Make It So Documentation

Release 2.1.3

Peter Schmitt

May 05, 2011

CONTENTS

Release 2.1.3

Date May 03, 2011

CHAPTER

ONE

INTRODUCTION

1.1 Overview

Make It So is a tool that simplifies configuration and execution of numerical models on complex compute environments. It use used to create

- Instructions on how to execute a model
- Shell scripts to run (coupled) model(s)
- Input files for model(s)

1.2 Main Features

- Prompts the user for the minimal set of input parameters to run a model. User input works in several ways:
  - Input from a shell terminal (stdin)
  - From a text configuration file. These configuration files are automatically generated each time makeItSo is run. Look for a file ending with "_makeItSo.config" in your output run directory.
  - Web Interface (Coming soon!)
- Generates model input files for a single model or a collection of coupled models
- Generates shell scripts on a variety of computer systems, taking into account parallel job queues
- Is highly extensible
  - Easily add new models (stand-alone or coupled); see Models/README for details.
  - Easily supports new compute environments & queueing systems; see Environments/README for details.

CHAPTER

TWO

INSTALLING MAKE IT SO

Add "/path/to/LR/misc/python" to your $PATH & $PYTHONPATH:

```
export PATH=/path/to/MakeItSo:$PATH
export PYTHONPATH=/path/to/MakeItSo:$PYTHONPATH
```

CHAPTER

THREE

USING MAKE IT SO

Obtain a short description and list of options via:

```
makeItSo.py --about
makeItSo.py --help
```

To configure a model, execute this command & follow the prompts:

```
makeItSo.py
```

When prompted for a variable, pressing <enter> without typing a value in will use the default value. For example, consider the following prompt:

```
Output directory ( . ):
RUN_IDENTIFIER ( ChangeName ): MyRun
```

Here the user wishes to use the default "." (current directory) for makeItSo output, yet would like to change the RUN_IDENTIFIER from "ChangeName" to "MyRun".

CHAPTER

FOUR

CONFIGURING MAKE IT SO

4.1 Adding a Model

Adding a new model requires four changes to Make It So:

4.1.1 1. Create a YAML file describing the model & its parameters.

At a minimum, the YAML file must contain some metadata and one or more variables/parameters. Below is an example:

```
%YAML 1.1
---
__metadata__:
  model_name: <your_model>
  output_prefix: <inputfile_prefix>
  output_suffix: <inputfile_suffix>

Parameter Type:
  - parameter 1:
      description:  <text_description>
      status: < enabled | disabled | mandatory > (optional)
      type: < boolean | integer | float | string >
      user_input: < required | optional | expert_only | ignore | shared_model_parameter >
      value: <variable value>
      format_regex:  <Regular expression that the value must match.>
      format_human:  <human-readable description of format_regex; used to prompt user>
```

See `MakeItSo/Models/LFM.yml` for an example.

New in LTR-2.2.0: "Status" flag can specify whether a variable is enabled, disabled or required. Each model source file (see step 2, below) must decide what to do with this information.

4.1.2 2. Python source code that specifies a class derived from Model.

You must implement the following functions:

- get_model_setup
- get_iteration_setup
- get_iteration_teardown
- get_run_instructions

- get_input_file
- set_shared_parameters
- set_time_information

See `MakeItSo/Models/LFM.py`, `MakeItSo/Models/MIX.py`, etc. for examples.

New in LTR-2.2.0: "Status" flag can be used by get_input_file in order to generate the proper input file. See TIEGCM.py for an example of the "status" flag.

4.1.3 3. Append your new model to models.yml.

Coupled models are listed first. Individual models to be coupled are nested. For example, with CMIT, you need Shared, LFM, MIX and TIEGCM:

- CMIT: - Shared - LFM - MIX - TIEGCM

4.1.4 4. Add import statement to `Models/__init__.py`

You need to import your new model into MakeItSo by default. See the file at `MakeItSo/Models/__init__.py` for how some models (ie. the LFM) are imported.

4.2 Adding an Environment

To add scripting/queueing environments, you must define all parameters in a `[QUEING-ENVIRONMENT].yml`. The Environment class will locate the YAML file & provide an interface to the user to set up the environment.

We recommend that you start with a pre-existing environment& extend it to your own environment. For example, see `MakeItSo/EnvironmentsLSF.yml` or `MakeItSo/Environments/SGE.yml`.

CHAPTER

FIVE

PROJECT MODULES

5.1 MakeItSo Package

5.1.1 MakeItSo Package

Package that implements makeItSo logic.

5.1.2 Configuration Module

class MakeItSo.Configuration.**Configuration**
Bases: object

Configuration Class deals with makeItSo.config files:

1. Write config file out
2. Read config file & instantiate makeItSo objects.

FIXME: Overload __init__ so it like a function rather than a class e.g. (name, shared, models, env) = Configuration.init(filename)

readConfigurationFile(*filename*)
Read coupled model configuration from file and return properly configured script generator objects

Returns: coupled_name: Name of the coupled model (e.g. 'LFM-MIX') shared: Shared Model object models: A list of Model objects (e.g. [LFM, MIX, XJD] environment: Environment object (e.g. SH, LSF or SGE)

writeConfigurationFile(*filename*, *coupled_name*, *shared*, *models*, *environment*, *includeOptional=False*, *includeExpert=False*)
Write the model configuration to a YAML file.

Parameters: filename: File to write coupled_name: Name of the coupled model (e.g. 'LFM-MIX', 'CMIT', etc.) shared: Shared Model object models: list of Model objects (e.g. for LFM-MIX this might be LFM, MIX, XJD) environment: Compute Environment object includeOptional: Should we write out all optional variables? includeExpert: Should we write out all optional variables?

class MakeItSo.Configuration.**TestConfiguration**(*methodName='runTest'*)
Bases: unittest.TestCase

setUp()

tearDown()

test_readConfigurationFile()

test_write_file_when_parameters_are_changed()

5.1.3 Release Module

Release data for the MakeItSo project $Id: Release.py 1635 2011-05-03 21:13:54Z schmitt $

5.1.4 core Module

MakeItSo.core.**configure_model**(*coupled_model_name*, *run_name*, *output_directory*, *model*, *environment*, *start*, *stop*, *iteration*)

Write the model input file and return the set of commands necessary to run a model.

Parameters: coupled_model_name: Name of coupled model (e.g. CMIT, LFM-MIX, etc.) run_name: Name of the run chosen by the user. model: Model object (e.g. LFM, MIX, TIEGCM, etc.) environment: Environment object (e.g. SH, SGE, etc.) start: start datetime stop: stop datetime iteration: integer iteration number

Returns: model_setup_str: Commands used to prepare the model for execution. model_exec_commands_str: Commands used to actually run a model. model_teardown_str: Commands used to remove temporary/intermediary files used in a model run.

MakeItSo.core.**generateScripts**(*coupled_model_name*, *shared_model*, *models*, *env*, *options*)

Generate the set of scripts & input files given the input parameters.

Parameters: coupled_model_name: Name of coupled model (e.g. CMIT, LFM-MIX, etc.) shared_model: Shared model object models: List of Model objects (e.g. LFM, MIX, TIEGCM, etc.) env: Environment object (e.g. SH, SGE, etc.) options: makeItSo optparse options

MakeItSo.core.**getInputs**(*options*)

Get user input (either stdin or from a config file) necessary to generate scripts.

Parameters: options: makeItSo optparse options

Returns: coupled_model_name: Name of coupled model (e.g. CMIT, LFM-MIX, etc.) shared_model: Shared model object models: List of Model objects (e.g. LFM, MIX, TIEGCM, etc.) env: Environment object (e.g. SH, SGE, etc.)

MakeItSo.core.**main**()

Run makeItSo:

MakeItSo.core.**parse_arguments**()

Returns an optparse object of parsed command-line arguments.

MakeItSo.core.**writeSetupFile**(*coupled_model_name*, *shared_model*, *models*, *env*)

Writes a *RUN_IDENTIFIER-setup.sh* script that preconfigures models. Returns run instructions that should be piped to stdout.

MakeItSo.core.**writeSimulationScripts**(*coupled_model_name*, *shared_model*, *models*, *env*, *spinup_iterations*)

Write the job scripts & input files necessary for the simulation.

Notes:

- Parameters are the same as generateScripts(...)
- If any model spinup is required, those scripts should've been written in writeSpinupScripts(...).

MakeItSo.core.**writeSpinupScripts**(*coupled_model_name*, *shared_model*, *models*, *env*)

Create segments for 'spinup' times, if necessary. These scripts initialize the model & its components.

Notes:

- Parameters are the same as generateScripts(...)
- The shared_model parameter 'spinup' determines if any spinup scripts are necessary.

MakeItSo.core.**write_run_script**(*run_name*, *output_directory*, *environment*, *uses_intercomm*, *setup*, *commands*, *teardown*, *iteration*)

Write coupled model run script to file.

Parameters: run_name: environment: uses_intercomm: setup: commands: teardown: iteration:

Returns: None

5.1.5 share Module

MakeItSo.share.**matches_regex**(*regex*, *inString*)

Does inString match regex?

'Some people, when confronted with a problem, think *I know! I'll use regular expressions.* Now they have two problems.'

– Jamie Zawinski

MakeItSo.share.**setupOutputDirectory**(*dirpath*)

Check if the directory exists, create it if it doesn't. Print warnings if it's not empty.

Parameters: dirpath: directory that we'll generate input files & scripts in.

MakeItSo.share.**stdin_get_variable**(*variable_name*, *variable_dict*)

Query the user (via stdin) to enter data for the given variable. Return the value entered by the user.

Parameters: variable_name: variable_dict: Dictionary that should have the following keys:

- description: <text_description>
- type: < boolean | integer | float | string >
- user_input: < required | optional | expert_only | ignore | shared_model_parameter >
- value: <variable value>
- format_regex: <Python 're' module regular expression that value must match.>
- format_human: <human-readable description of what the variable should contain>

Returns: Value entered by user

MakeItSo.share.**stdin_select**(*name*, *optionList*, *sorted=True*)

Prompt the user via stdin input to select one item from a list.

Parameters: name: Identify what the list represents optionList: List of options to select from sorted: Should we auto-sort the list alphabetically?

Returns: Integer into original selection list

Example:

stdin_select('model', ['CMIT', 'LFM-MIX', 'LFM-RCM', 'LFM', 'TIEGCM']) Available models: 0 CMIT 1 LFM 2 LFM-MIX 3 LFM-RCM 4 TIEGCM Choose a model (0-4) to configure:

MakeItSo.share.**unittestAssertBigStrings**(*testObj*, *s1*, *s2*)

Perform a line-by-line assertEqual on two LARGE strings for a unit test.

Inputs: testObj: an object of a class derived from unittest.TestCase s1: Big string with line breaks to compare against s2. s2: Big string with line breaks to compare against s1.

MakeItSo.share.**word_wrap**(*string*, *width=80*, *ind1=0*, *ind2=0*, *prefix=''*)
word wrapping function. string: the string to wrap width: the column number to wrap at prefix: prefix each line with this string (goes before any indentation) ind1: number of characters to indent the first line ind2: number of characters to indent the rest of the lines

Shamelessly taken from http://www.saltycrane.com/blog/2007/09/python-word-wrap-function/ Could probably adapt from Python's textwrap module http://docs.python.org/library/textwrap.html

5.1.6 Subpackages

Environments Package

Environments Package

To add scripting/queueing environments, you must define all parameters in a [QUEING-ENVIRONMENT].yml. The Environment class will locate the YAML file & provide an interface to the user to set up the environment.

Environment Module

class MakeItSo.Environments.Environment.**Environment**
Bases: object

Set the job scripting environment

get_num_cpus(*cpuString*)
Return the integer number of processors to use.

FIXME: This currently the 'n_processors' string to determines how many CPU's should be used. In the future, I could add a __metadata__ flag like 'nCpusPerNode' and use this to determine how many nodes & cpus should be used

get_pvm_startup()
Return a string containing the code necessary to configure & start the PVM daemon.

get_run_command(*is_mpi*, *executable*, *flags*, *stdin*, *stdout_and_stderr*)
Get the command used to run an executable.

For example usage, see 'test_get_run_instructions'.

Parameters: is_mpi: Boolean to determine whether this particular command needs to be run via MPI. executable: Name of the executable flags: flags for executable stdin: file to stdin redirect stdout_and_stderr: stdout/stderr redirects.

Returns: string: Complete Unix-style run command.

get_script_header(*run_identifier*, *iteration*, *has_InterComm*)
Returns a string containing the header of a job script, including any queueing envrinoment-specific settings.

WARNING: Must input the scripting parameter values first!

get_sh_script_footer(*run_identifier*, *iteration*, *has_InterComm*)
Returns a string containing the footer of a job script.

get_variable_dict(*variable_type*, *variable_name*)
Search the parameters dictionary for *variable_name*, which is of *variable_type*.

This function uses a 'soft copy' of (akin to a C pointer to) the variable, and hence it's the proper way to get and set variables.

selectEnvironment(*selection*)
Select a particular environment & set helper member variables internal to the Environment class.

class MakeItSo.Environments.Environment.**TestEnvironment**(*methodName='runTest'*)
Bases: unittest.TestCase

setUp()

test_get_num_cpus()

test_get_num_cpus_TRM()

Torque Resource Manager (TRM) requires a different format for nCpus: -lnodes=3:ppn:8

This will utilize 3 nodes and 8 CPUs per node, so 24 CPUs total.

test_get_run_command()

inputs Module

Functions that help get user input to Environment parameters.

MakeItSo.Environments.inputs.**input_parameter_values_from_stdin**(*env*, *include_optional*, *include_expert*, *has_InterComm=True*)
Query the user to select an environment & then enter all required input parameters via stdin input

Parameters: env: Environment object to get/set variables include_optional: When true, query user to input optional variables include_expert: When true, query user to input expert variables has_InterComm: When true, the user needs to input InterComm & PVM variables

MakeItSo.Environments.inputs.**stdin**(*include_optional*, *include_expert*, *has_InterComm=True*)
Select a run environment & input its parameters

Parameters: include_optional: When true, query user to input optional variables include_expert: When true, query user to input expert variables has_InterComm: When true, the user needs to input InterComm & PVM variables

Returns: env - Environment object (i.e. SH, SGE, LSF, etc.)

Models Package

Models Package

Coupled and stand-alone models must be defined in 'models.yml'.

To add additional models, you must:

- Define it in 'models.yml'
- Create [MODEL-NAME].yml defining parameters
- Create [MODEL-NAME].py, including a [MODEL-NAME] class derived from Model. See LFM.py for an example.
- Add import statement to Models/__init__.py

See README for more details.

MakeItSo.Models.**getModelCoupling**()
Get dictionary of queueing environments

LFM Module

class MakeItSo.Models.LFM.**LFM**
Bases: MakeItSo.Models.Model.Model

LFM Model Type

get_input_file()
Get contents of LFM INPUT1.xml file.

Returns: str_script: String of job script

FIXME: If no module variables are added, the XML still gets an empty <module>...</module> tag...

Note: Required function (standardized parameters & return variables)

get_iteration_setup(*model_name*, *iteration*)
Setup environment to run a model, ends with newline when not empty

get_iteration_teardown(*model_name*, *iteration*)
Teardown environment to run a model, ends with newline when not empty

Note: this is /bin/sh specific!

get_model_setup(*model_name*, *has_spinup*, *run_identifier*, *start_datetime*)

get_run_instructions(*model_name*, *iteration*)
Get the executable, inputs & outputs needed to run the model.

For example usage, see 'test_get_run_instructions'.

Note: Required function (standardized parameters & return variables)

Returns: boolean: True when parallel executable: Name of executable to run flags: any command-line arguments stdin: where to redirect stdin input from stdout_and_stderr: Where to redirect stdout/stderr to?

get_spinup_coupled_model_name(*coupled_model_name*)

set_shared_model_parameters(*run_prefix*, *lfm_mix_exchange_frequency*, *dumpInterval*, *use_intercomm*)
Setup internal structure of generic LFM shared parameters

This sets four parameters defined in LFM.yml:

- HDFBASE_IN, HDFBASE_OUT, RUN_NAME: strings used for naming LFM input/output files
- TIME_DUMP_INTERVAL: Number of seconds between file dumps
- USE_INTERCOMM: do we use InterComm (True) or adhoc (False) coupling?

set_shared_parameters(*coupled_model_name*, *shared_model*, *env=None*)
Note: Required function (standardized parameters & return variables)

set_time_information(*start_datetime*, *stop_datetime*)
Setup the internal structure of the LFM parameters to generate the correct set of scripts.

This function sets two parameters defined in LFM.yml:

- **START_DATETIME: Whitespace delimited integer array of YYYY** MM DD HH MM SS for the first timestep on the current segment.
- **STOP_DATETIME: Whitespace delimited integer array of YYYY** MM DD HH MM SS for the final timestep on the current segment.

start_datetime: datetime object stop_datetime: datetime object

Note: Required function (standardized parameters & return variables)

class MakeItSo.Models.LFM.**TestLFM**(*methodName='runTest'*)
Bases: unittest.TestCase

setUp()

test__resolution_string()

test_get_iteration_setup()

test_get_iteration_teardown()

test_get_model_setup()

test_get_run_instructions()

test_get_spinup_coupled_model_name()

test_set_shared_model_paraeters()

test_set_time_information()

MIX Module

class MakeItSo.Models.MIX.**MIX**
Bases: MakeItSo.Models.Model.Model

MIX Model Type

get_input_file()
Get contents of MIX parameter file

Returns: str_script: String of job script

Note: Required function (standardized parameters & return variables)

get_iteration_setup(*model_name*, *iteration*)
Setup environment to run a model, ends with newline when not empty

get_iteration_teardown(*model_name*, *iteration*)
Teardown environment to run a model, ends with newline when not empty

get_model_setup(*model_name*, *has_spinup*, *run_identifier*, *start_datetime*)

get_run_instructions(*model_name*, *iteration*)
Get the executable, inputs & outputs needed to run the model.

For example usage, see 'test_get_run_instructions'.

Note: Required function (standardized parameters & return variables)

set_shared_model_parameters(*run_prefix*, *exchange_frequency*, *dump_interval*, *use_intercomm*)
Setup internal structure of generic MIX shared parameters

This sets two parameters defined in MIX.yml:

- OUTPUT_FILE_PREFIX: string to prepend to each MIX output file.
- FLUSH_FREQUENCY: Number of LFM-MIX exchanges between file dumps (set to match the 'dumpInterval').
- USE_INTERCOMM: Is coupling Adhoc or via InterComm?

Parameters: run_prefix: string to prepend to LFM output files. exchange_frequency: Number of seconds between exchanges with LFM dump_interval: Number of seconds between file dumps use_intercomm: do we use InterComm (True) or adhoc (False) coupling?

set_shared_parameters(*coupled_model_name*, *shared_model*, *env=None*)
Note: Required function (standardized parameters & return variables)

set_time_information(*start_datetime*, *stop_datetime*)
Set start time (i.e. to point to the correct MHD_GRID_FILE_NAME).

This sets one parameters defined in MIX.yml:

- MHD_GRID_FILE_NAME: to something like 'CMIT_mhd_YYYY-MM-DDTHH-MM-SSZ.hdf'

Parameters:

start_datetime: datetime object specifying the starting year-month-day-hour-minute-second for this segment.

stop_datetime: Unused. Stop time determined by LFM kill signal.

Note: Required function (standardized parameters & return variables)

class MakeItSo.Models.MIX.**TestMIX**(*methodName='runTest'*)
Bases: unittest.TestCase

setUp()

test_get_iteration_setup()

test_get_iteration_teardown()

test_get_model_setup()

test_get_run_instructions()

test_set_shared_model_parameters()

test_set_shared_parameters()

test_set_time_information()

Model Module

Individual model parameters for LFM, MIX, TIEGCM, and any variables shared between the models.

class MakeItSo.Models.Model.**Model**(*filename*)
Bases: object

Abstract base class for code models

get_input_file()

get_iteration_setup(*model_name*, *iteration*)

get_iteration_teardown(*model_name*, *iteration*)

get_model_setup(*model_name*, *has_spinup*, *run_identifier*, *start_datetime*)

get_run_instructions(*model_name*, *iteration*)

get_spinup_coupled_models()

get_spinup_times(*start*)
Get segments required for model spinup (ending at 'start' datetime).

Parameters: start: datetime used for starting a TIE-GCM run

Returns: spinup_start: list of datetime objects corresponding to start date & time for model spinup segments. spinup_stop: list of datetime objects corresponding to stop date & time for model spinup segments. spinup_dt: List of float objects corresponding to the time dump interval for the spinup

FIXME: There's a peculiar error here: Given certain CMIT inputs:

- •START_YMDHMS: 1996 5 19 16 0 0
- •STOP_YMDHMS: 1996 5 20 6 0 0
- •TIME_DUMP_INTERVAL: 120.0

this function returns a spinup_dt where at least one element of the list are datetime.timedelta objects rather than a floating-point number of seconds. I suspect this happens because it jumps from returning number of seconds to either number of minutes or hours.

get_variable_dict (*variable_type*, *variable_name*)
Search the parameters dictionary for *variable_name*, which is of *variable_type*.

This function uses a 'soft copy' of (akin to a C pointer to) the variable, and hence it's the proper way to get and set variables.

set_shared_parameters (*shared_model*, *env=None*)

set_time_information (*start_datetime*, *stop_datetime*)

NewModelTemplate Module

RCM Module

class MakeItSo.Models.RCM.**RCM**
Bases: MakeItSo.Models.Model.Model

RCM Model Type

get_input_file ()
Get contents of [] input parameter file

Returns: str_script: String of job script

Note: Required function (standardized parameters & return variables)

get_iteration_setup (*model_name*, *iteration*)
Setup environment to run a model, ends with newline when not empty

RCM requires a few files:

- Coupling parameters: rcm_lfm.param
- **RCMfiles subdirectory containing:**
  - rcm.params namelist
  - enchan.dat (defines constants)
  - dktable (defines constants)

get_iteration_teardown (*model_name*, *iteration*)
Teardown environment to run a model, ends with newline when not empty

get_model_setup (*model_name*, *has_spinup*, *run_identifier*, *start_datetime*)

get_run_instructions(*model_name*, *iteration*)
Get the executable, inputs & outputs needed to run the model.

For example usage, see 'test_get_run_instructions'.

Note: Required function (standardized parameters & return variables)

Returns: boolean: True when parallel executable: Name of executable to run flags: any command-line arguments stdin: where to redirect stdin input from stdout_and_stderr: Where to redirect stdout/stderr to?

set_shared_parameters(*coupled_model_name*, *shared_model*, *env=None*)
Note: Required function (standardized parameters & return variables)

set_time_information(*start_datetime*, *stop_datetime*)
Take care of any special behavior for start/stop times for a run.

Parameters:

start_datetime: datetime object specifying the starting year-month-day-hour-minute-second for this segment.

stop_datetime: Unused. Stop time determined by LFM kill signal.

Note: Required function (standardized parameters & return variables)

class MakeItSo.Models.RCM.**TestRCM**(*methodName='runTest'*)
Bases: unittest.TestCase

setUp()

test_get_iteration_setup()

test_get_iteration_teardown()

test_get_model_setup()

test_get_run_instructions()

test_set_shared_parameters()

Shared Module

class MakeItSo.Models.Shared.**Shared**
Bases: MakeItSo.Models.Model.Model

Shared model parameters

get_segment_time_slices()
Calculate the segment start and stop times via the user input variables defined in Models/Shared.yml

The user should input reasonable values for the following variables before calling this function:

•START_YMDHMS

•STOP_YMDHMS

•TIME_DUMP_INTERVAL

•WALLCOCK_HOURS_PER_SEGMENT

•SIMULATION_HOURS_PER_SEGMENT

Returns:

- segment_start_times: array of datetime objects
- segment_stop_times: array of datetime objects

is_Coupled_Via_InterComm(*coupled_model_name*)
Check if USE_INTERCOMM is enabled if coupled_model_name is a coupled model that is capable of utilizing InterComm.

Parameters: coupled_model_name: name of coupled model (e.g. LFM-MIX, CMIT, etc.)

class MakeItSo.Models.Shared.**TestShared**(*methodName='runTest'*)
Bases: unittest.TestCase

setUp()

test_get_segment_time_slices()

test_get_segment_time_slices_fast_cpu()

test_get_segment_time_slices_slow_cpu()

test_is_Coupled_Via_InterComm()

TIEGCM Module

class MakeItSo.Models.TIEGCM.**TIEGCM**
Bases: MakeItSo.Models.Model.Model

TIEGCM Model Type

get_input_file()
Get contents of TIEGCM input namelist

Returns: str_script: String of job script

Note: Required function (standardized parameters & return variables)

get_iteration_setup(*model_name*, *iteration*)
Setup environment to run a model, ends with newline when not empty

get_iteration_teardown(*model_name*, *iteration*)
Teardown environment to run a model, ends with newline when not empty

get_model_setup(*model_name*, *has_spinup*, *run_identifier*, *start_datetime*)
FIXME: In the future, provide support for 2.5 or 5 degree resolution. FIXME: In the future, allow for MPI or serial tiegcm.

get_run_instructions(*model_name*, *iteration*)
Get the executable, inputs & outputs needed to run the model.

For example usage, see 'test_get_run_instructions'.

Note: Required function (standardized parameters & return variables)

get_spinup_coupled_model_name(*coupled_model_name*)

get_spinup_times(*start*)
Overloaded Model::get_spinup_times(...) that ensures the initial starting time is 00:00:00 for any given Year/Month/Day.

Parameters: start: datetime used for starting a TIE-GCM run

Returns: spinup_start: list of datetime objects corresponding to start date & time for model spinup segments. spinup_stop: list of datetime objects corresponding to stop date & time for model spinup segments.

set_shared_model_parameters (*label*, *dump_interval*)
Setup internal structure of generic TIEGCM shared parameters

This sets some shared parameters defined in TIEGCM.yml:

- LABEL
- HIST: Frequency to save primary history files
- SECHIST: Frequency to save secondary history files

Due to a 'feature' (or is it a bug?) with TIEGCM, we must also set the following to match HIST & SECHIST:

•SAVE

•SECSAVE

Function Parameters: label: string dump_interval: floating-point number of seconds between dumps

set_shared_parameters (*coupled_model_name*, *shared_model*, *env=None*)
Note: Required function (standardized parameters & return variables)

set_time_information (*start*, *stop*)

This function sets the following TIEGCM parameters based on function input parameters.

- START_YEAR (4-digit integer)
- START_DAY (day of year)
- SOURCE (Filename: prefix_TIE_YYYY-MM-DDTHH.nc)
- SOURCE_START (DayOfYear,Hour,Minute)
- START (DayOfYear,Hour,Minute)
- STOP (DayOfYear,Hour,Minute)
- OUTPUT: list of silenames ('prefix_TIE_YYYY-MM-DDThh.nc','prefix_TIE_YYYY-MM-DDThh.nc',...)
- MXHIST_PRIM (integer)
- SECSTART (DayOfYear,Hour,Minute)
- SECSTOP (DayOfYear,Hour,Minute)
- SECOUT list of silenames ('prefix_sech_TIE_YYYY-MM-DDThh.nc','prefix_sech_TIE_YYYY-MM-DDThh.nc',...)
- MXHIST_SECH (integer)

Note: This function contains a considerable section of business logic to determine the primary & secondary history file names.

Function input parameters: start: segment start time stop: segment stop time

Note: Required function (standardized parameters & return variables)

class MakeItSo.Models.TIEGCM.**TestTIEGCM** (*methodName='runTest'*)
Bases: unittest.TestCase

setUp()

test__get_filename_string()

test__seconds_to_days_hours_minutes()

test_get_input_file()
Make sure the default input file matches the expected input

test_get_iteration_setup()

test_get_model_setup()

test_get_run_instructions()

test_get_spinup_coupled_model_name()

test_get_spinup_times()

test_set_time_information()

test_shared_model_parameters()

XJD Module

class MakeItSo.Models.XJD.**TestXJD**(*methodName='runTest'*)
Bases: unittest.TestCase

setUp()

test_get_model_setup()

test_set_shared_parameters()

class MakeItSo.Models.XJD.**XJD**
Bases: MakeItSo.Models.Model.Model

InterComm XML Job Description file

Since we only need one or two XJD files for a whole run interval, most functions do nothing.

get_input_file()

get_iteration_setup(*model_name*, *iteration*)
Setup environment to run a model, ends with newline when not empty

get_iteration_teardown(*model_name*, *iteration*)
Teardown environment to run a model, ends with newline when not empty

get_model_setup(*model_name*, *has_spinup*, *run_identifier*, *start_datetime*)

get_run_instructions(*model_name*, *iteration*)

get_xjd_file(*components*)

set_shared_parameters(*coupled_model_name*, *shared_model*, *env=None*)
Note: Required function (standardized parameters & return variables)

set_time_information(*start_datetime*, *stop_datetime*)

write_xjd_file(*filename*, *components*)
Write an XJD file with a particular set of components

Parameters: filename: write XJD file to here components: list of components usef for coupled models

Example: write_xjd_file('LFM-MIX-RCM.xjd', ['mhd','cpl','im'])

inputs Module

Functions that help get user input to model parameters.

MakeItSo.Models.inputs.**stdin**(*include_optional=False*, *include_expert=False*)
Select a model & input its parameters.

Models & their dependencies are defined in 'models.yml'

Parameters: include_optional: Query user to input optional variables? include_expert: Query user to input expert_only variables?

Returns: coupled_model_name - name of the coupled model to configure. models - list of model parameters shared_model_index - integer index into 'models' list of shared model parameters

MakeItSo.Models.inputs.**stdin_input_parameters**(*coupled_model_name*, *model_object*, *include_optional=False*, *include_expert=False*)
Query the user to enter all required input parameters via stdin input. Note the model_object is an input & output parameter, since it reads & writes member data.

Parameters:

coupled_model_name: Name of the coupled model (CMIT, LFM-MIX, etc.)

model_object: An object derived from a subclass of Model (e.g. LFM, MIX, etc.)

include_optional: Should we prompt the user for optional variables?

include_expert: Should we prompt the user for expert_only variables?

Returns: None: Has no return, but this function modifies the 'model_object' object.

yaml Package

yaml Package

class MakeItSo.yaml.**YAMLObject**
Bases: object

An object that can dump itself to a YAML stream and load itself from a YAML stream.

classmethod from_yaml(*loader*, *node*)
Convert a representation node to a Python object.

classmethod to_yaml(*dumper*, *data*)
Convert a Python object to a representation node.

yaml_dumper
alias of Dumper

yaml_loader
alias of Loader

class MakeItSo.yaml.**YAMLObjectMetaclass**(*name*, *bases*, *kwds*)
Bases: type

The metaclass for YAMLObject.

MakeItSo.yaml.**add_constructor**(*tag*, *constructor*, *Loader=<class 'MakeItSo.yaml.loader.Loader'>*)

Add a constructor for the given tag. Constructor is a function that accepts a Loader instance and a node object and produces the corresponding Python object.

MakeItSo.yaml.**add_implicit_resolver**(*tag*, *regexp*, *first=None*, *Loader=<class 'MakeItSo.yaml.loader.Loader'>*, *Dumper=<class 'MakeItSo.yaml.dumper.Dumper'>*)

Add an implicit scalar detector. If an implicit scalar value matches the given regexp, the corresponding tag is assigned to the scalar. first is a sequence of possible initial characters or None.

MakeItSo.yaml.**add_multi_constructor**(*tag_prefix*, *multi_constructor*, *Loader=<class 'MakeItSo.yaml.loader.Loader'>*)

Add a multi-constructor for the given tag prefix. Multi-constructor is called for a node if its tag starts with tag_prefix. Multi-constructor accepts a Loader instance, a tag suffix, and a node object and produces the corresponding Python object.

MakeItSo.yaml.**add_multi_representer**(*data_type*, *multi_representer*, *Dumper=<class 'MakeItSo.yaml.dumper.Dumper'>*)

Add a representer for the given type. Multi-representer is a function accepting a Dumper instance and an instance of the given data type or subtype and producing the corresponding representation node.

MakeItSo.yaml.**add_path_resolver**(*tag*, *path*, *kind=None*, *Loader=<class 'MakeItSo.yaml.loader.Loader'>*, *Dumper=<class 'MakeItSo.yaml.dumper.Dumper'>*)

Add a path based resolver for the given tag. A path is a list of keys that forms a path to a node in the representation tree. Keys can be string values, integers, or None.

MakeItSo.yaml.**add_representer**(*data_type*, *representer*, *Dumper=<class 'MakeItSo.yaml.dumper.Dumper'>*)

Add a representer for the given type. Representer is a function accepting a Dumper instance and an instance of the given data type and producing the corresponding representation node.

MakeItSo.yaml.**compose**(*stream*, *Loader=<class 'MakeItSo.yaml.loader.Loader'>*)

Parse the first YAML document in a stream and produce the corresponding representation tree.

MakeItSo.yaml.**compose_all**(*stream*, *Loader=<class 'MakeItSo.yaml.loader.Loader'>*)

Parse all YAML documents in a stream and produce corresponsing representation trees.

MakeItSo.yaml.**dump**(*data*, *stream=None*, *Dumper=<class 'MakeItSo.yaml.dumper.Dumper'>*, ***kwds*)

Serialize a Python object into a YAML stream. If stream is None, return the produced string instead.

MakeItSo.yaml.**dump_all**(*documents*, *stream=None*, *Dumper=<class 'MakeItSo.yaml.dumper.Dumper'>*, *default_style=None*, *default_flow_style=None*, *canonical=None*, *indent=None*, *width=None*, *allow_unicode=None*, *line_break=None*, *encoding='utf-8'*, *explicit_start=None*, *explicit_end=None*, *version=None*, *tags=None*)

Serialize a sequence of Python objects into a YAML stream. If stream is None, return the produced string instead.

MakeItSo.yaml.**emit**(*events*, *stream=None*, *Dumper=<class 'MakeItSo.yaml.dumper.Dumper'>*, *canonical=None*, *indent=None*, *width=None*, *allow_unicode=None*, *line_break=None*)

Emit YAML parsing events into a stream. If stream is None, return the produced string instead.

MakeItSo.yaml.**load**(*stream*, *Loader=<class 'MakeItSo.yaml.loader.Loader'>*)

Parse the first YAML document in a stream and produce the corresponding Python object.

MakeItSo.yaml.**load_all**(*stream*, *Loader=<class 'MakeItSo.yaml.loader.Loader'>*)

Parse all YAML documents in a stream and produce corresponding Python objects.

MakeItSo.yaml.**parse**(*stream*, *Loader=<class 'MakeItSo.yaml.loader.Loader'>*)
Parse a YAML stream and produce parsing events.

MakeItSo.yaml.**safe_dump**(*data*, *stream=None*, ***kwds*)
Serialize a Python object into a YAML stream. Produce only basic YAML tags. If stream is None, return the produced string instead.

MakeItSo.yaml.**safe_dump_all**(*documents*, *stream=None*, ***kwds*)
Serialize a sequence of Python objects into a YAML stream. Produce only basic YAML tags. If stream is None, return the produced string instead.

MakeItSo.yaml.**safe_load**(*stream*)
Parse the first YAML document in a stream and produce the corresponding Python object. Resolve only basic YAML tags.

MakeItSo.yaml.**safe_load_all**(*stream*)
Parse all YAML documents in a stream and produce corresponding Python objects. Resolve only basic YAML tags.

MakeItSo.yaml.**scan**(*stream*, *Loader=<class 'MakeItSo.yaml.loader.Loader'>*)
Scan a YAML stream and produce scanning tokens.

MakeItSo.yaml.**serialize**(*node*, *stream=None*, *Dumper=<class 'MakeItSo.yaml.dumper.Dumper'>*, ***kwds*)
Serialize a representation tree into a YAML stream. If stream is None, return the produced string instead.

MakeItSo.yaml.**serialize_all**(*nodes*, *stream=None*, *Dumper=<class 'MakeItSo.yaml.dumper.Dumper'>*, *canonical=None*, *indent=None*, *width=None*, *allow_unicode=None*, *line_break=None*, *encoding='utf-8'*, *explicit_start=None*, *explicit_end=None*, *version=None*, *tags=None*)
Serialize a sequence of representation trees into a YAML stream. If stream is None, return the produced string instead.

composer Module

class MakeItSo.yaml.composer.**Composer**
Bases: object

check_node()

compose_document()

compose_mapping_node(*anchor*)

compose_node(*parent*, *index*)

compose_scalar_node(*anchor*)

compose_sequence_node(*anchor*)

get_node()

get_single_node()

exception MakeItSo.yaml.composer.**ComposerError**(*context=None*, *context_mark=None*, *problem=None*, *problem_mark=None*, *note=None*)
Bases: MakeItSo.yaml.error.MarkedYAMLError

constructor Module

class MakeItSo.yaml.constructor.**BaseConstructor**
Bases: object

classmethod add_constructor(*tag*, *constructor*)

classmethod add_multi_constructor(*tag_prefix*, *multi_constructor*)

check_data()

construct_document(*node*)

construct_mapping(*node*, *deep=False*)

construct_object(*node*, *deep=False*)

construct_pairs(*node*, *deep=False*)

construct_scalar(*node*)

construct_sequence(*node*, *deep=False*)

get_data()

get_single_data()

class MakeItSo.yaml.constructor.**SafeConstructor**
Bases: MakeItSo.yaml.constructor.BaseConstructor

construct_mapping(*node*, *deep=False*)

construct_scalar(*node*)

construct_undefined(*node*)

construct_yaml_binary(*node*)

construct_yaml_bool(*node*)

construct_yaml_float(*node*)

construct_yaml_int(*node*)

construct_yaml_map(*node*)

construct_yaml_null(*node*)

construct_yaml_object(*node*, *cls*)

construct_yaml_omap(*node*)

construct_yaml_pairs(*node*)

construct_yaml_seq(*node*)

construct_yaml_set(*node*)

construct_yaml_str(*node*)

construct_yaml_timestamp(*node*)

flatten_mapping(*node*)

class MakeItSo.yaml.constructor.**Constructor**
Bases: MakeItSo.yaml.constructor.SafeConstructor

class classobj

Constructor.**construct_python_complex**(*node*)

Constructor.**construct_python_long**(*node*)

Constructor.**construct_python_module**(*suffix*, *node*)

Constructor.**construct_python_name**(*suffix*, *node*)

Constructor.**construct_python_object**(*suffix*, *node*)

Constructor.**construct_python_object_apply**(*suffix*, *node*, *newobj=False*)

Constructor.**construct_python_object_new**(*suffix*, *node*)

Constructor.**construct_python_str**(*node*)

Constructor.**construct_python_tuple**(*node*)

Constructor.**construct_python_unicode**(*node*)

Constructor.**find_python_module**(*name*, *mark*)

Constructor.**find_python_name**(*name*, *mark*)

Constructor.**make_python_instance**(*suffix*, *node*, *args=None*, *kwds=None*, *newobj=False*)

Constructor.**set_python_instance_state**(*instance*, *state*)

exception MakeItSo.yaml.constructor.**ConstructorError**(*context=None*, *context_mark=None*, *problem=None*, *problem_mark=None*, *note=None*)

Bases: MakeItSo.yaml.error.MarkedYAMLError

cyaml Module

dumper Module

class MakeItSo.yaml.dumper.**BaseDumper**(*stream*, *default_style=None*, *default_flow_style=None*, *canonical=None*, *indent=None*, *width=None*, *allow_unicode=None*, *line_break=None*, *encoding=None*, *explicit_start=None*, *explicit_end=None*, *version=None*, *tags=None*)

Bases: MakeItSo.yaml.emitter.Emitter, MakeItSo.yaml.serializer.Serializer, MakeItSo.yaml.representer.BaseRepresenter, MakeItSo.yaml.resolver.BaseResolver

class MakeItSo.yaml.dumper.**SafeDumper**(*stream*, *default_style=None*, *default_flow_style=None*, *canonical=None*, *indent=None*, *width=None*, *allow_unicode=None*, *line_break=None*, *encoding=None*, *explicit_start=None*, *explicit_end=None*, *version=None*, *tags=None*)

Bases: MakeItSo.yaml.emitter.Emitter, MakeItSo.yaml.serializer.Serializer, MakeItSo.yaml.representer.SafeRepresenter, MakeItSo.yaml.resolver.Resolver

class MakeItSo.yaml.dumper.**Dumper**(*stream*, *default_style=None*, *default_flow_style=None*, *canonical=None*, *indent=None*, *width=None*, *allow_unicode=None*, *line_break=None*, *encoding=None*, *explicit_start=None*, *explicit_end=None*, *version=None*, *tags=None*)

Bases: MakeItSo.yaml.emitter.Emitter, MakeItSo.yaml.serializer.Serializer, MakeItSo.yaml.representer.Representer, MakeItSo.yaml.resolver.Resolver

emitter Module

class MakeItSo.yaml.emitter.**Emitter**(*stream*, *canonical=None*, *indent=None*, *width=None*, *allow_unicode=None*, *line_break=None*)

Bases: object

analyze_scalar(*scalar*)

check_empty_document()

check_empty_mapping()

check_empty_sequence()

check_simple_key()

choose_scalar_style()

determine_block_hints(*text*)

emit(*event*)

expect_alias()

expect_block_mapping()

expect_block_mapping_key(*first=False*)

expect_block_mapping_simple_value()

expect_block_mapping_value()

expect_block_sequence()

expect_block_sequence_item(*first=False*)

expect_document_end()

expect_document_root()

expect_document_start(*first=False*)

expect_first_block_mapping_key()

expect_first_block_sequence_item()

expect_first_document_start()

expect_first_flow_mapping_key()

expect_first_flow_sequence_item()

expect_flow_mapping()

expect_flow_mapping_key()

expect_flow_mapping_simple_value()

expect_flow_mapping_value()

expect_flow_sequence()

expect_flow_sequence_item()

expect_node(*root=False*, *sequence=False*, *mapping=False*, *simple_key=False*)

expect_nothing()

expect_scalar()

expect_stream_start()

flush_stream()

increase_indent(*flow=False*, *indentless=False*)

need_events(*count*)

need_more_events()

prepare_anchor(*anchor*)

prepare_tag(*tag*)

prepare_tag_handle(*handle*)

prepare_tag_prefix(*prefix*)

prepare_version(*version*)

process_anchor(*indicator*)

process_scalar()

process_tag()

write_double_quoted(*text*, *split=True*)

write_folded(*text*)

write_indent()

write_indicator(*indicator*, *need_whitespace*, *whitespace=False*, *indention=False*)

write_line_break(*data=None*)

write_literal(*text*)

write_plain(*text*, *split=True*)

write_single_quoted(*text*, *split=True*)

write_stream_end()

write_stream_start()

write_tag_directive(*handle_text*, *prefix_text*)

write_version_directive(*version_text*)

exception MakeItSo.yaml.emitter.**EmitterError**
Bases: MakeItSo.yaml.error.YAMLError

error Module

class MakeItSo.yaml.error.**Mark**(*name*, *index*, *line*, *column*, *buffer*, *pointer*)
Bases: object

get_snippet(*indent=4*, *max_length=75*)

exception MakeItSo.yaml.error.**YAMLError**
Bases: exceptions.Exception

exception MakeItSo.yaml.error.**MarkedYAMLError**(*context=None*, *context_mark=None*, *problem=None*, *problem_mark=None*, *note=None*)
Bases: MakeItSo.yaml.error.YAMLError

events Module

class MakeItSo.yaml.events.**AliasEvent**(*anchor*, *start_mark=None*, *end_mark=None*)
Bases: MakeItSo.yaml.events.NodeEvent

class MakeItSo.yaml.events.**CollectionEndEvent**(*start_mark=None*, *end_mark=None*)
Bases: MakeItSo.yaml.events.Event

class MakeItSo.yaml.events.**CollectionStartEvent**(*anchor*, *tag*, *implicit*, *start_mark=None*, *end_mark=None*, *flow_style=None*)
Bases: MakeItSo.yaml.events.NodeEvent

class MakeItSo.yaml.events.**DocumentEndEvent**(*start_mark=None*, *end_mark=None*, *explicit=None*)
Bases: MakeItSo.yaml.events.Event

class MakeItSo.yaml.events.**DocumentStartEvent**(*start_mark=None*, *end_mark=None*, *explicit=None*, *version=None*, *tags=None*)
Bases: MakeItSo.yaml.events.Event

class MakeItSo.yaml.events.**Event**(*start_mark=None*, *end_mark=None*)
Bases: object

class MakeItSo.yaml.events.**MappingEndEvent**(*start_mark=None*, *end_mark=None*)
Bases: MakeItSo.yaml.events.CollectionEndEvent

class MakeItSo.yaml.events.**MappingStartEvent**(*anchor*, *tag*, *implicit*, *start_mark=None*, *end_mark=None*, *flow_style=None*)
Bases: MakeItSo.yaml.events.CollectionStartEvent

class MakeItSo.yaml.events.**NodeEvent**(*anchor*, *start_mark=None*, *end_mark=None*)
Bases: MakeItSo.yaml.events.Event

class MakeItSo.yaml.events.**ScalarEvent**(*anchor*, *tag*, *implicit*, *value*, *start_mark=None*, *end_mark=None*, *style=None*)
Bases: MakeItSo.yaml.events.NodeEvent

class MakeItSo.yaml.events.**SequenceEndEvent**(*start_mark=None*, *end_mark=None*)
Bases: MakeItSo.yaml.events.CollectionEndEvent

class MakeItSo.yaml.events.**SequenceStartEvent**(*anchor*, *tag*, *implicit*, *start_mark=None*, *end_mark=None*, *flow_style=None*)
Bases: MakeItSo.yaml.events.CollectionStartEvent

class MakeItSo.yaml.events.**StreamEndEvent**(*start_mark=None*, *end_mark=None*)
Bases: MakeItSo.yaml.events.Event

class MakeItSo.yaml.events.**StreamStartEvent**(*start_mark=None*, *end_mark=None*, *encoding=None*)
Bases: MakeItSo.yaml.events.Event

loader Module

class MakeItSo.yaml.loader.**BaseLoader**(*stream*)
Bases: MakeItSo.yaml.reader.Reader, MakeItSo.yaml.scanner.Scanner, MakeItSo.yaml.parser.Parser, MakeItSo.yaml.composer.Composer, MakeItSo.yaml.constructor.BaseConstructor, MakeItSo.yaml.resolver.BaseResolver

class MakeItSo.yaml.loader.**SafeLoader**(*stream*)
Bases: MakeItSo.yaml.reader.Reader, MakeItSo.yaml.scanner.Scanner,

MakeItSo.yaml.parser.Parser, MakeItSo.yaml.composer.Composer, MakeItSo.yaml.constructor.SafeConstructor, MakeItSo.yaml.resolver.Resolver

class MakeItSo.yaml.loader.**Loader**(*stream*)
Bases: MakeItSo.yaml.reader.Reader, MakeItSo.yaml.scanner.Scanner, MakeItSo.yaml.parser.Parser, MakeItSo.yaml.composer.Composer, MakeItSo.yaml.constructor.Constructor, MakeItSo.yaml.resolver.Resolver

nodes Module

class MakeItSo.yaml.nodes.**CollectionNode**(*tag*, *value*, *start_mark=None*, *end_mark=None*, *flow_style=None*)
Bases: MakeItSo.yaml.nodes.Node

class MakeItSo.yaml.nodes.**MappingNode**(*tag*, *value*, *start_mark=None*, *end_mark=None*, *flow_style=None*)
Bases: MakeItSo.yaml.nodes.CollectionNode

class MakeItSo.yaml.nodes.**Node**(*tag*, *value*, *start_mark*, *end_mark*)
Bases: object

class MakeItSo.yaml.nodes.**ScalarNode**(*tag*, *value*, *start_mark=None*, *end_mark=None*, *style=None*)
Bases: MakeItSo.yaml.nodes.Node

class MakeItSo.yaml.nodes.**SequenceNode**(*tag*, *value*, *start_mark=None*, *end_mark=None*, *flow_style=None*)
Bases: MakeItSo.yaml.nodes.CollectionNode

parser Module

class MakeItSo.yaml.parser.**Parser**
Bases: object

check_event(**choices*)

get_event()

parse_block_mapping_first_key()

parse_block_mapping_key()

parse_block_mapping_value()

parse_block_node()

parse_block_node_or_indentless_sequence()

parse_block_sequence_entry()

parse_block_sequence_first_entry()

parse_document_content()

parse_document_end()

parse_document_start()

parse_flow_mapping_empty_value()

parse_flow_mapping_first_key()

parse_flow_mapping_key(*first=False*)

parse_flow_mapping_value()

parse_flow_node()

parse_flow_sequence_entry(*first=False*)

parse_flow_sequence_entry_mapping_end()

parse_flow_sequence_entry_mapping_key()

parse_flow_sequence_entry_mapping_value()

parse_flow_sequence_first_entry()

parse_implicit_document_start()

parse_indentless_sequence_entry()

parse_node(*block=False*, *indentless_sequence=False*)

parse_stream_start()

peek_event()

process_directives()

process_empty_scalar(*mark*)

exception MakeItSo.yaml.parser.**ParserError**(*context=None*, *context_mark=None*, *problem=None*, *problem_mark=None*, *note=None*)

Bases: MakeItSo.yaml.error.MarkedYAMLError

reader Module

class MakeItSo.yaml.reader.**Reader**(*stream*)

Bases: object

check_printable(*data*)

determine_encoding()

forward(*length=1*)

get_mark()

peek(*index=0*)

prefix(*length=1*)

update(*length*)

update_raw(*size=1024*)

exception MakeItSo.yaml.reader.**ReaderError**(*name*, *position*, *character*, *encoding*, *reason*)

Bases: MakeItSo.yaml.error.YAMLError

representer Module

class MakeItSo.yaml.representer.**BaseRepresenter**(*default_style=None*, *default_flow_style=None*)

Bases: object

classmethod add_multi_representer(*data_type*, *representer*)

classmethod add_representer(*data_type*, *representer*)

get_classobj_bases(*cls*)

ignore_aliases(*data*)

represent(*data*)

represent_data(*data*)

represent_mapping(*tag*, *mapping*, *flow_style=None*)

represent_scalar(*tag*, *value*, *style=None*)

represent_sequence(*tag*, *sequence*, *flow_style=None*)

class MakeItSo.yaml.representer.**SafeRepresenter**(*default_style=None*, *default_flow_style=None*)

Bases: MakeItSo.yaml.representer.BaseRepresenter

ignore_aliases(*data*)

represent_bool(*data*)

represent_date(*data*)

represent_datetime(*data*)

represent_dict(*data*)

represent_float(*data*)

represent_int(*data*)

represent_list(*data*)

represent_long(*data*)

represent_none(*data*)

represent_set(*data*)

represent_str(*data*)

represent_undefined(*data*)

represent_unicode(*data*)

represent_yaml_object(*tag*, *data*, *cls*, *flow_style=None*)

class MakeItSo.yaml.representer.**Representer**(*default_style=None*, *default_flow_style=None*)

Bases: MakeItSo.yaml.representer.SafeRepresenter

represent_complex(*data*)

represent_instance(*data*)

represent_long(*data*)

represent_module(*data*)

represent_name(*data*)

represent_object(*data*)

represent_str(*data*)

represent_tuple(*data*)

represent_unicode(*data*)

exception MakeItSo.yaml.representer.**RepresenterError**

Bases: MakeItSo.yaml.error.YAMLError

resolver Module

class MakeItSo.yaml.resolver.**BaseResolver**
Bases: object

classmethod add_implicit_resolver(*tag*, *regexp*, *first*)

classmethod add_path_resolver(*tag*, *path*, *kind=None*)

ascend_resolver()

check_resolver_prefix(*depth*, *path*, *kind*, *current_node*, *current_index*)

descend_resolver(*current_node*, *current_index*)

resolve(*kind*, *value*, *implicit*)

class MakeItSo.yaml.resolver.**Resolver**
Bases: MakeItSo.yaml.resolver.BaseResolver

scanner Module

class MakeItSo.yaml.scanner.**Scanner**
Bases: object

add_indent(*column*)

check_block_entry()

check_directive()

check_document_end()

check_document_start()

check_key()

check_plain()

check_token(**choices*)

check_value()

fetch_alias()

fetch_anchor()

fetch_block_entry()

fetch_block_scalar(*style*)

fetch_directive()

fetch_document_end()

fetch_document_indicator(*TokenClass*)

fetch_document_start()

fetch_double()

fetch_flow_collection_end(*TokenClass*)

fetch_flow_collection_start(*TokenClass*)

fetch_flow_entry()

fetch_flow_mapping_end()

fetch_flow_mapping_start()

fetch_flow_scalar(*style*)

fetch_flow_sequence_end()

fetch_flow_sequence_start()

fetch_folded()

fetch_key()

fetch_literal()

fetch_more_tokens()

fetch_plain()

fetch_single()

fetch_stream_end()

fetch_stream_start()

fetch_tag()

fetch_value()

get_token()

need_more_tokens()

next_possible_simple_key()

peek_token()

remove_possible_simple_key()

save_possible_simple_key()

scan_anchor(*TokenClass*)

scan_block_scalar(*style*)

scan_block_scalar_breaks(*indent*)

scan_block_scalar_ignored_line(*start_mark*)

scan_block_scalar_indentation()

scan_block_scalar_indicators(*start_mark*)

scan_directive()

scan_directive_ignored_line(*start_mark*)

scan_directive_name(*start_mark*)

scan_flow_scalar(*style*)

scan_flow_scalar_breaks(*double*, *start_mark*)

scan_flow_scalar_non_spaces(*double*, *start_mark*)

scan_flow_scalar_spaces(*double*, *start_mark*)

scan_line_break()

scan_plain()

scan_plain_spaces(*indent*, *start_mark*)

scan_tag()

scan_tag_directive_handle(*start_mark*)

scan_tag_directive_prefix(*start_mark*)

scan_tag_directive_value(*start_mark*)

scan_tag_handle(*name*, *start_mark*)

scan_tag_uri(*name*, *start_mark*)

scan_to_next_token()

scan_uri_escapes(*name*, *start_mark*)

scan_yaml_directive_number(*start_mark*)

scan_yaml_directive_value(*start_mark*)

stale_possible_simple_keys()

unwind_indent(*column*)

exception MakeItSo.yaml.scanner.**ScannerError**(*context=None*, *context_mark=None*, *problem=None*, *problem_mark=None*, *note=None*)

Bases: MakeItSo.yaml.error.MarkedYAMLError

serializer Module

class MakeItSo.yaml.serializer.**Serializer**(*encoding=None*, *explicit_start=None*, *explicit_end=None*, *version=None*, *tags=None*)

Bases: object

anchor_node(*node*)

close()

generate_anchor(*node*)

open()

serialize(*node*)

serialize_node(*node*, *parent*, *index*)

exception MakeItSo.yaml.serializer.**SerializerError**

Bases: MakeItSo.yaml.error.YAMLError

tokens Module

class MakeItSo.yaml.tokens.**AliasToken**(*value*, *start_mark*, *end_mark*)

Bases: MakeItSo.yaml.tokens.Token

class MakeItSo.yaml.tokens.**AnchorToken**(*value*, *start_mark*, *end_mark*)

Bases: MakeItSo.yaml.tokens.Token

class MakeItSo.yaml.tokens.**BlockEndToken**(*start_mark*, *end_mark*)

Bases: MakeItSo.yaml.tokens.Token

class MakeItSo.yaml.tokens.**BlockEntryToken**(*start_mark*, *end_mark*)

Bases: MakeItSo.yaml.tokens.Token

class MakeItSo.yaml.tokens.**BlockMappingStartToken**(*start_mark*, *end_mark*)
Bases: MakeItSo.yaml.tokens.Token

class MakeItSo.yaml.tokens.**BlockSequenceStartToken**(*start_mark*, *end_mark*)
Bases: MakeItSo.yaml.tokens.Token

class MakeItSo.yaml.tokens.**DirectiveToken**(*name*, *value*, *start_mark*, *end_mark*)
Bases: MakeItSo.yaml.tokens.Token

class MakeItSo.yaml.tokens.**DocumentEndToken**(*start_mark*, *end_mark*)
Bases: MakeItSo.yaml.tokens.Token

class MakeItSo.yaml.tokens.**DocumentStartToken**(*start_mark*, *end_mark*)
Bases: MakeItSo.yaml.tokens.Token

class MakeItSo.yaml.tokens.**FlowEntryToken**(*start_mark*, *end_mark*)
Bases: MakeItSo.yaml.tokens.Token

class MakeItSo.yaml.tokens.**FlowMappingEndToken**(*start_mark*, *end_mark*)
Bases: MakeItSo.yaml.tokens.Token

class MakeItSo.yaml.tokens.**FlowMappingStartToken**(*start_mark*, *end_mark*)
Bases: MakeItSo.yaml.tokens.Token

class MakeItSo.yaml.tokens.**FlowSequenceEndToken**(*start_mark*, *end_mark*)
Bases: MakeItSo.yaml.tokens.Token

class MakeItSo.yaml.tokens.**FlowSequenceStartToken**(*start_mark*, *end_mark*)
Bases: MakeItSo.yaml.tokens.Token

class MakeItSo.yaml.tokens.**KeyToken**(*start_mark*, *end_mark*)
Bases: MakeItSo.yaml.tokens.Token

class MakeItSo.yaml.tokens.**ScalarToken**(*value*, *plain*, *start_mark*, *end_mark*, *style=None*)
Bases: MakeItSo.yaml.tokens.Token

class MakeItSo.yaml.tokens.**StreamEndToken**(*start_mark*, *end_mark*)
Bases: MakeItSo.yaml.tokens.Token

class MakeItSo.yaml.tokens.**StreamStartToken**(*start_mark=None*, *end_mark=None*, *encoding=None*)
Bases: MakeItSo.yaml.tokens.Token

class MakeItSo.yaml.tokens.**TagToken**(*value*, *start_mark*, *end_mark*)
Bases: MakeItSo.yaml.tokens.Token

class MakeItSo.yaml.tokens.**Token**(*start_mark*, *end_mark*)
Bases: object

class MakeItSo.yaml.tokens.**ValueToken**(*start_mark*, *end_mark*)
Bases: MakeItSo.yaml.tokens.Token

CHAPTER

SIX

FREQUENTLY ASKED QUESTIONS

6.1 Configuration Questions

6.1.1 I get the error `python: No such file or directory`

- Solution: Your environment cannot find the path to the Python executable. Locate the `python` binary and either add it to your `PATH` environment variable:

  ```
  export PATH=/path/to/python/bin:$PATH)
  ```

  or execute makeItSo via the full path to Python, such as:

  ```
  /path/to/bin/python makeItSo.py
  ```

6.1.2 I get the error `ImportError: No module named Environments`

- Solution: You must not move "makeItSo.py" or any of the subdirectories (Environments, Models, share, yaml, etc.). To run makeItSo, use an absolute path to the makeItSo.py script.

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

m

INDEX

D

E

F

G

I

K

L

M

N

O

P

R

S

T

U

V

W

X

Y