runmanager

Release 3.0.0rc2.dev22+g2181d10

labscript suite contributors

Jan 29, 2021

DOCUMENTATION

CHAPTER

ONE

API REFERENCE

exception runmanager.**ExpansionError**
Bases: Exception

An exception class so that error handling code can tell when a parsing exception was caused by a mismatch with the expansion mode

class runmanager.**TraceDictionary**(**args*, ***kwargs*)
Bases: dict

runmanager.**add_expansion_groups**(*filename*)
backward compatability, for globals files which don't have expansion groups. Create them if they don't exist. Guess expansion settings based on datatypes, if possible.

runmanager.**compile_labscript**(*labscript_file*, *run_file*)
Compiles labscript_file with the run file, returning the processes return code, stdout and stderr.

runmanager.**compile_labscript_async**(*labscript_file*, *run_file*, *stream_port*, *done_callback*)
Compiles labscript_file with run_file. This function is designed to be called in a thread. The stdout and stderr from the compilation will be shovelled into stream_port via zmq push as it spews forth, and when compilation is complete, done_callback will be called with a boolean argument indicating success. Note that the zmq communication will be encrypted, or not, according to security settings in labconfig. If you want to receive the data on a zmq socket, do so using a PULL socket created from a labscript_utils.ls_zprocess.Context, or using a labscript_utils.ls_zprocess.ZMQServer. These subclasses will also be configured with the appropriate security settings and will be able to receive the messages.

runmanager.**compile_labscript_with_globals_files**(*labscript_file*, *globals_files*, *output_path*)
Creates a run file output_path, using all the globals from globals_files. Compiles labscript_file with the run file, returning the processes return code, stdout and stderr.

runmanager.**compile_labscript_with_globals_files_async**(*labscript_file*, *globals_files*, *output_path*, *stream_port*, *done_callback*)

Same as compile_labscript_with_globals_files, except it launches a thread to do the work and does not return anything. Instead, stderr and stdout will be put to stream_port via zmq push in the multipart message format ['stdout','hello, world

'] etc. When compilation is finished, the function done_callback will be called a boolean argument indicating success or failure. If you want to receive the data on a zmq socket, do so using a PULL socket created from a labscript_utils.ls_zprocess.Context, or using a labscript_utils.ls_zprocess.ZMQServer. These subclasses will also be configured with the appropriate security settings and will be able to receive the messages.

runmanager.**compile_multishot_async**(*labscript_file*, *run_files*, *stream_port*, *done_callback*)
Compiles labscript_file with run_files. This function is designed to be called in a thread. The stdout and stderr from the compilation will be shovelled into stream_port via zmq push as it spews forth, and when each

compilation is complete, done_callback will be called with a boolean argument indicating success. Compilation will stop after the first failure. If you want to receive the data on a zmq socket, do so using a PULL socket created from a labscript_utils.ls_zprocess.Context, or using a labscript_utils.ls_zprocess.ZMQServer. These subclasses will also be configured with the appropriate security settings and will be able to receive the messages.

runmanager.**copy_group**(*source_globals_file*, *source_groupname*, *dest_globals_file*, *delete_source_group=False*)

This function copies the group source_groupname from source_globals_file to dest_globals_file and renames the new group so that there is no name collision. If delete_source_group is False the copyied files have a suffix '_copy'.

runmanager.**dict_diff**(*dict1*, *dict2*)

Return the difference between two dictionaries as a dictionary of key: [val1, val2] pairs. Keys unique to either dictionary are included as key: [val1, '-'] or key: ['-', val2].

runmanager.**evaluate_globals**(*sequence_globals*, *raise_exceptions=True*)

Takes a dictionary of globals as returned by get_globals. These globals are unevaluated strings. Evaluates them all in the same namespace so that the expressions can refer to each other. Iterates to allow for NameErrors to be resolved by subsequently defined globals. Throws an exception if this does not result in all errors going away. The exception contains the messages of all exceptions which failed to be resolved. If raise_exceptions is False, any evaluations resulting in an exception will instead return the exception object in the results dictionary

runmanager.**expand_globals**(*sequence_globals*, *evaled_globals*, *expansion_config=None*, *return_dimensions=False*)

Expands iterable globals according to their expansion settings. Creates a number of 'axes' which are to be outer product'ed together. Some of these axes have only one element, these are globals that do not vary. Some have a set of globals being zipped together, iterating in lock-step. Others contain a single global varying across its values (the globals set to 'outer' expansion). Returns a list of shots, each element of which is a dictionary for that shot's globals.

runmanager.**find_comments**(*src*)

Return a list of start and end indices for where comments are in given Python source. Comments on separate lines with only whitespace in between them are coalesced. Whitespace preceding a comment is counted as part of the comment.

runmanager.**flatten_globals**(*sequence_globals*, *evaluated=False*)

Flattens the data structure of the globals. If evaluated=False, saves only the value expression string of the global, not the units or expansion.

runmanager.**get_all_groups**(*h5_files*)

returns a dictionary of group_name: h5_path pairs from a list of h5_files.

runmanager.**get_globals**(*groups*)

Takes a dictionary of group_name: h5_file pairs and pulls the globals out of the groups in their files. The globals are strings storing python expressions at this point. All these globals are packed into a new dictionary, keyed by group_name, where the values are dictionaries which look like {global_name: (expression, units, expansion), ...}

runmanager.**get_shot_globals**(*filepath*)

Returns the evaluated globals for a shot, for use by labscript or lyse. Simple dictionary access as in dict(h5py.File(filepath).attrs) would be fine except we want to apply some hacks, so it's best to do that in one place.

runmanager.**globals_diff_groups**(*active_groups*, *other_groups*, *max_cols=1000*, *return_string=True*)

Given two sets of globals groups, perform a diff of the raw and evaluated globals.

runmanager.**make_run_file_from_globals_files**(*labscript_file*, *globals_files*, *output_path*, *config=None*)

Creates a run file output_path, using all the globals from globals_files. Uses labscript_file to determine the

sequence_attrs only

runmanager.**make_run_files**(*output_folder*, *sequence_globals*, *shots*, *sequence_attrs*, *filename_prefix*, *shuffle=False*)

Does what it says. sequence_globals and shots are of the datatypes returned by get_globals and get_shots, one is a nested dictionary with string values, and the other a flat dictionary. sequence_attrs is a dict of the attributes pertaining to this sequence to be initially set at the top-level group of the h5 file, as returned by new_sequence_details. output_folder and filename_prefix determine the directory shot files will be output to, as well as their filenames (this function will generate filenames with the shot number and .h5 extension appended to filename_prefix). Sensible defaults for these are also returned by new_sequence_details(), so preferably these should be used.

Shuffle will randomise the order that the run files are generated in with respect to which element of shots they come from. This function returns a *generator*. The run files are not actually created until you loop over this generator (which gives you the filepaths). This is useful for not having to clean up as many unused files in the event of failed compilation of labscripts. If you want all the run files to be created at some point, simply convert the returned generator to a list. The filenames the run files are given is simply the sequence_id with increasing integers appended.

runmanager.**make_single_run_file**(*filename*, *sequenceglobals*, *runglobals*, *sequence_attrs*, *run_no*, *n_runs*)

Does what it says. runglobals is a dict of this run's globals, the format being the same as that of one element of the list returned by expand_globals. sequence_globals is a nested dictionary of the type returned by get_globals. sequence_attrs is a dict of attributes pertaining to this sequence, as returned by new_sequence_details. run_no and n_runs must be provided, if this run file is part of a sequence, then they should reflect how many run files are being generated in this sequence, all of which must have identical sequence_attrs.

runmanager.**new_sequence_details**(*script_path*, *config=None*, *increment_sequence_index=True*)

Generate the details for a new sequence: the toplevel attrs sequence_date, sequence_index, sequence_id; and the the output directory and filename prefix for the shot files, according to labconfig settings. If increment_sequence_index=True, then we are claiming the resulting sequence index for use such that it cannot be used by anyone else. This should be done if the sequence details are immediately about to be used to compile a sequence. Otherwise, set increment_sequence_index to False, but in that case the results are indicative only and one should call this function again with increment_sequence_index=True before compiling the sequence, as otherwise the sequence_index may be used by other code in the meantime.

runmanager.**next_sequence_index**(*shot_basedir*, *dt*, *increment=True*)

Return the next sequence index for sequences in the given base directory (i.e. <experiment_shot_storage>/<script_basename>) and the date of the given datetime object, and increment the sequence index atomically on disk if increment=True. If not setting increment=True, then the result is indicative only and may be used by other code at any time. One must increment the sequence index prior to use.

runmanager.**remove_comments_and_tokenify**(*src*)

Removes comments from source code, leaving it otherwise intact, and returns it. Also returns the raw tokens for the code, allowing comparisons between source to be made without being sensitive to whitespace.

CHAPTER

TWO

LABSCRIPT SUITE COMPONENTS

The *labscript suite* is modular by design, and is comprised of:

Table 1: Python libraries

	labscript — Expressive composition of hardware-timed experiments
	labscript-devices — Plugin architecture for controlling experiment hardware
	labscript-utils — Shared modules used by the *labscript suite*

Table 2: Graphical applications

	runmanager — Graphical and remote interface to parameterized experiments
	blacs — Graphical interface to scientific instruments and experiment supervision
	lyse — Online analysis of live experiment data
	runviewer — Visualize hardware-timed experiment instructions

PYTHON MODULE INDEX

r

INDEX