Cicada Documentation

Release v1.4.1

Project 8 Collaboration

Apr 24, 2023

Contents

Contents:

CHAPTER 1

Introduction

Cicada contains the libraries describing the objects generated by the Katydid processing. These objects should be contained in ROOT files: as such they inherit from the TObject class (the "mother of all ROOT objects"). Their nature should be very close to TTree objects.

CHAPTER 2

Getting started

2.1 Installation

The following steps will build Cicada from scratch. Starting with a terminal window ...

1. Clone the repository and make a build directory as recommended above. You will also have to initialize the submodules.

```
git clone "https://github.com/project8/cicada"
cd cicada
git submodule update --init --recursive
mkdir build
```

2. To configure the installation you can use cmake, ccmake, or cmake-gui. For a first configuration, using either ccmake or cmake-gui is highly recommended. The following instructions are for ccmake, but the steps with cmake-gui would be approximately the same.

```
cd build
ccmake ..
```

You will be prompted to press [c] to configure, and the window will fill up with several options. You should set the CMake variable *CMAKE_BUILD_TYPE* to either *RELEASE*, *STANDARD*, or *DEBUG* (default), in order of how much text output you would like (from least to most) and how much compiler optimization should be performed (from most to least). The install prefix is specified by the CMake variable *CMAKE_INSTALL_PREFIX*. The library, binaries, and header files will be installed in the lib, bin, and include subdirectories. The default install prefix is the build directory. After you've finished, if you've changed anything press [c] again to configure. Then [g] to generate and exit.

3. Build and install.

```
make install
```

Or if you want to take advantage of parallel building to get things done faster:

```
make -j install
```

If the compiler runs into errors during the build, first check that you've updated the submodules and that you have all of the required dependencies installed (many are called "optional" on this page, but if you want to build without them you must also specify this in the cmake window). If you made a change to the dependencies or submodules, you may have to wipe the build directory and start again from step 1; simply writing *make install* again will not always work.

4. Use *this_cicada.sh* to set your *PYTHONPATH* and *ROOT_INCLUDE_PATH*:

```
source /path/to/cicada/install/bin/this_cicada.sh
```

This will allow you to access the Python and ROOT interfaces and the python example scripts coming with cicada.

5. **Developpers:** Add the installation folder (where the bin and lib have been installed) path to your paths.

```
export PATH=/path/to/cicada/install/bin:$PATH
export LD_LIBRARY_PATH=/path/to/cicada/install/lib:$LD_LIBRARY_PATH
```

2.2 How to use

Cicada does not do anything per se, but is a library retaining the structure of some objects generated by Katydid (see *Objects' structure and description* section for more details). An example on how to read a Katydid object is present in *Library/python*. The core of this scripts is:

```
import CicadaPy
CicadaPy.loadLibraries()

import ROOT.Katydid as KT
from ROOT import TFile, TTreeReader, TTreeReaderValue

def ReadKTOutputFile(filename,var):
    # Change to point to the ROOT file you want
    # filename = "../../scripts/PhaseI_analysis_roofit/events_000001097_katydid_v2.7.
↪0_concat.root"
    file = TFile.Open(filename)
    if not file:
        raise FileNotFoundError("File {} does not exist".format(filename))

    # Extract tree from file
    tree = file.Get("multiTrackEvents")
    # Create TTreeReader
    treeReader = TTreeReader(tree)
    # Create object TMultiTrackEventData to "point" to the object "Event" in the tree
    multiTrackEventObject = TTreeReaderValue(KT.TMultiTrackEventData)(treeReader,
↪"Event")

    resultList = []
    # Go through the events
    while treeReader.Next():
        exec("resultList.append(multiTrackEvents.Get{}())\n".format(var))
        print(resultList[-1])
    return resultList
```

1. The first two lines add the Cicada libraries into the ROOT module, so they can be imported by the second import. Note that here we call *ROOT.Katydid* and not *ROOT.Cicada*: we are using the *Katydid* namespace that was added to ROOT. One day once the Cicada library will be used as a dependency of Katydid, we will use *ROOT.Cicada*.

2. The function *ReadKTOutputFile* takes a file name and a variable of interest and it will print the value of this variable. To do this, it extracts the tree called *multiTrackEvents* containing the TMultiTrackEventData object and makes use of

the ROOT TTreeReader to get each value of this object. The iterator of the tree reader is then used to append to a list and print the value of the parameters *var* of the event object using a *GetX* method (defined by the class). The list is then returned.

CHAPTER 3

Objects' structure and description

Here is described the content and associated description of the Cicada objects.

3.1 Version 1.2.0

3.1.1 TMultiTrackEventData

- **Component** – UInt_t:
- **AcquisitionID** – UInt_t:
- **EventID** – UInt_t: number of the event in the katydid instance
- **TotalEventSequences** – UInt_t: number of sequences in the event
- **StartTimeInRunC** – Double_t: start time of the event in the Katydid instance
- **StartTimeInAcq** – Double_t: start time of the event within the chunk of data
- **EndTimeInRunC** – Double_t: end time of the event in the Katydid instance
- **TimeLength** – Double_t: duration of the event
- **StartFrequency** – Double_t: start frequency of the event
- **EndFrequency** – Double_t: end frequency of the event
- **MinimumFrequency** – Double_t: minimum frequency reached by the event
- **MaximumFrequency** – Double_t: maximum frequency reached by the event
- **FrequencyWidth** – Double_t: range of frequencies covered by the event tracks
- **StartTimeInRunCSigma** – Double_t: error on the start time of the event in the Katydid instance
- **EndTimeInRunCSigma** – Double_t: error on the end time of the event in the Katydid instance
- **StartFrequencySigma** – Double_t: error on the start frequency of the event

- **EndFrequencySigma** – Double_t: error on the end frequency of the event
- **FrequencyWidthSigma** – Double_t: error on the range of frequencies covered by the event tracks
- **FirstTrackID** – UInt_t: ID number of the first track in the event
- **FirstTrackTimeLength** – Double_t: length of the first track
- **FirstTrackFrequencyWidth** – Double_t: range of frequencies covered by the first track
- **FirstTrackSlope** – Double_t: slope of the first track
- **FirstTrackIntercept** – Double_t: intercept at t=0 of the first track
- **FirstTrackTotalPower** – Double_t: sum of the power of the bins composing the first track
- **FirstTrackNTrackBins** – UInt_t: number of bins in the first track
- **FirstTrackTotalTrackSNR** – Double_t: Sum of the first track bins SNR
- **FirstTrackMaxTrackSNR** – Double_t: Max of the first track bins SNR
- **FirstTrackTotalTrackNUP** – Double_t: Sum of the first track bins Normalized Unitless Power (NUP)
- **FirstTrackMaxTrackNUP** – Double_t: Max of the first track bins Normalized Unitless Power (NUP)
- **FirstTrackTotalWideTrackSNR** – Double_t: Sum of the first track extended bins SNR
- **FirstTrackTotalWideTrackNUP** – Double_t: Sum of the first track extended bins Normalized Unitless Power (NUP)
- **UnknownEventTopology** – Double_t: boolean describing if a track has some weird topology

3.1.2 TProcessedTrackData

- **Component** – UInt_t:
- **AcquisitionID** – UInt_t:
- **TrackID** – UInt_t: number of the track in the event
- **EventID** – UInt_t: number of the event in the katydid instance
- **EventSequenceID** – UInt_t: number of the sequence in the event reconstruction
- **IsCut** – Bool_t: should this line be cut when calculating the start frequency. . . ?
- **StartTimeInRunC** – Double_t: start time of the track in the Katydid instance
- **StartTimeInAcq** – Double_t: start time of the track within the chunk of data
- **EndTimeInRunC** – Double_t: end time of the track in the Katydid instance
- **TimeLength** – Double_t: track length
- **StartFrequency** – Double_t: start frequency of the track
- **EndFrequency** – Double_t: end frequency of the track
- **FrequencyWidth** – Double_t: range of frequencies covered by the track
- **Slope** – Double_t: slope of the track
- **Intercept** – Double_t: intercept at t=0 of the track
- **TotalPower** – Double_t: sum of the power of the bins composing the track
- **NTrackBins** – UInt_t: number of bins in the track

- **TotalTrackSNR** – Double_t: Sum of the track bins SNR
- **MaxTrackSNR** – Double_t: Max of the track bins SNR
- **TotalTrackNUP** – Double_t: Sum of the track bins Normalized Unitless Power (NUP)
- **MaxTrackNUP** – Double_t: Max of the track bins Normalized Unitless Power (NUP)
- **TotalWideTrackSNR** – Double_t: Sum of the track extended bins SNR
- **TotalWideTrackNUP** – Double_t: Sum of the track extended bins Normalized Unitless Power (NUP)
- **StartTimeInRunCSigma** – Double_t: error on the start time of the track in the Katydid instance
- **EndTimeInRunCSigma** – Double_t: error on the end time of the track in the Katydid instance
- **TimeLengthSigma** – Double_t: error on the track length
- **StartFrequencySigma** – Double_t: error on the track start frequency
- **EndFrequencySigma** – Double_t: error on the track end frequency
- **FrequencyWidthSigma** – Double_t: error on the range of frequencies covered by the track
- **SlopeSigma** – Double_t: error on the slope track
- **InterceptSigma** – Double_t: error on the track intercept
- **TotalPowerSigma** – Double_t: error on the sum of the power of the bins composing the track

3.2 Version 1.1.0

3.2.1 TMultiTrackEventData

- **Component** – UInt_t:
- **AcquisitionID** – UInt_t:
- **EventID** – UInt_t: number of the event in the katydid instance
- **TotalEventSequences** – UInt_t: number of sequences in the event
- **StartTimeInRunC** – Double_t: start time of the event in the Katydid instance
- **StartTimeInAcq** – Double_t: start time of the event within the chunk of data
- **EndTimeInRunC** – Double_t: end time of the event in the Katydid instance
- **TimeLength** – Double_t: duration of the event
- **StartFrequency** – Double_t: start frequency of the event
- **EndFrequency** – Double_t: end frequency of the event
- **MinimumFrequency** – Double_t: minimum frequency reached by the event
- **MaximumFrequency** – Double_t: maximum frequency reached by the event
- **FrequencyWidth** – Double_t: range of frequencies covered by the event tracks
- **NTrackBins** – UInt_t: number of bins in the track
- **TotalTrackSNR** – Double_t: Sum of the track bins SNR
- **MaxTrackSNR** – Double_t: Max of the track bins SNR
- **TotalTrackNUP** – Double_t: Sum of the track bins Normalized Unitless Power (NUP)

- **MaxTrackNUP** – Double_t: Max of the track bins Normalized Unitless Power (NUP)
- **TotalWideTrackSNR** – Double_t: Sum of the track extended bins SNR
- **TotalWideTrackNUP** – Double_t: Sum of the track extended bins Normalized Unitless Power (NUP)
- **StartTimeInRunCSigma** – Double_t: error on the start time of the event in the Katydid instance
- **EndTimeInRunCSigma** – Double_t: error on the end time of the event in the Katydid instance
- **StartFrequencySigma** – Double_t: error on the start frequency of the event
- **EndFrequencySigma** – Double_t: error on the end frequency of the event
- **FrequencyWidthSigma** – Double_t: error on the range of frequencies covered by the event tracks
- **FirstTrackID** – UInt_t: ID number of the first track in the event
- **FirstTrackTimeLength** – Double_t: length of the first track
- **FirstTrackFrequencyWidth** – Double_t: range of frequencies covered by the first track
- **FirstTrackSlope** – Double_t: slope of the first track
- **FirstTrackIntercept** – Double_t: intercept at t=0 of the first track
- **FirstTrackTotalPower** – Double_t: sum of the power of the bins composing the first track
- **UnknownEventTopology** – Double_t: boolean describing if a track has some weird topology

3.2.2 TProcessedTrackData

- **Component** – UInt_t:
- **AcquisitionID** – UInt_t:
- **TrackID** – UInt_t: number of the track in the event
- **EventID** – UInt_t: number of the event in the katydid instance
- **EventSequenceID** – UInt_t: number of the sequence in the event reconstruction
- **IsCut** – Bool_t: should this line be cut when calculating the start frequency. . . ?
- **StartTimeInRunC** – Double_t: start time of the track in the Katydid instance
- **StartTimeInAcq** – Double_t: start time of the track within the chunk of data
- **EndTimeInRunC** – Double_t: end time of the track in the Katydid instance
- **TimeLength** – Double_t: track length
- **StartFrequency** – Double_t: start frequency of the track
- **EndFrequency** – Double_t: end frequency of the track
- **FrequencyWidth** – Double_t: range of frequencies covered by the track
- **Slope** – Double_t: slope of the track
- **Intercept** – Double_t: intercept at t=0 of the track
- **TotalPower** – Double_t: sum of the power of the bins composing the track
- **StartTimeInRunCSigma** – Double_t: error on the start time of the track in the Katydid instance
- **EndTimeInRunCSigma** – Double_t: error on the end time of the track in the Katydid instance
- **TimeLengthSigma** – Double_t: error on the track length

- **StartFrequencySigma** – Double_t: error on the track start frequency
- **EndFrequencySigma** – Double_t: error on the track end frequency
- **FrequencyWidthSigma** – Double_t: error on the range of frequencies covered by the track
- **SlopeSigma** – Double_t: error on the slope track
- **InterceptSigma** – Double_t: error on the track intercept
- **TotalPowerSigma** – Double_t: error on the sum of the power of the bins composing the track

3.3 Version 0.4.2

3.3.1 TMultiTrackEventData

- **Component** – UInt_t:
- **AcquisitionID** – UInt_t:
- **EventID** – UInt_t: number of the event in the katydid instance
- **TotalEventSequences** – UInt_t: number of sequences in the event
- **StartTimeInRunC** – Double_t: start time of the event in the Katydid instance
- **StartTimeInAcq** – Double_t: start time of the event within the chunk of data
- **EndTimeInRunC** – Double_t: end time of the event in the Katydid instance
- **TimeLength** – Double_t: duration of the event
- **StartFrequency** – Double_t: start frequency of the event
- **EndFrequency** – Double_t: end frequency of the event
- **MinimumFrequency** – Double_t: minimum frequency reached by the event
- **MaximumFrequency** – Double_t: maximum frequency reached by the event
- **FrequencyWidth** – Double_t: range of frequencies covered by the event tracks
- **StartTimeInRunCSigma** – Double_t: error on the start time of the event in the Katydid instance
- **EndTimeInRunCSigma** – Double_t: error on the end time of the event in the Katydid instance
- **StartFrequencySigma** – Double_t: error on the start frequency of the event
- **EndFrequencySigma** – Double_t: error on the end frequency of the event
- **FrequencyWidthSigma** – Double_t: error on the range of frequencies covered by the event tracks
- **FirstTrackID** – UInt_t: ID number of the first track in the event
- **FirstTrackTimeLength** – Double_t: length of the first track
- **FirstTrackFrequencyWidth** – Double_t: range of frequencies covered by the first track
- **FirstTrackSlope** – Double_t: slope of the first track
- **FirstTrackIntercept** – Double_t: intercept at t=0 of the first track
- **FirstTrackTotalPower** – Double_t: sum of the power of the bins composing the first track
- **UnknownEventTopology** – Double_t: boolean describing if a track has some weird topology

3.3.2 TProcessedTrackData

- **Component** – UInt_t:
- **AcquisitionID** – UInt_t:
- **TrackID** – UInt_t: number of the track in the event
- **EventID** – UInt_t: number of the event in the katydid instance
- **EventSequenceID** – UInt_t: number of the sequence in the event reconstruction
- **IsCut** – Bool_t: should this line be cut when calculating the start frequency. . . ?
- **StartTimeInRunC** – Double_t: start time of the track in the Katydid instance
- **StartTimeInAcq** – Double_t: start time of the track within the chunk of data
- **EndTimeInRunC** – Double_t: end time of the track in the Katydid instance
- **TimeLength** – Double_t: track length
- **StartFrequency** – Double_t: start frequency of the track
- **EndFrequency** – Double_t: end frequency of the track
- **FrequencyWidth** – Double_t: range of frequencies covered by the track
- **Slope** – Double_t: slope of the track
- **Intercept** – Double_t: intercept at t=0 of the track
- **TotalPower** – Double_t: sum of the power of the bins composing the track
- **StartTimeInRunCSigma** – Double_t: error on the start time of the track in the Katydid instance
- **EndTimeInRunCSigma** – Double_t: error on the end time of the track in the Katydid instance
- **TimeLengthSigma** – Double_t: error on the track length
- **StartFrequencySigma** – Double_t: error on the track start frequency
- **EndFrequencySigma** – Double_t: error on the track end frequency
- **FrequencyWidthSigma** – Double_t: error on the range of frequencies covered by the track
- **SlopeSigma** – Double_t: error on the slope track
- **InterceptSigma** – Double_t: error on the track intercept
- **TotalPowerSigma** – Double_t: error on the sum of the power of the bins composing the track

CHAPTER 4

Versions

As Katydid expands, the content of the Cicada might change: the table below summarizes the compatibility between Cicada and Katydid versions.

Cicada version	Katydid files version
0.1.0+	2.7.0+
0.4.2+	2.10.0+
1.0.0+	2.10.1+
1.1.0+	2.1X.Y+
1.4.0+	?

CHAPTER 5

Contributions

5.1 Reporting bugs

You can report bugs using the Cicada issue tracker. When doing so, please provide your configuration (gcc, cmake and ROOT versions, virtual environment) and a detailed description of the steps to reproduce the bug.

5.2 Development scheme, code testing et release procedure

5.2.1 Development scheme

The Project 8 collaboration has adopted the Git flow development scheme: you can find details and the correct git commands about how to use it on this page. For a more visual way of developing code, the use of the SourceTree application is recommended as it natively integrates the Git flow scheme.

5.2.2 Code testing with Docker

If you would like to modify your local installation of Cicada (to add features or resolve any bugs), we recommend you use a Docker container as a uniform test bench. To do so: * Install Docker: https://docs.docker.com/engine/installation/ * Clone and pull the latest master version of Cicada * Inside the cicada folder, execute

```
docker build -t cicada .
docker run -it cicada bash
```

A new terminal prompter (for example, *root@413ab10d7a8f:*) should appear. Then you have to load the environment:

```
source /setup.sh
```

You may make changes to Cicada either inside or outside of the Docker container. If you wish to work outside of the container, you will need to mount a local folder in the container (see Docker documentation).

You can remove the container image using

```
docker rmi Cicada_Cicada
```

5.2.3 Release procedure

When making a release or a hotfix, several steps shall be done:

- update the documentation (see Documentation section)
- update the project number in the top CMakeLists.txt file
- update the authors lists (if applicable)
- update the Documentation/ValidationLog.rst file

After making the release:

- create a release entry on Github and add the corresponding entry from the ValidationLog.rst file

5.3 Contributing to the code

If you are new to the code and are willing to contribute by developing new features or maintaining the code, please refer to the issue tracker. There are issues you can look at and decide on solving. When you have found your ideal issue, please comment in the issue tracker, so the main developers are aware you are working on this.

If you wish to contribute to maintaining a proper documentation, please refer to the Documentation section.

5.4 Documentation

The documentation of Cicada happens at several levels:

- in the repository, we maintain README.md files describing the content of each folder.
- in the Documentation folder, we maintain RST files. With each release of the code, ReadTheDocs reads these files and produces pages (such as this one).
- inside the code, documentation is provided as comments and Doxygen headers. Once compiled by ReadTheDocs, it produces a proper Doxygen documentation of the code.
- a ValidationLog.rst file keeps track of the new features or fixes added to the code. For each Github issue solved, a entry describing the solved issue (and its Github number) should be added in the upcoming release subsection.

This documentation must be updated at any release/hotfix/pull-request to keep the repository as up-to-date as possible.

CHAPTER 6

Validation Log

6.1 Guidelines

- All new features incorporated into a tagged release should have their validation documented. * Document the new feature. * Perform tests to validate the new feature. * If the feature is slated for incorporation into an official analysis, perform tests to show that the overall analysis works and benefits from this feature. * Indicate in this log where to find documentation of the new feature. * Indicate in this log what tests were performed, and where to find a writeup of the results.
- Fixes to existing features should also be validated. * Perform tests to show that the fix solves the problem that had been indicated. * Perform tests to show that the fix does not cause other problems. * Indicate in this log what tests were performed and how you know the problem was fixed.

6.2 Log

6.2.1 Version: v1.4.0

Release Date: Apr 24, 2023

Fixes:

- Updated Scarab to v3.9.4
- Fixed new CMake build

6.2.2 Version: v1.4.0

Release Date: July 21, 2021

New Features:

- Updated Scarab to v3.6.1
- Modernized CMake build
- Removed Katydid namespace classes

6.2.3 Version: v1.3.3

Release Date: March 28th 2019

Fixes:

- ReadKTOutput: Avoid crash when object doesn't exist in file

6.2.4 Version: 1.3.2

Release Date: Dec 6, 2018

Fixes:

- New docker dependencies

6.2.5 Version: 1.3.1

Release Date: Dec 5, 2018

Fixes:

- Removing unused libraries (yaml, json, param) from build
- Switched docker build to use COPY instead of git clone

6.2.6 Version: 1.3.0

Release Date: Nov 29, 2018

New Features:

- New Dockerfile based on the p8compute-dependencies container
- Added this_cicada.sh script to properly set ROOT include path and python path
- Fixed the ROOT dictionary build so that it doesn't hard-code source-tree paths

6.2.7 Version: 1.2.1

Release Date: June 27th, 2018

Fixes:

- Update Scarab

6.2.8 Version: 1.2.0

Release Date: June 15th, 2018

New Features:

- **TMultiTrackEventData: adding SNR and NUP based quantities for the first track:**
  - FirstTrackNTrackBins
  - FirstTrackTotalSNR
  - FirstTrackMaxSNR
  - FirstTrackTotalNUP
  - FirstTrackMaxNUP
  - FirstTrackTotalWideSNR
  - FirstTrackTotalWideNUP

6.2.9 Version: 1.1.1

Release Date: June 7th, 2018

Fixes:

- Bumping ClassDef version for TProcessedTrackData

6.2.10 Version: 1.1.0

Release Date: June 1st, 2018

New Features:

- **TProcessedTrackData: adding SNR and NUP based quantities:**
  - NTrackBins
  - TotalTrackSNR
  - MaxTrackSNR
  - TotalTrackNUP
  - MaxTrackNUP
  - TotalWideTrackSNR

  - TotalWideTrackNUP

- **Moving the default object name from the Katydid Writer into Cicada:**
  - TMultiTrackEventData
  - TProcessedTrackData
  - TProcessedMPTData (not done as inherits from TMultiTrackEventData)
  - TClassifierResultsData
- **ReadKTOutputFile:**
  - Extraction of tracks information from TMultiTrackEventData.
  - Extraction of multiple branches without one execution.

6.2.11 Version: 1.0.2

Release Date: April 12, 2018

New Features:

- ReadKTOutputFile: Support of Cicada and Katydid namespaces and access to TMultiTrackEventData members.

Fixes:

- Documentation update about the python libraries.

6.2.12 Version: 1.0.1

Release Date: April 10, 2018

Fixes:

- Docker: Sleep time after chmod of installation script.

6.2.13 Version: 1.0.0

Release Date: March 29, 2018

New Features:

- Classification related objects; added CMTEWithClassifierResultsData, CClassifierResultsData, and CProcessedMPTData.
- Definition of a Cicada-specific prefixes for Set, Get and variables; added CMemberVariables.hh.

Fixes:

- Docker: correction of the installation location (from /cicada/build to /build).
- Documentation/Doxygen updates.

6.2.14 Version: 0.4.2

Release Date: March 14, 2018

New Features:

Fixes:

- Fixed the namespace in the constructor for the Tracks TClonesArray in TMultiTrackEventData.

6.2.15 Version: 0.4.1

Release Date: February 22, 2018

New Features:

Fixes:

- Add const return of the Tracks TClonesArray in TMultiTrackEventData.

6.2.16 Version: 0.4.0

Release Date: February 14, 2018

New Features:

- Python interface via `` `import CicadaPy` `` after installation #2
- Dockerfile
- A proper documentation

Fixes:

6.2.17 Version: v0.3.0

Release Date: January 29, 2018

New Features:

- Classes TProcessedTrackData and TMultiTrackEventData defined across Katydid and Cicada namespaces

6.3 Template

6.3.1 Version:

Release Date:

New Features:

- **Feature 1**
  - Details
- **Feature 2**
  - Details

Fixes:

- **Fix 1**
  - Details
- **Fix 2**
  - Details

Full Doxygen API Reference