JETDEF DESIGN NOTES

PETER SHERWOOD

1. Introduction

1.1. **Purpose.** This note is a brief description of the intent and design of the software used to configure the jet slice software. It is intended to help those who need to modify the code.

1.2. **What JetDef does.** JetDef carries out the following tasks

- Limited checks the inputs for format and content.
- Rearranges the input information in a format that is more convenient for the rest of the program.
- Supplies information that is inferred - but cannot be deduced - from the incoming data.
- Determines sequences to be constructed.
- Determines which Algorithms make up the sequences.
- Determines the names of the input and output trigger elements.
- Determines the chain signatures.
- Creates and returns a ChainDef object.

1.3. **Interaction with the environment.** The entrance point to the software is the function:

```
JetDef.generateHLTChainDef(caller_data, use_atlas_config=True):
```

The caller_data argument is a dictionary. The dictionary format specification can be found at:

```
https://svnweb.cern.ch/trac/atlasoff/browser/Trigger/
TriggerCommon/TriggerMenu/trunk/python/menu/SignatureDicts.py
```

Date: 30 April 2014.

The argument use_atlas_config determines whether Atlas python configuration classes are instantiated. JetDef produces its own Algorithm classes that act as proxies to the Atlas configuration classes. This allows JetDef development to proceed even if the Atlas classes are not available. If use_atlas_config is set to True - as is the case when JetDef is used by the central menu code - objects of these classes are converted to Atlas configuration objects.

JetDef is designed to be run standalone or to be imported and used by the central menu code. Standalone running allows convenient development and debugging. Typing 'python JetDef.py' at the command line will dump the ChainDef objects corresponding to the strawman menu. The command 'python run_testsuite.py' will run the unit test suite.

1.4. **Module organization.** Fig. 1 shows the dependencies among the main modules and the important methods.

2. Overview of sequence of actions

2.1. **Preliminaires.** [JetDef.generateHLTChainDef]

The function JetDef.generateHLTChainDef is passed a dictionary of input data. The desired Algorithm instantiator is created using the instantiator factory. Both dictionary and and an instantiator are passed to the next step.

The following calculation returns a ChainDef object. If problems are encountered deeper inside the code, an exception is raised. The exception is caught in this method, and an ErrorChainDef is returned. Printing ErrorChainDef objects often provides useful debug information.

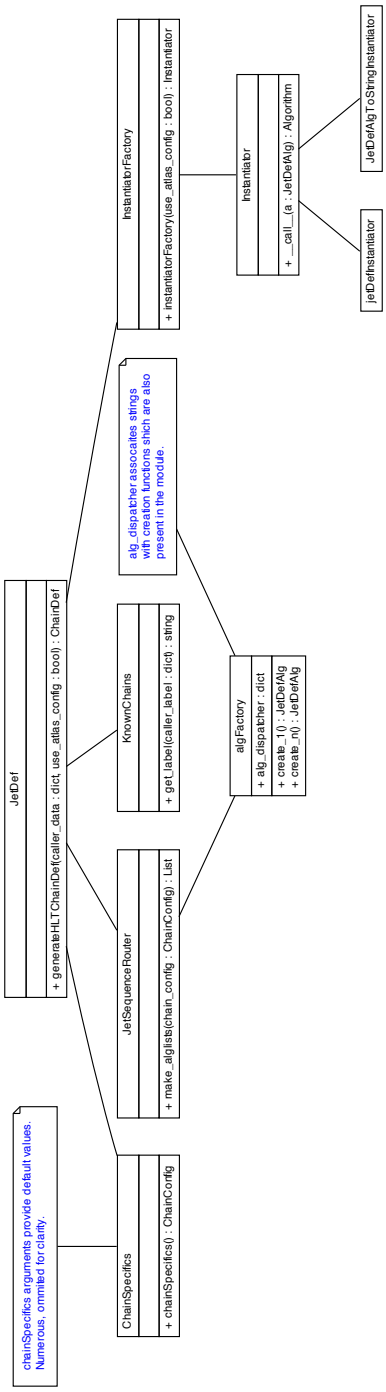
2.2. **Initial checks on input dictionary.** [JetDef._fix_input, JetDef._check_inputs]

Some adjustment of the names in the input dictionary are made to maintain some of the 2012 naming conventions (this may be changed in the future). Checks are made in JetDef._check_inputs to ensure that the input dictionary has the data necessary for JetDef to run.

2.3. **Check chain is to be built is known to JetDef.** [JetDef._make_chaindef, JetDef._decode, KnownChains.getLabel]

The input dictionary contents are checked to see if they correspond to a chain known to JetDef. If so, a string label that uniquely identifyies the chain is passed back to JetDef._decode. If the input dictionary contents is recognized, it is condensed and rearranged into a new dictionary for convenience.

FIGURE 1. Important modules of the JetDef software



2.4. **Additional information is added to the input data.** [JetDef._decode, ChainSpecifics.chainSpecifics]

The input dictionary provides enough information to unambiguously identify the chain to be built, but not enough information to build it. The ChainSpecifics.chainSpecifics function combines the incoming data with extra necessary data. It further packs this information by sequence. Currently the 'tt' (trigger tower scan), 'jr_fex' (JetRec) and 'jr_hypo' sequences are supported. This phase finishes with the creation of a ChainConfig object which contains MenuData objects. MenuData objects contain fex and hypo data for a single sequence.

2.5. **Sequence algorithm list creation.** [JetDef._make_chaindef, JetSequenceRouter.make_alglists] The ChainConfig object is sent to JetSequenceRouter.make_alglists. This examines which MenuData objects are present to determine the sequences to be created. The MenuData objects are routed to the methods that examine them, and use algFactory to obtain the appropriate JetDef Algorithm object. A list of JetDef Algorihm objects is returned for each sequence that is to be built.

2.6. **Sequence creation.** [JetDef._make_chaindef, JetDef._make_sequences] The chain_name, name of the alg list and L1 name are used to create the input and output trigger names. These together with the alg lists complete the sequences specification.

2.7. **Algorithm instantiation.** [JetDef._make_chaindef, Instantiator] The instantiator is passed to the sequences, and instantiates the JetDef Algorithms. The action of instantiation depends on the instantiator class used (that is, it depends on the instantiator produced by the instantiator factory). The action range from returning a JetDef Algorithm object (*i.e.* do nothing), convert the JetDef Algorithm to a string, or convert the JetDef Algorithm to an Atlas Athena configuration object. The last possibility is used in production.

2.8. **ChainDef instantiation.** [JetDef._make_chaindef]

The components required for ChainDef instantiation are now available. The ChainDef object is created and returned.

3. Example: modifying JetDef to allow for a new chain with a new hypo algorithm

The outline of actions to take presented here assumes that the sequence structure does not need changing. Rather, the information to create the new hypothesis algorithm will be present in the input dictionary. The hypothesis algorithm will run in the jr_hypo sequence.

- Ensure that JetDef._check_input performs the checks required to ensure the that input dictionary contents suffice for the production of

- In KnownChains.getLabel, perform adequate checks to determine if the chain is known. Return a sensible for the new chain.