SAND2015-6869 TR




*Exceptional service in the national interest*

# Dakota Software Training

## Input Syntax: Configuring Dakota Components

http://dakota.sandia.gov



Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

# Module Learning Goals

- Develop an accurate “mental model” of Dakota components
- Understand how to configure Dakota components using a Dakota input file
- Become familiar with the Dakota Reference Manual


Sandia National Laboratories

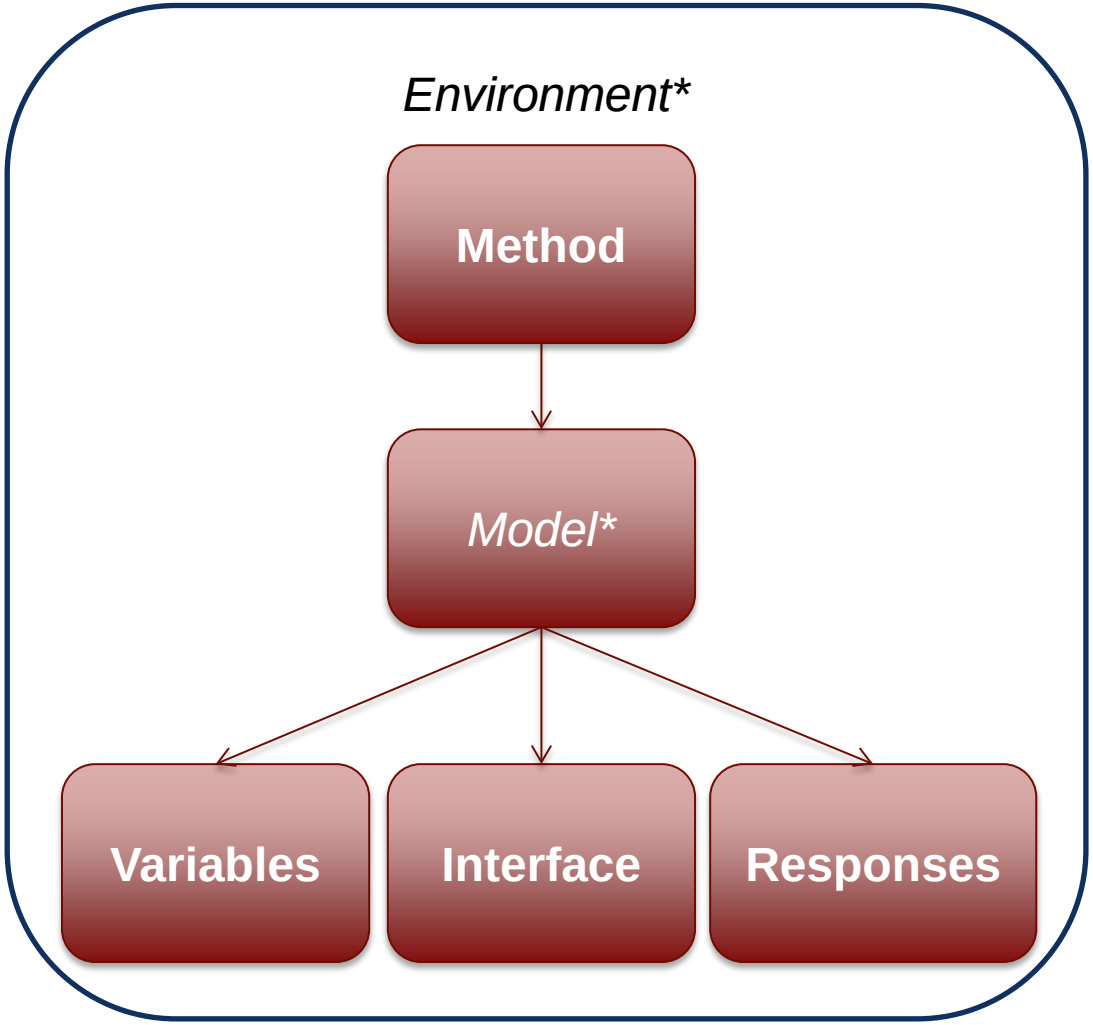
Dakota Input and Components

# DAKOTA COMPONENTS

# Dakota Blocks

Every Dakota study is composed of six types of blocks

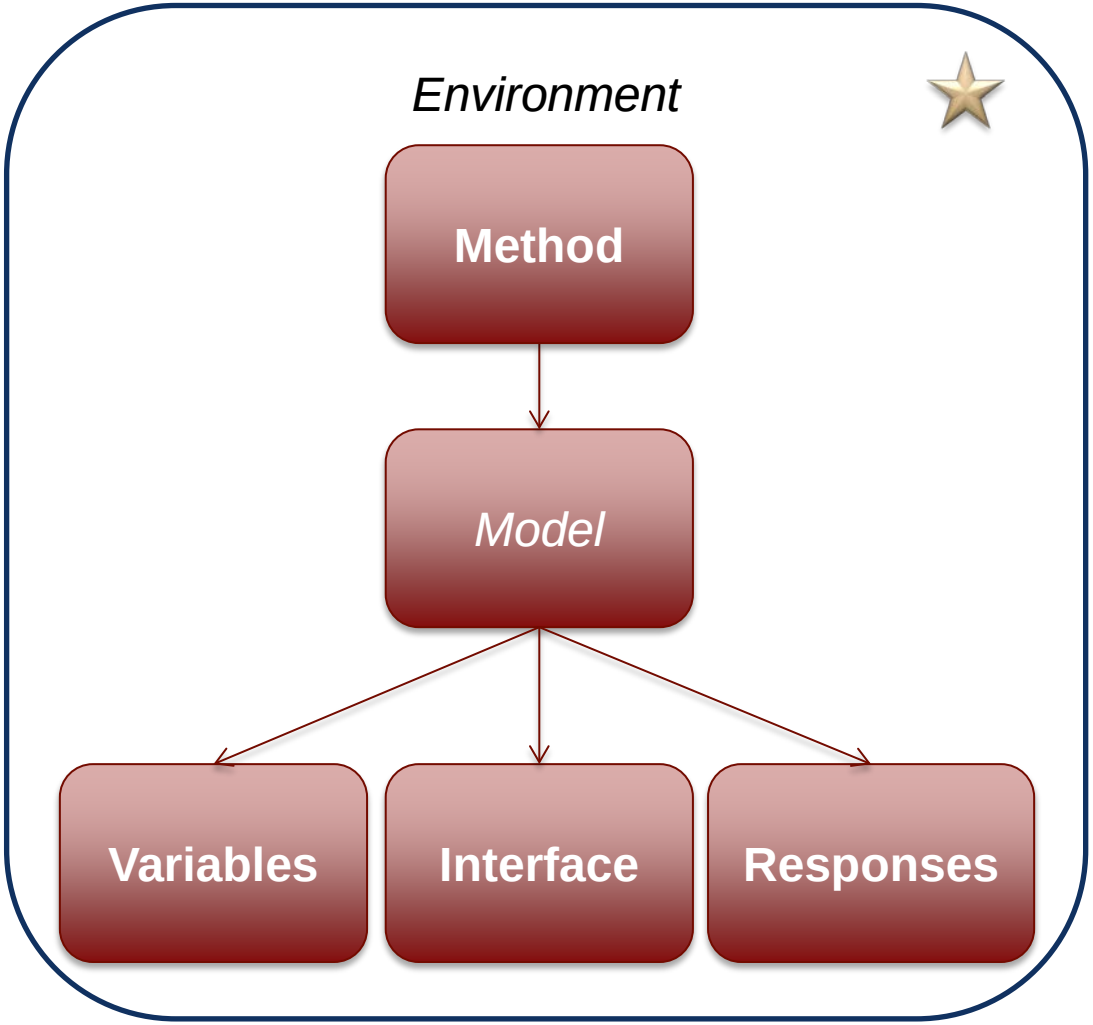
- *Environment**
- **Method**
- *Model**
- **Variables**
- **Interface**
- **Responses**




*Optional

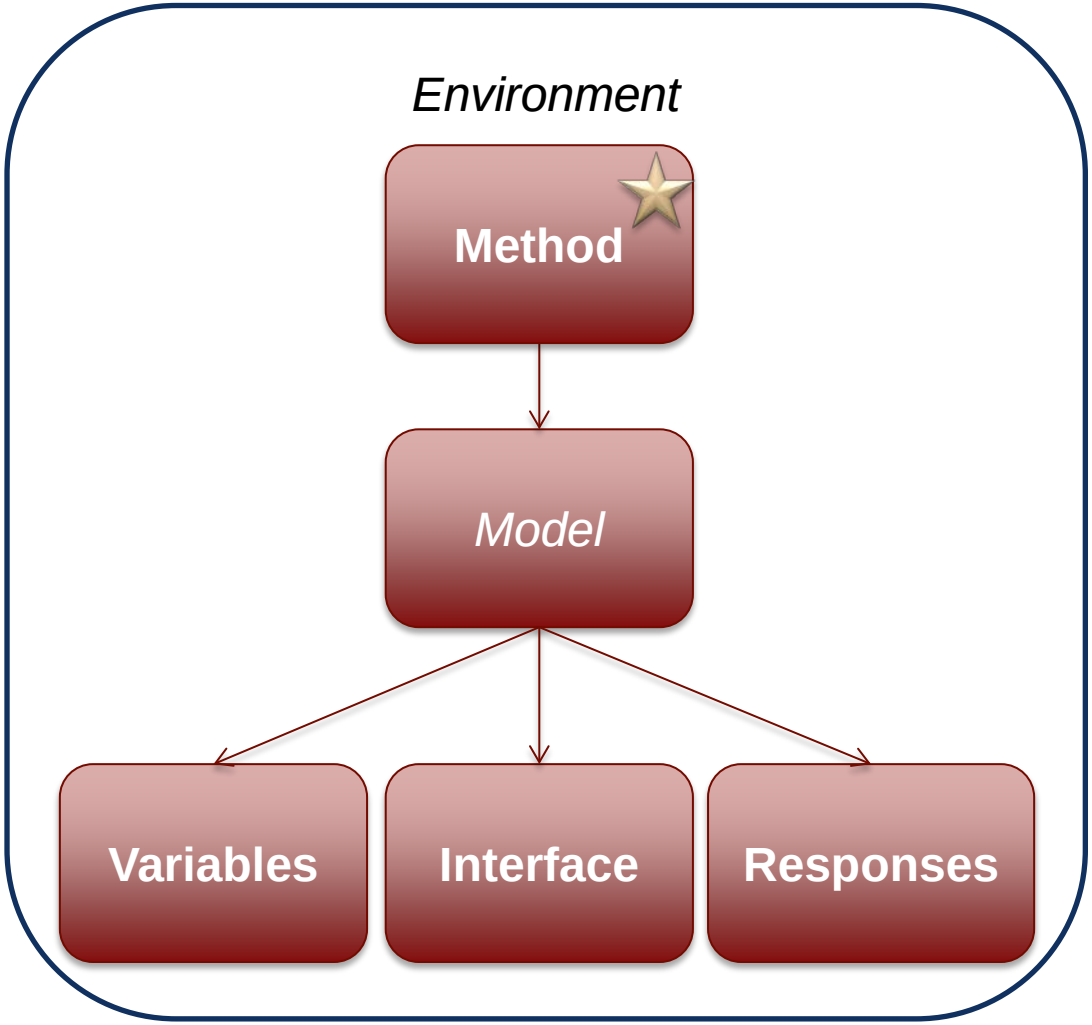
# Environment: General Study Settings

- All other blocks are embedded in one environment block
- Example settings
  - Output precision
  - Name of output & error files
- At most one environment specification is permitted
- Environment specification is *optional*



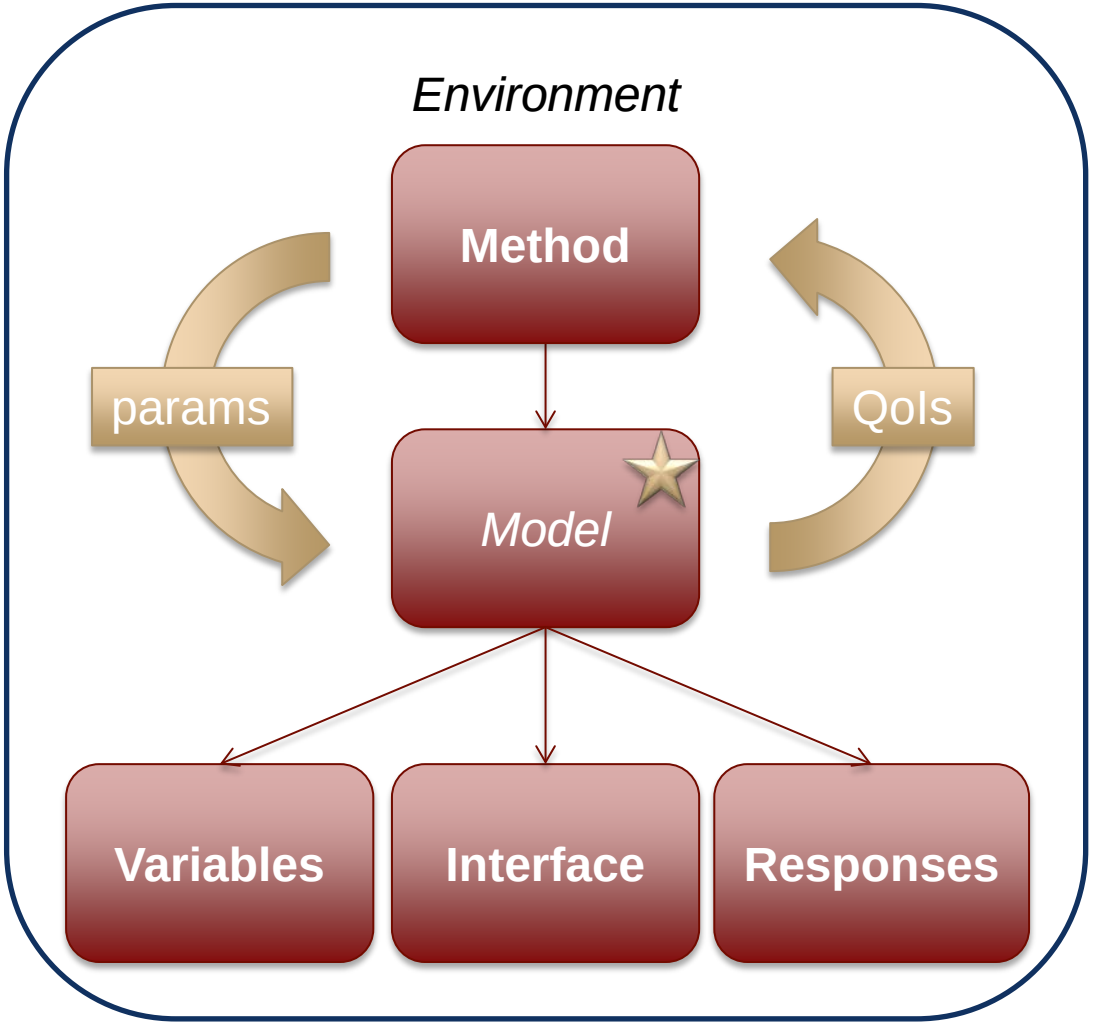
# Method: What Dakota Does

- Specifies an algorithm, e.g., sampling, along with its options
- Invokes the model for a variables-to-responses map
- At least one method block is **required**
  - Multiple may be specified in complex studies



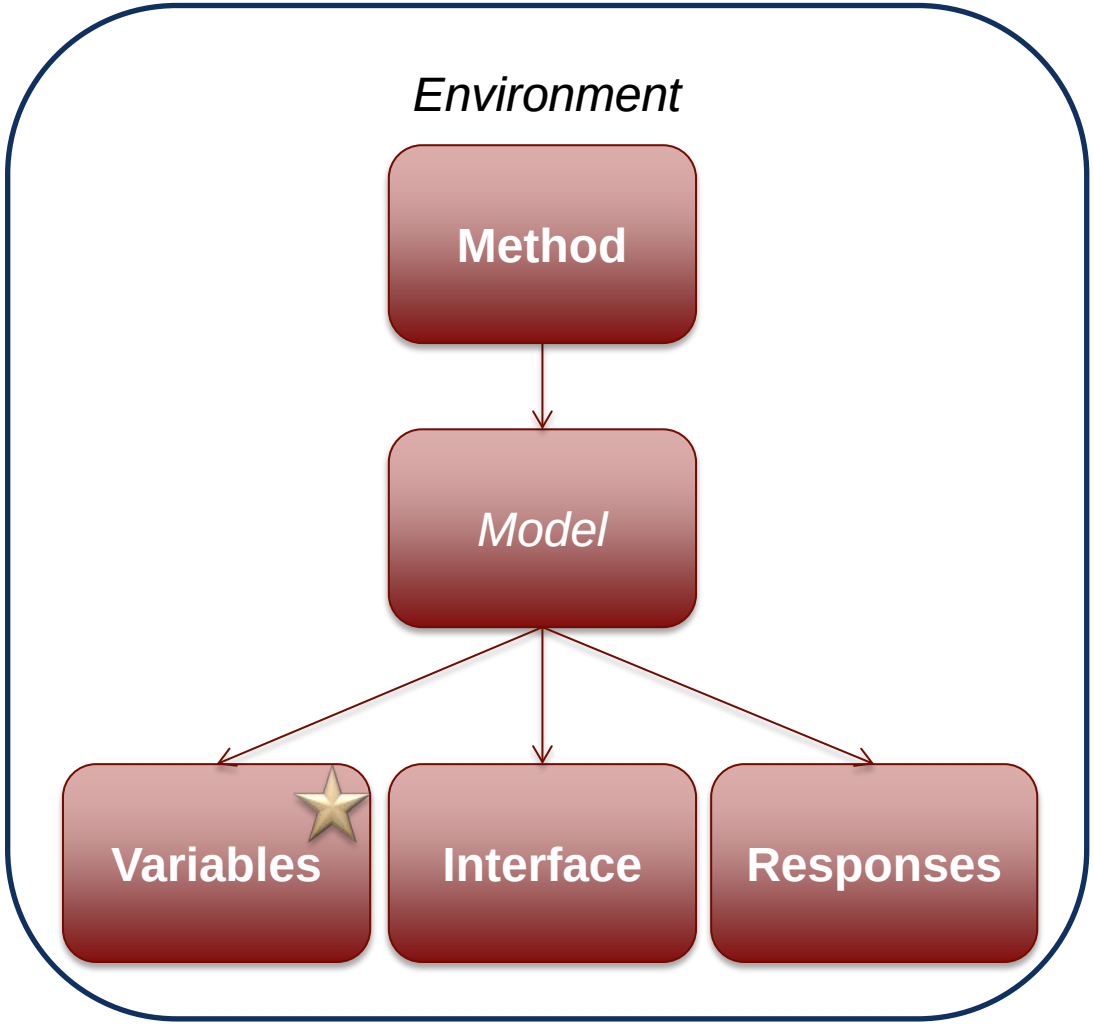
# Model: Maps Variables to Responses

- A model block fulfills requests from a method to compute responses from variables
- Mapping can be through a simulation interface
  - Complex studies may use surrogate model, nested model/method
- Model is *optional* in simple studies
  - Multiple may be specified in complex studies



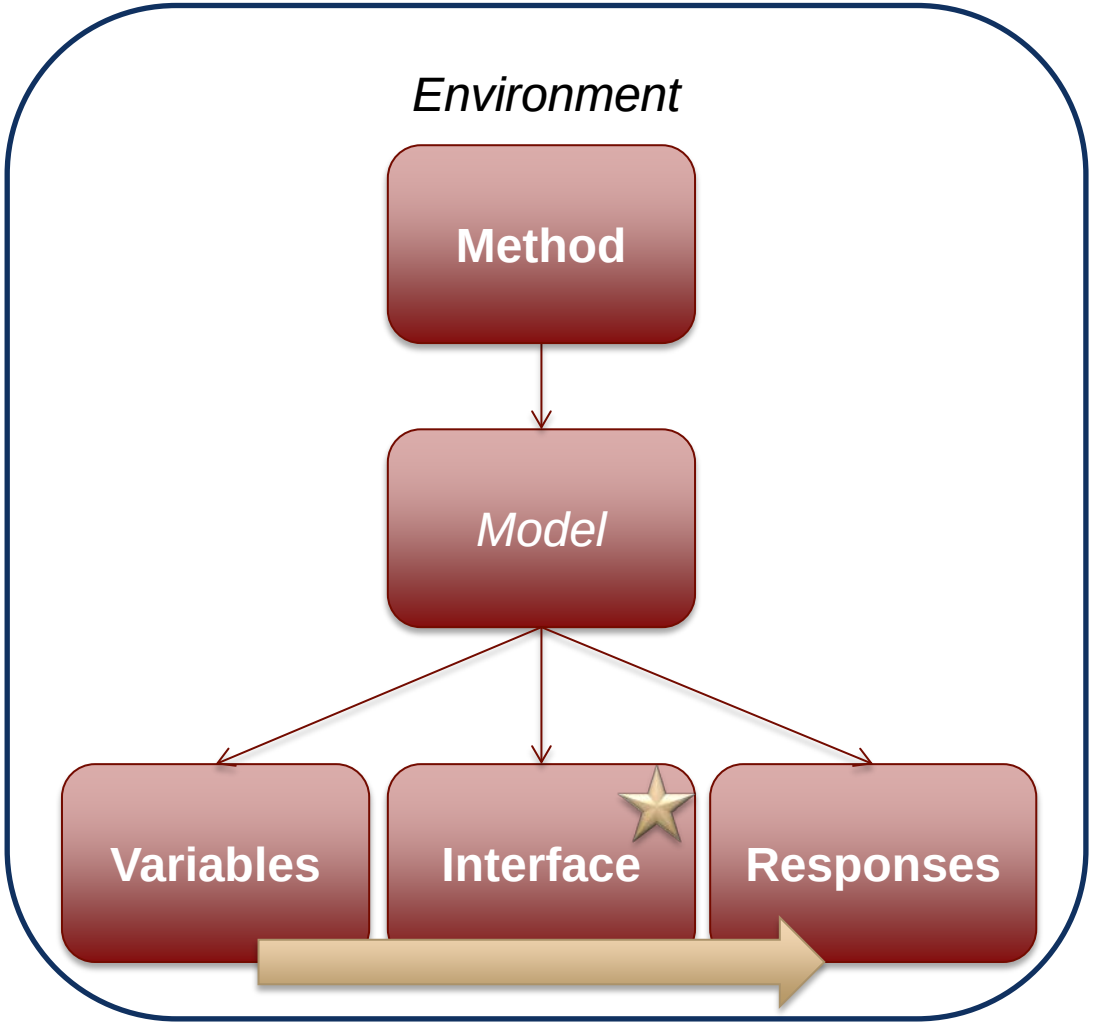
# Variables: What Dakota… Varies

- Specify the number, type, other properties of the parameters
- Types are categorized along three dimensions
  - Design, uncertain, or state
  - Continuous or discrete
  - Real, integer, or string
- *Design* variables are used in optimization and calibration
- *Uncertain* variables are used in UQ and sensitivity studies
- *State* variables are typically fixed parameters and not varied by the method
- At least one variables block is **required**



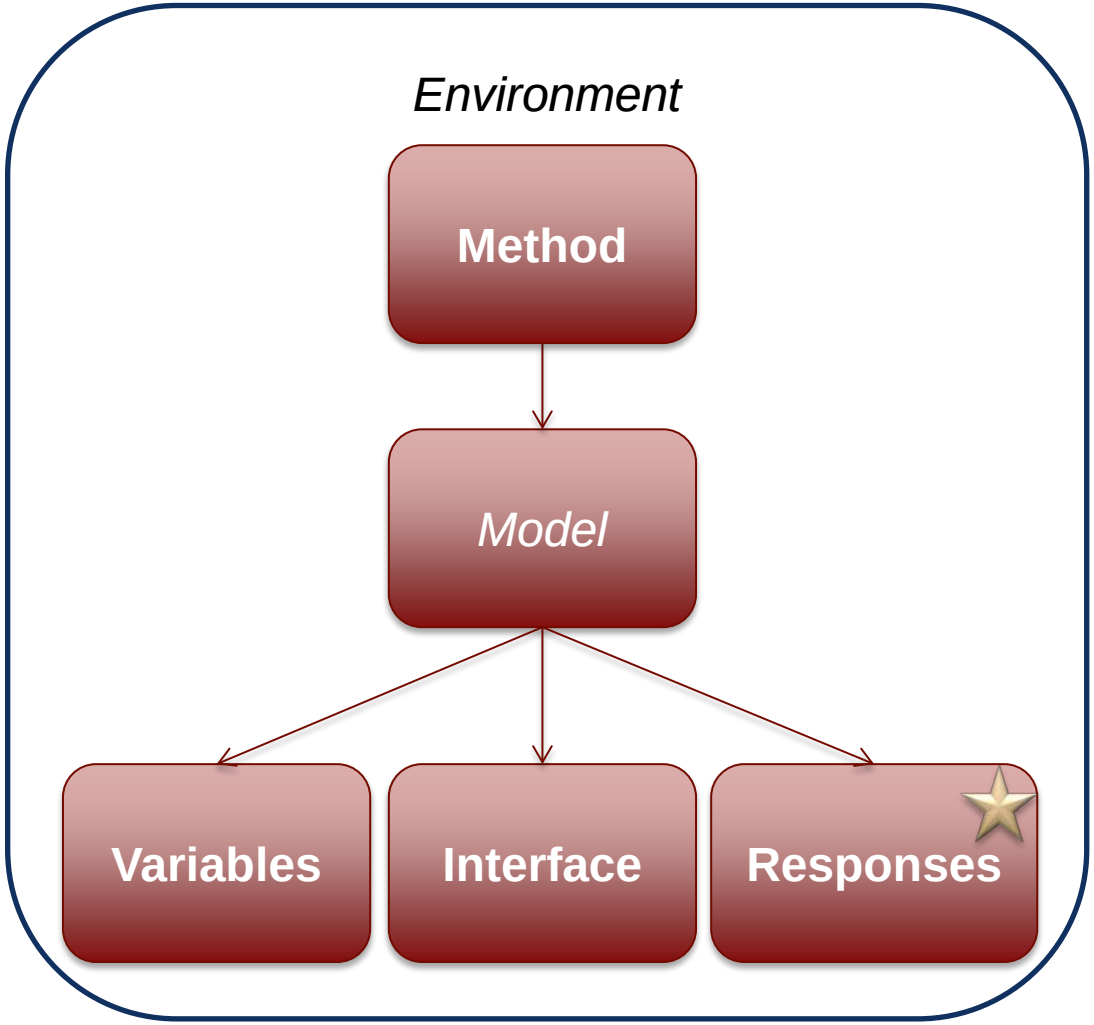
# Interface: Communicate with a Simulation

- Specific values of variables are mapped through an interface to obtain responses at those values
- Settings include
  - Path/name to driver
  - Names of files used in I/O
  - Concurrency settings
- At least one interface specification is **required**



# Responses: Your Simulation Results

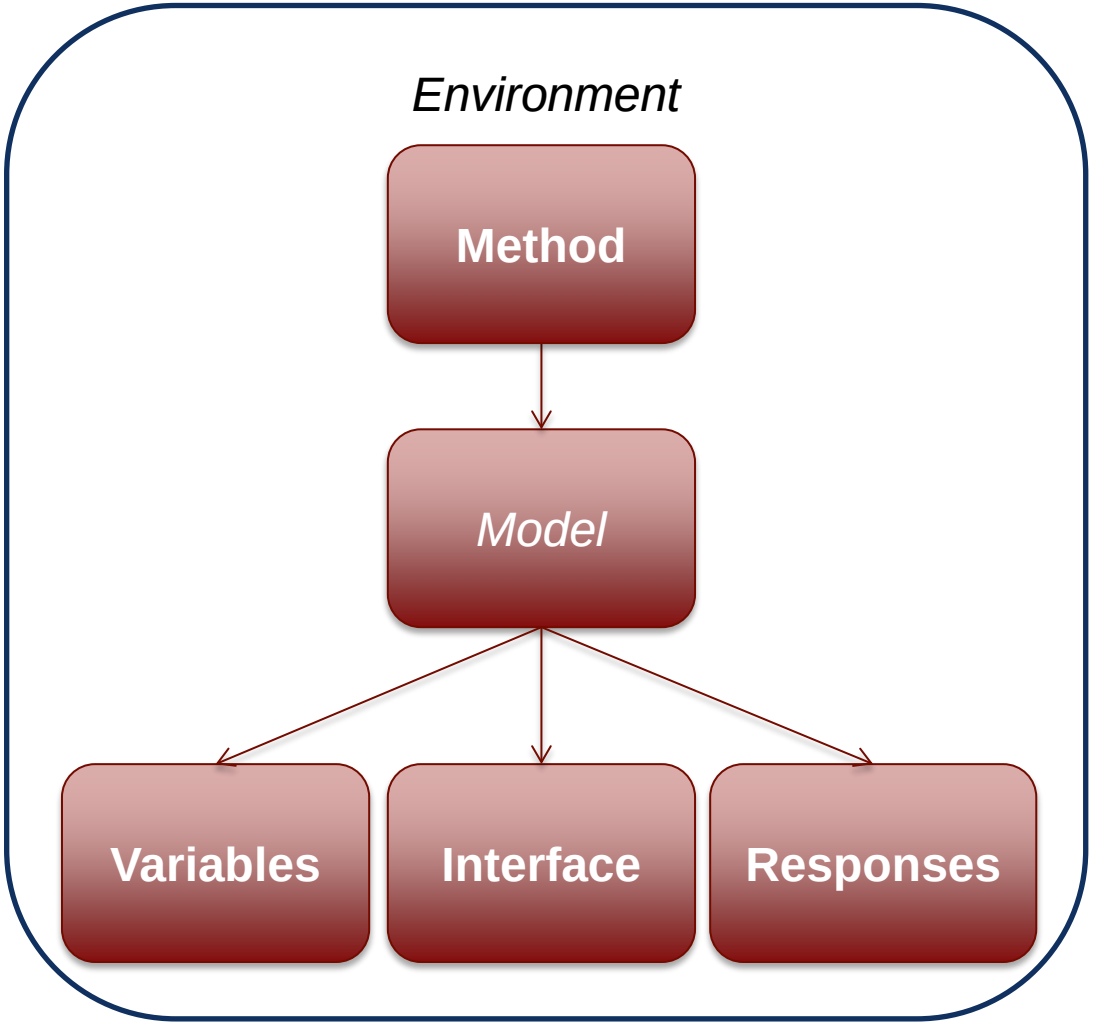
- Data Dakota expects back after setting parameters and running the interface
- Categorized based on usage
  - Objective functions→ optimization
  - Calibration terms → calibration
  - Response functions → SA & UQ
- At least one responses block is **required**



# Dakota Blocks In Summary

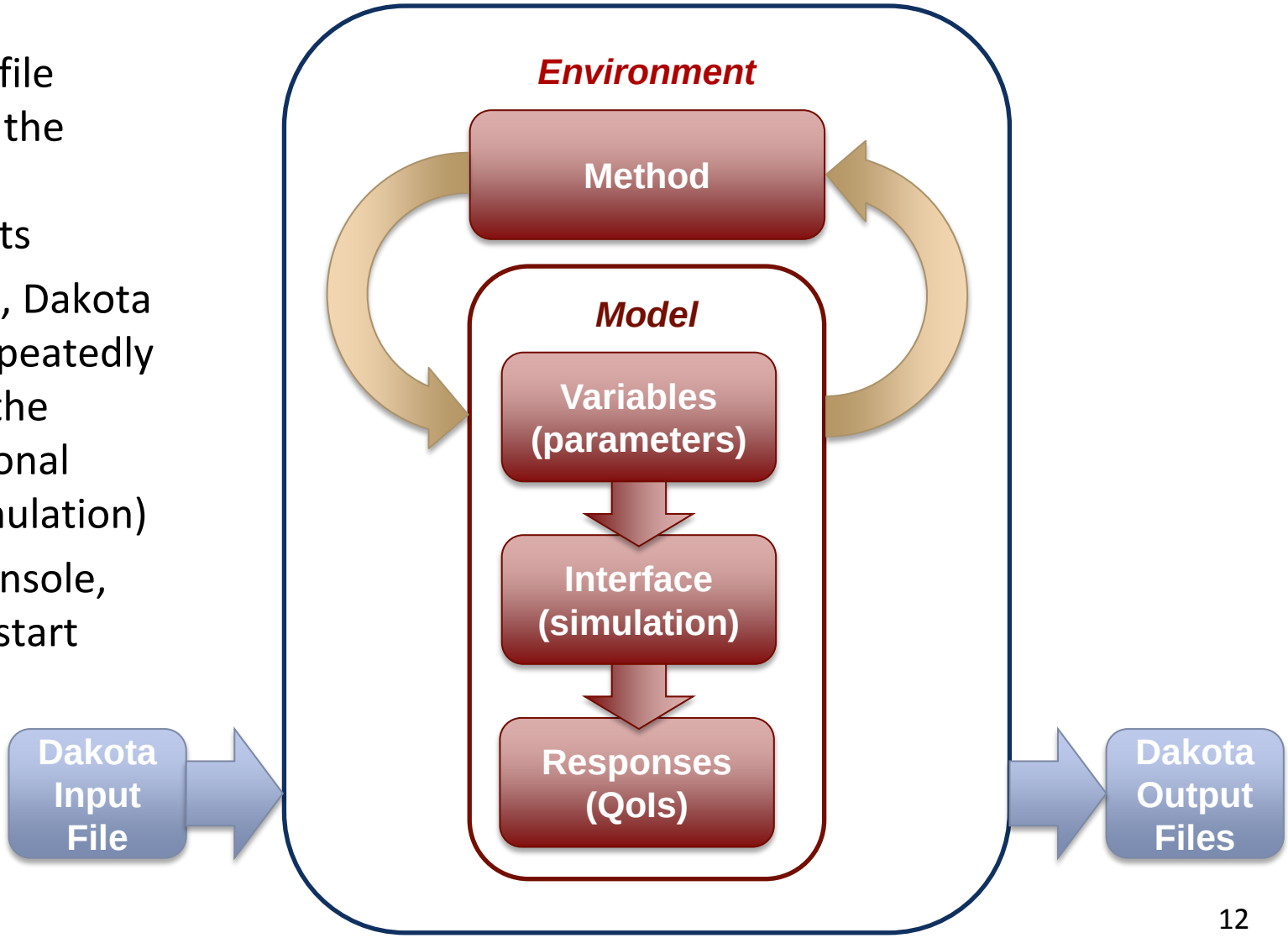
"In each iteration of its algorithm, a **method** block requests a **variables**-to-**responses** mapping from its **model**, which the model fulfills through an **interface***"

*Or surrogate or nested method



# Overall Information Flow

- Text input file configures the Dakota components
- At runtime, Dakota method repeatedly evaluates the computational model (simulation)
- Output: console, tabular, restart

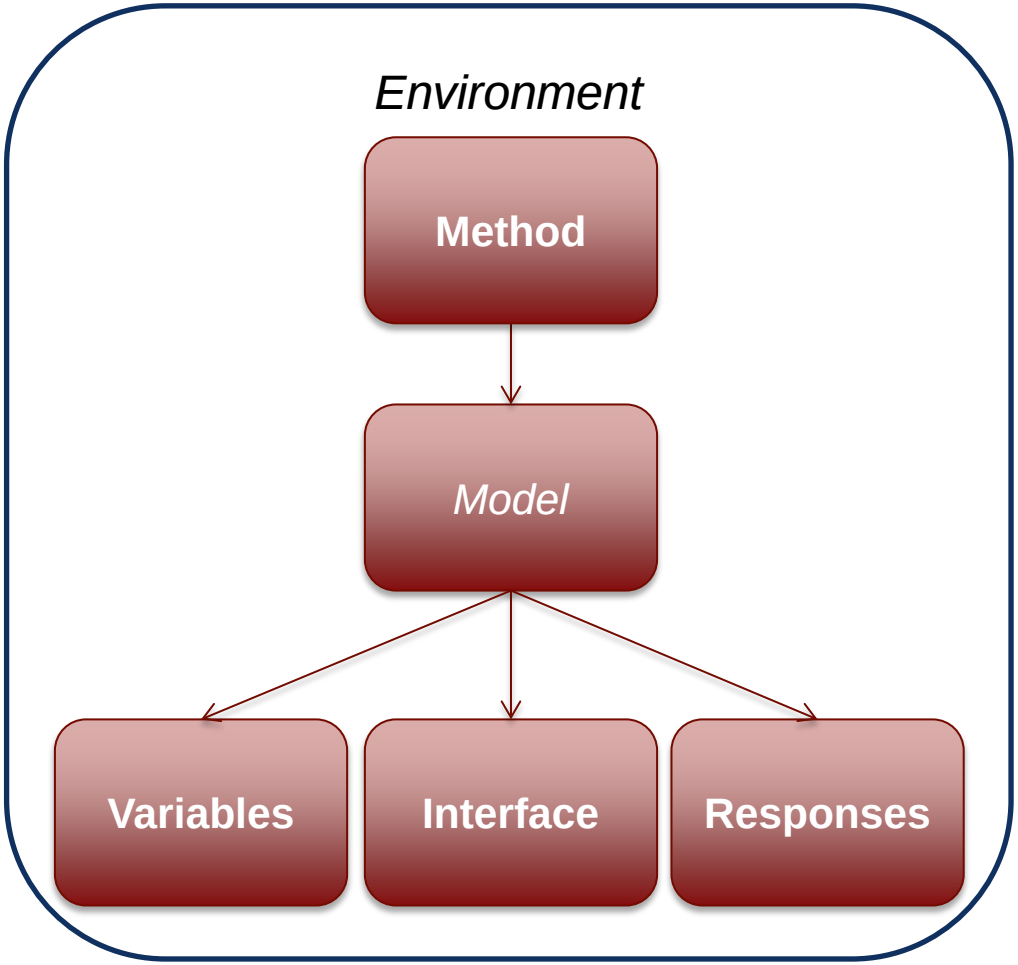


Dakota Input and Components

# DAKOTA INPUT FILE

# Dakota Input

```
environment
    tabular_data output_precision 1e-16

method
  centered_parameter_study
    step_vector 0.1 0.1 2.0 10. 1e5 5. 10.
    steps_per_variable 2

model
  single

variables
  active all
  continuous_design = 3
    initial_point  1.0     1.0     20.0
    descriptors    'w'     't'      'L'
  continuous_state = 4
    initial_state  500.0 29.E+6  50. 100.
    descriptors    'p'    'E'    'X'  'Y'

interface
  fork analysis_drivers = 'driver.sh'

responses
  response_functions = 3
  descriptors = 'area' 'stress' 'displacement'
  no_gradients no_hessians
```



# Input Formatting

See the Input Spec section of the Reference Manual

```
environment
  tabular_data output_precision 1e-16

method
  centered_parameter_study
    step_vector 0.1 0.1 2.0
                10 1.e5 5. 10.
    steps_per_variable 2

variables
  active all
  continuous_design = 3
    initial_point  1.0     1.0    20.0
    descriptors    "w"     "t"     "L"
  continuous_state = 4
    initial_state  500. 29.E+6 50. 100.
    descriptors    'p'   'E'   'X'  'Y'

interface
  fork
    analysis_driver = 'driver.sh'

responses
  response_functions = 3
  descriptors = 'area'
                'stress'
                'displacement'
  no_gradients no_hessians
```

Rules

- "Flat" text only
- Whitespace is ignored
- Comments begin with # and continue to the end of the line
- Keyword order largely unimportant as long as major sections are respected and there is no ambiguity
- Equal signs are optional
- Strings surrounded by single or double quotes (beware of "fancy" quotes)
- Scientific notation is fine
- Shortcuts to be used with care
  - Repeated values in lists: N*Value
  - Ranges: Lower:Step:Upper (Range will include the upper bound)
  - Unambiguous abbreviations (BUT..)

# Input Best Practice: Block Pointers and IDs

```
environment
    tabular_data output_precision 1e-16

method
  id_method 'pstudy'
  model_pointer 'pstudy_model'
  centered_parameter_study
    step_vector 0.1 0.1 2.0 10. 1e5 5. 10.
    steps_per_variable 2

model
  id_model 'pstudy_model'
  variables_pointer 'cantilever_vars'
  interface_pointer 'cantilever_inf'
  responses_pointer 'cantilever_resp'
  single

variables
  id_variables 'cantilever_vars'
  active all
  continuous_design = 3
    initial_point  1.0     1.0     20.0
    descriptors    'w'     't'      'L'
  continuous_state = 4
    initial_state  500.0 29.E+6  50. 100.
    descriptors    'p'    'E'    'X'  'Y'

interface
  id_interface 'cantilever_inf'
  fork analysis_drivers = 'driver.sh'

responses
  id_responses 'cantilever_resp'
  response_functions = 3
  descriptors = 'area' 'stress' 'displacement'
  no_gradients no_hessians
```

- Block pointers and IDs enable explicit tracking
  - To which model a method is applied
  - Which variables, interface, responses comprise a model
- Best practice for simple Dakota input
- Critical for complex Dakota input


Sandia National Laboratories

Dakota Input and Components

# DAKOTA REFERENCE MANUAL

# Dakota Reference Manual

- All permitted Dakota keywords are documented in the Dakota Reference Manual
- The Keywords section of the manual is organized hierarchically and contains
  - Keyword descriptions
  - Usage examples
  - Defaults
  - .. and more
- Let's take a tour… https://dakota.sandia.gov/content/manuals

# Exercise

Goal: Convert a centered parameter study to a different kind of parameter study

1. Copy the example files located in `~/exercises/input/` to a new working directory.
2. Using your favorite text editor, open the copy of `dakota_cantilever_centered.in` that you made. Change it to a different kind of parameter study
   1. **Hint 1:** *Use the Reference Manual!*
   2. **Hint 2:** *You may also have to change the variables block! Check the example on the Reference Manual page!*
3. Run Dakota using your modified input file:

```
dakota -i dakota_cantilever_centered.in
```

   Does it do what you expect based on the method description in the Reference Manual? Discuss any challenges with your neighbor.

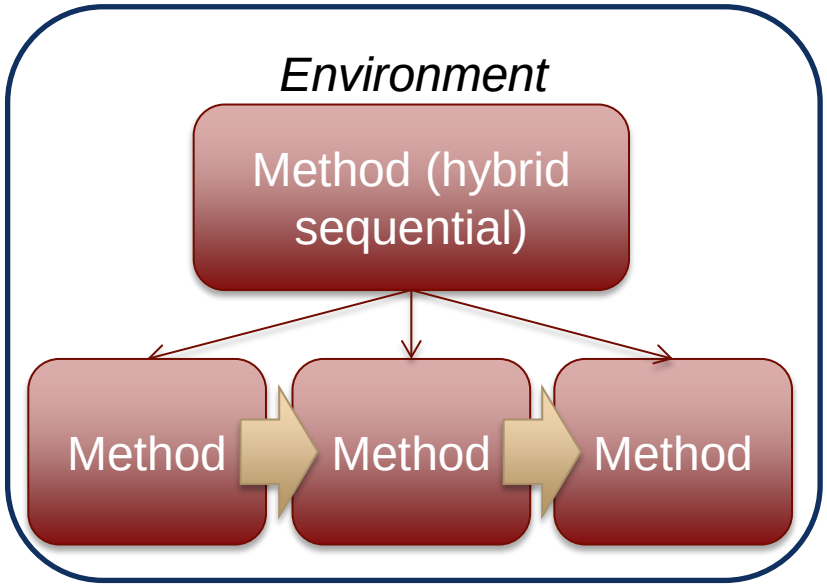

Sandia National Laboratories

Dakota Input and Components

# ADVANCED DAKOTA INPUT (TIME PERMITTING)

# Multiple Methods: Sequential Hybrid Example

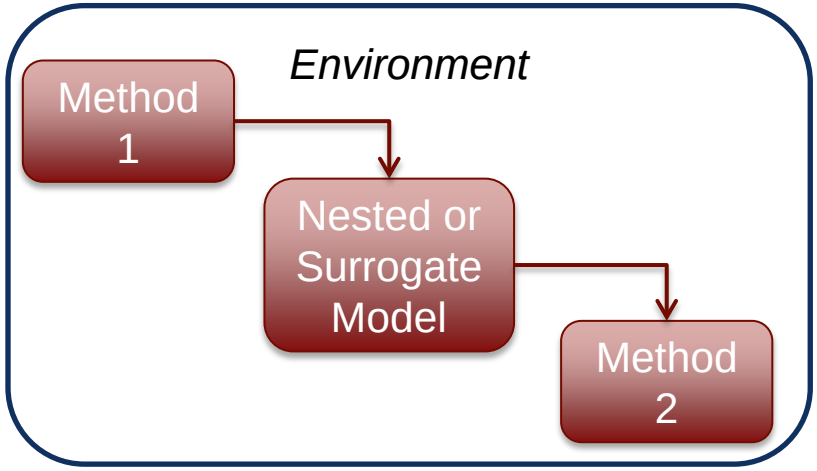
- Multiple methods may be specified, e.g., to apply a sequence of methods to a problem
- Environment block is required in this case to identify top-level method
- Block pointers and IDs are critical to identify execution order




```
environment
  top_method_pointer = 'HS'

method
  id_method = 'HS'
  hybrid sequential
    method_pointer_list = 'GA' 'PS' 'NLP'

method
  id_method = 'GA'
  model_pointer = 'M1'
  coliny_ea

method
  id_method = 'PS'
  model_pointer = 'M1'
  coliny_pattern_search

method
  id_method = 'NLP'
  model_pointer = 'M1'
  optpp_newton

model
  id_model 'M1'
  variables_pointer 'cantilever_vars'
  interface_pointer 'cantilever_inf'
  responses_pointer 'cantilever_resp'
  single
```

# Multiple Models: Surrogate Example

- Multiple models (and multiple methods) may be specified for complex analyses, e.g., surrogate-based approaches
- Block pointers and IDs are critical to identify top-level method and which methods are applied to which models
- More on this in surrogate module




```
environment
  method_pointer = 'METHOD_ON_SURR'

method
  id_method = 'METHOD_ON_SURR'
  sampling
  sample_type lhs
  samples = 100
  seed = 3487
  model_pointer = 'SURR_MODEL'

model
  id_model = 'SURR_MODEL'
  surrogate global
  dace_method_pointer = 'DACE'
  polynomial linear

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling
  samples = 20
  seed = 3492

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'
  variables_pointer = 'V1'
  responses_pointer = 'R1'
```