AMES (V4.0) TestCase File Specification

Sean L. Mooney, Dheepak Krishnamurthy

April 11, 2017

Abstract

This is a working document exploring file formats for the AMES (V4.0) TestCase input files.

1 Introduction

The AMES (V4.0) TestCase file describes the basic parameters and characteristics of the TestCase. Data sections in the file include: global parameters, basic generator characteristics, transmission grid characteristics, etc. The original file format (e.g. AMES-2.05 release) uses a general file format that is made up of clearly marked data sections, where the data in each section is a matrix which each row and column having a specific meaning. Though functional, this input format is brittle and as AMES (V4.0) continues to evolve and add new data sections, it would be good to find a more general file format that will simplify the amount of custom parsing for each data section, as well as make it obvious that the correct data is being assigned to the correct internal parameters.

2 Data Sections

Almost all of the data sections, with the exception of the ZoneName section, are comprised of nest key/value type data declarations. Each of these sections is comprised of the 'section type', followed by a list of agents defined for the section, along with values for each agent in the section. The agents appearing in the TestCase configuration file are either LSEs or GenCos.

2.1 Dictionaries

The data declarations could borrow python's dictionary initializer syntax, which is composed of a comma separated list of key/value pairs. Each dictionary is enclosed in curly-braces ({}). Using this format also removes the need for a strict line based (an entry per line) data file specification. An example dictionary of SCUC input parameters might look like:

```
{MinUpTime : 8, MinDownTime : 4, NominalRampUp : 22.4,
 NominalRampDown : 22.4, StartupRampLim : 400, ShutdownRampLim : 448}
```

2.2 Associating data

The next task, after defining a dictionary, is to associate a dictionary with the type of data the dictionary stores, as well as with the entity to which the data belongs. This document proposes two different ways to declare these associations.

2.3 Defined data sections

The first proposal is a variation on the current file format. Each data section is prefixed by a `DATA_SECTION_Start` and closed with a `DATA_SECTION_END` declaration. Nested data sections are not supported. The dictionary based approach would simple list each agent by name, followed by the dictionary of attributes for the agent. Each entry must begin on a new line.

```
#ScucInputStart
GenCo1 {MinUpTime : 8, MinDownTime : 4, NominalRampUp : 22.4,
        NominalRampDown : 22.4, StartupRampLim : 400, ShutdownRampLim : 448}
GenCo2 {MinUpTime : 8, MinDownTime : 4, NominalRampUp : 22.4,
        NominalRampDown : 22.4, StartupRampLim : 400, ShutdownRampLim : 448}
#ScucInputEnd
```

2.4 Data section declarations

A second way to associate the type of the information with the entity it belongs to is to require each dictionary to be prefixed by both the data section and the agent name.

```
SCUCInput GenCo1 <- {MinUpTime : 8, MinDownTime : 4,
                     NominalRampUp : 22.4, NominalRampDown : 22.4,
                     StartupRampLim : 400, ShutdownRampLim : 448}
```

Tradeoffs The approach in §2.3 minimizes the number of times a data section must be delared, but is less flexible. The approach in section §2.4 is more flexible (e.g. all the data for a single genco can be declared in one section of the file), but will require more processing in the second stage of the reader to assemble the correct model.

3 Implementation Notes

The proposed format changes are designed to simplify the amount of custom code that must be written for each data section, as well as increase the obvious clarity of what each piece of data represents.

The biggest challenge with the columnar format as it currently exists is the lack of semantic information associated with any given piece of data. Any entry is assigned meaning only by virtue of its index. The implied meaning makes inspection of the code for correctness hard and makes adding a new column tedious.

These changes simplify the parsing allowing data to be declared with its meaning. The data file reader then only needs to add an entry for each key/value encountered in the data file. A second step of processing can assemble the internal representation of the data model by just looking up the values for certain keys. A missing keys indicates missing data/error in the TestCase.

The dictionary approach also means reading new data sections is semi-automatic. Since the format between sections is common, the only elements that need to be added to the reader are for looking up/assigning the values to appropriate part of the internal model.