, though changes in other aspects of those libraries will not be considered in HELICS versioning, this caveat includes anything in the `helics/external` and `helics/utilities` directories. Only changes which impact the signatures defined above will factor into versioning decisions. You are free to use them but they are not guaranteed to be backwards compatible on version changes.

CHAPTER 10

RoadMap

This document contains tentative plans for changes and improvements of note in upcoming versions of the HELICS library. All dates are approximate and subject to change, but this is a snapshot of the current planning thoughts. See the projects for additional details

10.1 [2.7] ~ 2020-11-15

This will be the last of the 2.X series releases, there will likely be at least one patch release after this before fully moving to 3.0

- Internal text based (probably JSON) message format option for general backwards compatibility
- Function deprecations to prepare people to move to 3.0

10.2 [3.0] ~ 2020-11-6 Beta, Final release Mid November

Upgrade minimum compilers and build systems. Currently planned minimum targets are gcc 7.0, clang 5.0, MSVC 2017 15.7, XCode 10.1, and CMake 3.10. This is a setup which should be supported on Ubuntu 18.04 repositories. Minimum Boost version will also be updated though Boost is becoming less critical for the HELICS core so may not be that important. The likely minimum tested target will likely be 1.65.1 though the core might work with older versions and its use can be disabled completely. Certain features may require a newer boost version(1.70) than what would be available on Ubuntu 18.04. General target requirements will allow HELICS to build on the most recent 2 LTS versions of Ubuntu using readily available repo packages. Minimum required compilers for building on macOS and systems using ICC will include Xcode 10 and ICC 19. The minimum ZMQ version will also be bumped up to 4.2. General policy for Mac builds will be supporting Xcode compilers on versions of MacOS that receive security upgrades which is generally the last 3 versions, though 10.1+ and 11 will likely be the only 2 supported at HELICS 3.0 release, and support minor releases for at least 2 years. MSVC compilers will be supported for at least 2 years from release date, an appropriate CMake (meaning slightly newer than the compiler) will also be required for Visual Studio builds.

- Control interface

- Targeted endpoints
- General API changes based on feedback and code review
- Remove deprecated functions
- Change values for log level enumerations
- Some additional renaming of CMake variables
- Renaming of some of the libraries and reorganization of the header locations

10.3 [3.1] ~ 2020-11-15

Mostly things that didn't quite make it into the 3.0 release and a number of bug fixes that come from transitioning to HELICS 3.0.

- SSL capable core (unlikely in 3.1 but someday)
- Full Dynamic Federation support
- Single thread cores (partial at release)
- Plugin architecture for user defined cores
- xSDK compatibility
- Much more general debugging support

You can find Doxygen documentation here.

Index

H