Casal2

Input File Specifications

Syntax Specifications for Casal2 Input Files

v2019.11

Author:
Scott Rasmussen
Zaita
scott@zaita.com

Table of Contents:

Document History

Version	Description	Author	Date
1.0	*Initial version - Draft*	S.Rasmussen	20/11/2012
V2016.1	*Updated for Casal2 information and actual func*	S.Rasmussen	20/01/2016
V2019.11	*Updated for Casal2 v1.0*	S.Rasmussen	14/11/2019

Introduction

This document outlines the different types of input files that can be given to Casal2 as part of the different run modes.

The files that Casal2 accepts are:

1. Casal2 Configuration Files – *This file contains the Model Definition Language specification for the model to be constructed and run by Casal2. For the majority of model runs, these will be the only input files used.*
2. Estimable Values File – *This file contains a list of values to be used for different sequential runs of the model for estimables/estimates.*
3. MCMC Objective File – *This file contains the covariance matrix and objective values from a previous MCMC run. This information can be passed to Casal2 to resume an MCMC run.*
4. MCMC Samples File – *This file contains the list of estimate values used during each iteration of the MCMC run. This information can be passed to Casal2 to resume an MCMC run.*
5. MPD File – *This file contains the estimates values and covariance matrix produced by a successful optimisation/minimisation. This information is used to start an MCMC run.*

The rest of this file will go into each file structure/syntax independently with examples of how the file is to be written.

Casal2 Model Definition Language Files

Casal2 uses a text files to define the model to be built and run. Models can be specified in either Windows or Unix line endings with UTF-8 encoding. Casal2 by default looks for config.csl2, this file name can be overridden with the -c command line parameter.

The Casal2 model definition language (MDL) is based on the Spatial Population Model (SPM) format with enhancements. There is no direct link between the CASAL and Casal2 model definition file formats.

Casal2 uses a block, parameter notation. All objects in the model are defined first within a parent @block. Every block has one or more parameters defined after it. The Casal2 MDL is a hierarchical structure where all parameters belong to the block defined before it. The start of a new @block definition changes the current scope for the MDL.

An example of two blocks being defined, each with multiple parameters below them is below:

```
@block1 label
parameter value
parameter value1 value2

@block2 label
parameter value
parameter value3 value4
```

Some general notes about writing configuration files:

1. Whitespace can be used freely. Tabs and spaces are both accepted
2. A block ends only at the beginning of a new block or end of the final configuration file
3. You can include another file from anywhere
4. Included files are placed inline, so you can continue a block in a new file
5. The configuration files support inline declarations of objects

Including Files

While Casal2 looks for config.csl2 as the entry point to the Model specification, Casal2 supports the including (chaining) of multiple files. This allows the user to break their complex models into separate logical parts, some of which can be even generated by external scripts/systems.

The inclusion of configuration files in the Casal2 MDL is done in a way very similar to the C++ MACRO concept. Casal2 will load the entire model definition language into memory as a singular in-memory file. Contents of include files will be copied into the locations where they have been specified before processing begins. Casal2 will track the origin (file name and line number) of every line loaded for error handling and reporting.

When including a file, you use the following syntax:

File: config.csl2

```
@process my_made_up_process
type mortality_with_table_of_m
categories *
!include "mortality_table_data.csl2"

@block next_process

Type ageing
```

```
Categories *
```

File: mortality_table_data.csl2

```
table m_data
year male female
1995 0.04 0.08
1996 0.03 0.09
1997 0.03 0.09
1998 0.02 0.10
end_table
```

Casal2 does not itself require case sensitivity for include file names, but Linux type Operating Systems do. It's recommended that you ensure case sensitivity when specifying include file names.

Casal2 by default uses the .csl2 file extension when looking for the entry point config.csl2, but there is no expectation for any of the include files to use this file extension. You can use any file extension you prefer.

Model Definition Language Structure

The name given to the format of the Casal2 configuration file is the "Casal2 Model Definition Language". It's a structured declarative language that allows a user to specify a custom model for building and execution by the Casal2 engine.

The Model Definition Language (MDL) is built from 3 primary structures: The block, the parameter and the value(s).

The Block and Object

A Casal2 model is built dynamically from different blocks as if they were lego pieces. The relationships between the different blocks determines how the model will be built and how it'll execute.

The entire model structure can be thought of as a hierarchical tree structure of blocks with everything branching from the top-level "@model" block specification. In some way, every other block in the system will eventually have a relationship to the model.

Every individual process, observation or report (etc) will be specified as a block in the MDL and assembled by Casal2. In most cases there is a 1:1 relationship between the block specification in the MDL and what Casal2 creates in memory. The only exception to this is the @estimate block (discussed later).

The block is what is defined in the configuration file. The object is the in-memory representation of that block that is built by the Casal2 engine.

Addressables

When blocks are constructed by Casal2 into objects they will register named parameters. These named parameters can be searched by other objects within the model to read/write their values.

In the Casal2 code base, an object will RegisterAddressable(NAME, USAGE) where name is the name for it to be searched on. The USAGE flag tells Casal2 what that addressable can be used for (e.g. time varying, estimation, reporting).

Addressables are used by most objects that want to communicate with other objects. When a report prints out a value, most times it's referencing an addressable on the target object.

All parameters used in estimations/MCMCs are addressables that have been registered with the "Estimate" flag.

The addressable system provides a nice, easy and consistent way for blocks/objects in Casal2 to

expose parameters to other blocks/objects for use.

Estimates

A parameter value that can be estimated or used in an MCMC run (i.e. a free parameter) is called an estimate. This means that it is an addressable that has been registered with the estimate flag.

Keywords and Reserved Characters

In order to allow efficient creation of input files, the MDL contains special keywords and characters that cannot be used for labels or parameter names.

'@' Block Definitions

Every block in the configuration file must start with the block definition character. The reserved character for this is the '@' character.

Example:

```
@block1 <label>
type <type>

@block2 <label>
type <type>
```

'type' Keyword

The 'type' keyword is used for declaring the sub-type of a defined block. Any block object that has multiple sub-types will use the type keyword. Almost every block in the MDL supports the type parameter, the only known exception to this is @categories.

Example:

```
@block1 <label>
type <sub_type>

@block2 <label>
type <sub_type>
```

(Single-Line Comment)

Comments are supported in the configuration file in either single-line (to end-of-line) or multi-line

Example:

```
@block <label>
type <sub_type> #Descriptive comment
#parameter <value_1> - This whole line is commented out
parameter <value_1> #<value_2>(value_2 is commented out)
```

/ and */ (Multi-Line Comment)*

Multiple line comments are supported by surrounding the comments in /* and */. The MDL uses a syntax the same as C++ for this.

Example:

```
@block <label>
type <sub_type>
parameter <value_1>
```

```
parameter <value_1> <value_2>

/* Do not load this process
@block <label>
type <sub_type>
parameter <value_1>
parameter <value_1> <value_2>
*/
```

'*' (Wildcard)

When referencing categories Casal2 supports the use of the wildcard operator to select all categories. It's important to note that the wildcard operator will preserve the order of definition of category names.

Example:

```
@categories
format sex.stage
names male.immature male.mature female.immature female.mature

@process my_recruitment_process
type constant_recruitment
categories * #Will select all categories defined preserving order
```

':' (Range Specifier)

The range specifier allows you to specify a range of values at once instead of having to input them manually. Ranges can be either incremental or decremental.

Example:

```
@process my_recruitment_process
type constant_recruitment
years_to_run 2000:2007 #With range specifier

@process my_mortality_process
type natural_mortality
years_to_run 2000 2001 2002 2003 2004 2005 2006 2007 #Without range specifier
```

',' (List Specifier)

When a parameter supports multiple values in a single entry you can use the list specifier to supply multiple values as a single parameter.

Example:

```
@categories
format sex.stage
names male,female.immature,mature #With list specifier

@categories
format sex.stage
#Without list specifier
names male.immature male.mature female.immature female.mature
```

'+' (Join Operator)

Some observation blocks allow the joining of multiple categories into a single collection for processing. Casal2 supports the use of a join operator for observations that support this.

```
@observation
```

```
type joinable
categories male+female female #Will show as 2 collections to observation
```

Combining Operators

Casal2 supports the use of multiple operators together in a single parameter definition.

Example:

```
@process my_recruitment_process
type constant_recruitment
years_to_run 2002:2005,2007:2009 #Multiple operators, skips 2006

@process my_mortality_process
type natural_mortality
years_to_run 2002 2003 2004 2005 2007 2008 2009 #Without operators
```

The join operator can also be combined with the wildcard operator to select all categories as a single collection.

Example:

```
@categories
format sex.stage
names male.immature male.mature female.immature


@observation my_joinable

type joinable

categories * #Will return: male.immature male.mature female.immature

categories *+ #Will return: male.immature+male.mature+female.immature
```

'table' and 'end_table' Keywords

Many blocks in Casal2 require tables of data. Casal2 supports space separated values. The table is wrapped with a ‘table’ and ‘end_table’ pair of keywords to denote the start and end of the table.

Where the table requires column headers to be specified, these must reside on the line directly after the ‘table’ keyword.

Following lines are rows of the table. Each row must have the same number of values as the number of columns specified.

The table definition must end with the 'end_table' keyword on its own line.

The first row of a table will be the name of the columns if required.

Example:

```
@block <label>
type <sub_type>
parameter <value_1>
table <table_label>
<column_1> <column_2> <column_n> #Table required column definitions
<row1_value1> <row1_value2> <row1_valueN>
<row2_value1> <row2_value2> <row2_valueN>
end_table
```

'[' and ']' (Inline Declarations)

When a block takes the label of another block as a parameter this can be replaced with an inline declaration. An inline declaration is a complete declaration of a block. This is designed to allow the configuration writer to simplify the configuration writing process.

When declaring a block inline, you can precede the declaration with a label to override the default one Casal2 would create. By default, Casal2 uses parent_block.object_index as the label for inline constructed objects.

Example:

```
#With inline declaration with label specified for time step
#Time step is created with label 'step_one'
@model
time_steps step_one=[type=iterative; processes=recruitment ageing]

#Same declaration as above without inline
@model
time_steps step_one

@time_step step_one
type iterative
processes recruitment ageing

#With inline declaration with default label (model.1)
@model
time_steps [type=iterative; processes=recruitment ageing]

#Same declaration as above without inline
@model
time_steps model.1

@time_step model.1
type iterative
processes recruitment ageing
```

Categories

Categories in Casal2 are one of the few things that are semi-fixed in terms of logic. They're typically a 2-dimensional data structure. The first dimension is the name of the category and the second dimension is the age/length/size/etc. The first dimension is fixed in that Casal2 always requires some form of unique lookup value (i.e. name) for each category.

By default, all categories inherit properties from the model block. This includes ages, lengths and years. Each of these values can be overridden at the category level to provide more customisation within the model. (e.g. the years a category is valid for can be modified within the @category definition).

Defining Categories

Because each category is quite complicated, the syntax for defining categories has been structured to allow complex definitions using a simple short-hand structure.

The "format" parameter is required and allows you to specify the category hierarchy to be expected by the model. By using a "." (period) character between each segment the model can build a hierarchical representation of the categories for short-hand lookups and validation.

The "names" parameter is a list of the category names. The syntax of these names will need to match the "format" parameter allowing the model to organise and search on them. Using the "list specifier" and range characters we can shorten this parameter significantly.

Example:

```
@categories
#Format to use for all examples below
format sex.stage.tag

#Good category definition with 4 categories
names male.immature.notag male.immature.2001 male.mature.notag male.mature.2001

names male.immature #Invalid: No tag information
names female #Invalid: no stage or tag information
names female.immature.notag.1 #Invalid: Extra format segment not defined

#Good definition using short-hand
names male,female.immature,mature.notag,2001:2005

#Same definition without short-hand. You'd have to write:
names male.immature.notag male.immature.2001 male.immature.2002
male.immature.2003 male.immature.2004 male.immature.2005 male.mature.notag
male.mature.2001 male.mature.2002 male.mature.2003 male.mature.2004
male.mature.2005 female.immature.notag female.immature.2001
female.immature.2002 female.immature.2003 female.immature.2004
female.immature.2005 female.mature.notag female.mature.2001 female.mature.2002
female.mature.2003 female.mature.2004 female.mature.2005
```

Overriding the active years

When we have specific data for a year in a category, we don't want the model to process this category during other years (or the initialisation stages). We can define a list of years where each category will be available, this will override the default of all years in the model. Any category where you overwrite the default will no longer be accessible in the initialisation phases.

Examples:

```
@model
start_year 1998
final_year 2010

@categories
format sex.stage.tag
names male,female.immature,mature.notag,2001:2005 #OK!
years tag=2001=1999:2003 tag=2005=2003:2007
#Categories with the tag value "2001" will be available during years 1999,
2000, 2001, 2002 and 2003
#Categories with the tag value "2005" will be available during the years 2003,
2004, 2005, 2006, 2007
```