Instant Documentation

Release 2018.1.0.dev0

FEniCS Project

Jan 08, 2018

Contents

Instant is a Python module that allows for instant inlining of C and C++ code in Python. It is a small Python module built on top of SWIG and Distutils.

Instant is part of the FEniCS Project.

For more information, visit http://www.fenicsproject.org.

CHAPTER 1

Documentation

1.1 Installation

Instant is normally installed as part of an installation of FEniCS. If you are using Instant as part of the FEniCS software suite, it is recommended that you follow the installation instructions for FEniCS.

To install Instant itself, read on below for a list of requirements and installation instructions.

1.1.1 Requirements and dependencies

Instant requires Python version 2.7 or later and depends on the following Python packages:

- NumPy
- SWIG

These packages will be automatically installed as part of the installation of Instant, if not already present on your system.

If running on a cluster with Infiniband with python 2, you also need to install a backport of the subprocess module from python 3 to get safe fork behaviour:

- subprocess32

In addition, Instant optionally depends on flufl.lock for NFS safe file locking flufl.lock can be installed

- flufl.lock (https://gitlab.com/warsaw/flufl.lock)

1.1.2 Installation instructions

To install Instant, download the source code from the Instant Bitbucket repository, and run the following command:

```
pip install .
```

To install to a specific location, add the `--prefix` flag to the installation command:

```
pip install --prefix=<some directory> .
```

1.1.3 Environment

Instant's behaviour depened on following environment variables:

- `INSTANT_CACHE_DIR`
- `INSTANT_ERROR_DIR`

  These options can override placement of default cache and error directories. The default directories are placed below the prefix of the currently active virtualenv or conda environment, in `.cache/instant/cache` and `.cache/instant/error`. If no such environment is active, the default directories are `~/.cache/instant/pythonM.N/cache` and `.cache/instant/pythonM.N/error`.

- `INSTANT_SYSTEM_CALL_METHOD`

  Choose method for calling external programs (pkgconfig, swig, cmake, make). Available values:

  - `SUBPROCESS`

    Uses pipes. Not OFED-fork safe on Python 2 unless subprocess32 has been installed. Default.

  - `OS_SYSTEM`

    Uses temporary files. Probably OFED-fork safe.

Warning: OFED-fork safe system call method might be required to avoid crashes on OFED-based (InfiniBand) clusters! If using python 2, installing subprocess32 is recommended.

1.2 User manual

Note: This page is work in progress.

1.3 instant package

1.3.1 Submodules

1.3.2 instant.build module

This module contains the main part of Instant, the build_module function.

`instant.build.`**`arg_strings`**`(x)`

`instant.build.`**`assert_is_bool`**`(x)`

`instant.build.`**`assert_is_str`**`(x)`

`instant.build.`**`assert_is_str_list`**`(x)`

instant.build.**build_module**(*modulename=None, source_directory='.', code='', init_code='', additional_definitions='', additional_declarations='', sources=[], wrap_headers=[], local_headers=[], system_headers=[], include_dirs=['.'], library_dirs=[], libraries=[], swigargs=['-c++', '-fcompact', '-O', '-I.', '-small'], swig_include_dirs=[], cppargs=['-O2'], lddargs=[], object_files=[], arrays=[], generate_interface=True, generate_setup=True, cmake_packages=[], signature=None, cache_dir=None*)

Generate and compile a module from C/C++ code using SWIG.

The keyword arguments are as follows:

- B{modulename}: - The name you want for the module.

  If specified, the module will not be cached. If missing, a name will be constructed based on a checksum of the other arguments, and the module will be placed in the global cache. String.

- B{source_directory}: - The directory where user supplied files reside. The files given in B{sources}, B{wrap_headers}, and B{local_headers} are expected to exist in this directory. String.

- B{code}: - A string containing C or C++ code to be compiled and wrapped. String.

- B{init_code}: - Code that should be executed when the Instant module is imported. This code is inserted in the SWIG interface file, and is used for instance for calling C{import_array()} used for the initialization of NumPy arrays. String.

- B{additional_definitions}: - Additional definitions (typically needed for inheritance) for interface file. These definitions should be given as triple-quoted strings in the case they span multiple lines, and are placed both in the initial block for C/C++ code (C{%{,%}}-block), and the main section of the interface file. String.

- B{additional_declarations}: - Additional declarations (typically needed for inheritance) for interface file. These declarations should be given as triple-quoted strings in the case they span multiple lines, and are plaves in the main section of the interface file. String.

- B{sources}: - Source files to compile and link with the module. These files are compiled togehter with the SWIG-generated wrapper file into the final library file. Should reside in directory specified in B{source_directory}. List of strings.

- B{wrap_headers}: - Local header files that should be wrapped by SWIG. The files specified will be included both in the initial block for C/C++ code (with a C directive) and in the main section of the interface file (with a SWIG directive). Should reside in directory specified in B{source_directory}. List of strings.

- B{local_headers}: - Local header files required to compile the wrapped code. The files specified will be included in the initial block for C/C++ code (with a C directive). Should reside in directory specified in B{source_directory}. List of strings.

- B{system_headers}: - System header files required to compile the wrapped code. The files specified will be included in the initial block for C/C++ code (with a C directive). List of strings.

- B{include_dirs}: - Directories to search for header files for building the extension module. Needs to be absolute path names. List of strings.

- B{library_dirs}: - Directories to search for libraries (C{-l}) for building the extension module. Needs to be absolute paths. List of strings.

- B{libraries}: - Libraries needed by the Instant module. The libraries will be linked in from the shared object file. The initial C{-l} is added automatically. List of strings.

- B{swigargs}: - List of arguments to swig, e.g. C{["-lpointers.i"]}

  to include the SWIG pointers.i library.

- B{swig_include_dirs}: - A list of directories to include in the 'swig' command.
- B{cppargs}: - List of arguments to the compiler, e.g. C{["-Wall", "-fopenmp"]}.
- B{lddargs}: - List of arguments to the linker, e.g. C{["-E", "-U"]}.
- B{object_files}: - If you want to compile the files yourself. TODO: Not yet supported.
- B{arrays}: - A nested list describing the C arrays to be made from NumPy arrays. The SWIG interface for fil NumPy is used. For 1D arrays, the inner list should contain strings with the variable names for the length of the arrays and the array itself. 2D matrices should contain the names of the dimensions in the two directions as well as the name of the array, and 3D tensors should contain the names of the dimensions in the three directions in addition to the name of the array. If the NumPy array har more than four dimensions, the inner list should contain strings with variable names for the number of dimensions, the length in each dimension as a pointer, and the array itself, respectively.
- B{generate_interface}: - A bool to indicate if you want to generate the interface files.
- B{generate_setup}: - A bool to indicate if you want to generate the setup.py file.
- B{cmake_packages}: - A list with CMake configured packages which are used to configure and build the extension module. If used it will override the default behaviour of using distutils.
- B{signature}: - A signature string to identify the form instead of the source code.
- B{cache_dir}: - A directory to look for cached modules and place new ones.

  If missing, a default directory is used. Note that the module will not be cached if B{modulename} is specified. The cache directory should not be used for anything else.

instant.build.**build_module_vmtk**(*c_code*, *cache_dir=None*)

instant.build.**build_module_vtk**(*c_code*, *cache_dir=None*)

instant.build.**copy_files**(*source*, *dest*, *files*)
Copy a list of files from a source directory to a destination directory. This may seem a bit complicated, but a lot of this code is error checking.

instant.build.**copy_to_cache**(*module_path*, *cache_dir*, *modulename*, *check_for_existing_path=True*)
Copy module directory to cache.

instant.build.**makedirs**(*path*)
Creates a directory (tree). If directory already excists it does nothing.

instant.build.**recompile**(*modulename*, *module_path*, *new_compilation_checksum*, *build_system='distutils'*)
Recompile module if the new checksum is different from the one in the checksum file in the module directory.

instant.build.**strip_strings**(*x*)

1.3.3 instant.cache module

This module contains helper functions for working with the module cache.

Example operations:

- modulename = modulename_from_checksum(checksum)
- modulename = modulename_from_checksum(compute_checksum(signature))
- module = import_module_directly(path, modulename)
- module = import_module(modulename)

- module = import_module(checksum)
- module = import_module(compute_checksum(signature))
- modules = cached_modules()
- modules = cached_modules(cache_dir)

instant.cache.**cached_modules**(*cache_dir=None*)
Return a list with the names of all cached modules.

instant.cache.**check_disk_cache**(*modulename*, *cache_dir*, *moduleids*)

instant.cache.**check_memory_cache**(*moduleid*)

instant.cache.**checksum_from_modulename**(*modulename*)
Construct a module name from a checksum for use in cache.

instant.cache.**import_and_cache_module**(*path*, *modulename*, *moduleids*)

instant.cache.**import_module**(*moduleid*, *cache_dir=None*)
Import module from cache given its moduleid and an optional cache directory.

The moduleid can be either

- the module name
- a signature string, of which a checksum is taken to look up in the cache
- a checksum string, which is used directly to look up in the cache
- a hashable non-string object with a function moduleid.signature() which is used to get a signature string

The hashable object is used to look up in the memory cache before signature() is called. If the module is found on disk, it is placed in the memory cache.

instant.cache.**import_module_directly**(*path*, *modulename*)
Import a module with the given module name that resides in the given path.

instant.cache.**is_valid_module_name**(*name*)

instant.cache.**memory_cached_module**(*moduleid*)
Returns the cached module if found.

instant.cache.**modulename_from_checksum**(*checksum*)
Construct a module name from a checksum for use in cache.

instant.cache.**place_module_in_memory_cache**(*moduleid*, *module*)
Place a compiled module in cache with given id.

1.3.4 instant.codegeneration module

This module contains helper functions for code generation.

instant.codegeneration.**create_typemaps**(*classes*)

instant.codegeneration.**find_vtk_classes**(*str*)

instant.codegeneration.**generate_interface_file_vtk**(*signature*, *code*)

instant.codegeneration.**generate_vtk_includes**(*classes*)

instant.codegeneration.**mapstrings**(*format*, *sequence*)

instant.codegeneration.**reindent**(*code*)

Reindent a multiline string to allow easier to read syntax.

Each line will be indented relative to the first non-empty line. Start the first line without text like shown in this example:

```
code = reindent("""
    Foo
    Bar
        Blatti
    Ping
    """)
```

makes all indentation relative to Foo.

instant.codegeneration.**unique**(*sequence*)

instant.codegeneration.**write_cmakefile**(*module_name*, *cmake_packages*, *csrcs*, *cppsrcs*, *local_headers*, *include_dirs*, *library_dirs*, *libraries*, *swig_include_dirs*, *swigargs*, *cppargs*, *lddargs*)

instant.codegeneration.**write_interfacefile**(*filename*, *modulename*, *code*, *init_code*, *additional_definitions*, *additional_declarations*, *system_headers*, *local_headers*, *wrap_headers*, *arrays*)

Generate a SWIG interface file. Intended for internal library use.

The input arguments are as follows:

- modulename (Name of the module)
- code (Code to be wrapped)
- init_code (Code to put in the init section of the interface file)
- additional_definitions (Definitions to be placed in initial block with C code as well as in the main section of the SWIG interface file)
- additional_declarations (Declarations to be placed in the main section of the SWIG interface file)
- system_headers (A list of system headers with declarations needed by the wrapped code)
- local_headers (A list of local headers with declarations needed by the wrapped code)
- wrap_headers (A list of local headers that will be included in the code and wrapped by SWIG)
- arrays (A nested list, the inner lists describing the different arrays)

The result of this function is that a SWIG interface with the name modulename.i is written to the current directory.

instant.codegeneration.**write_itk_cmakefile**(*name*)

instant.codegeneration.**write_setup**(*filename*, *modulename*, *csrcs*, *cppsrcs*, *local_headers*, *include_dirs*, *library_dirs*, *libraries*, *swig_include_dirs*, *swigargs*, *cppargs*, *lddargs*)

Generate a setup.py file. Intended for internal library use.

instant.codegeneration.**write_vmtk_cmakefile**(*name*)

instant.codegeneration.**write_vtk_interface_file**(*signature*, *code*)

1.3.5 instant.config module

This module contains helper functions for configuration using pkg-config.

instant.config.**check_and_set_swig_binary**(*binary='swig'*, *path=''*)
Check if a particular swig binary is available

instant.config.**check_swig_version**(*version*, *same=False*)
Check the swig version

Returns True if the version of the installed swig is equal or greater than the version passed to the function.

If same is True, the function returns True if and only if the two versions are the same.

Usage: if instant.check_swig_version('1.3.36'):

print "Swig version is greater than or equal to 1.3.36"

else: print "Swig version is lower than 1.3.36"

instant.config.**get_swig_binary**()
Return any cached swig binary

instant.config.**get_swig_version**()
Return the current swig version in a 'str'

instant.config.**header_and_libs_from_pkgconfig**(**packages*, ***kwargs*)
This function returns list of include files, flags, libraries and library directories obtain from a pkgconfig file.

The usage is: (includes, flags, libraries, libdirs) = header_and_libs_from_pkgconfig(*list_of_packages)

or: (includes, flags, libraries, libdirs, linkflags) = header_and_libs_from_pkgconfig(*list_of_packages, returnLinkFlags=True)

1.3.6 instant.inlining module

This module contains the inline* functions, which allows easy inlining of C/C++ functions.

instant.inlining.**get_func_name**(*c_code*)

instant.inlining.**inline**(*c_code*, ***kwargs*)
This is a short wrapper around the build_module function in instant.

It creates a module given that the input is a valid C function. It is only possible to inline one C function each time.

Usage:

```
>>> from instant import inline
>>> add_func = inline("double add(double a, double b){ return a+b; }")
>>> print "The sum of 3 and 4.5 is ", add_func(3, 4.5)
```

instant.inlining.**inline_module**(*c_code*, ***kwargs*)
This is a short wrapper around the build_module function in instant.

It creates a module given that the input is a valid C function. It is only possible to inline one C function each time.

Usage:

```
>>> from instant import inline
>>> add_func = inline("double add(double a, double b){ return a+b; }")
>>> print "The sum of 3 and 4.5 is ", add_func(3, 4.5)
```

instant.inlining.**inline_module_with_numpy**(*c_code*, ***kwargs*)

This is a short wrapper around the build_module function in instant.

It creates a module given that the input is a valid C function. It is only possible to inline one C function each time. The difference between this function and the inline function is that C-arrays can be used. The following example illustrates that.

Usage:

```
>>> import numpy
>>> import time
>>> from instant import inline_with_numpy
>>> c_code = """
    double sum (int n1, double* array1){
        double tmp = 0.0;
        for (int i=0; i<n1; i++) {
            tmp += array1[i];
        }
        return tmp;
    }
    """
>>> sum_func = inline_with_numpy(c_code,  arrays = [['n1', 'array1']])
>>> a = numpy.arange(10000000); a = numpy.sin(a)
>>> sum_func(a)
```

instant.inlining.**inline_vmtk**(*c_code*, *cache_dir=None*)

instant.inlining.**inline_vtk**(*c_code*, *cache_dir=None*)

instant.inlining.**inline_with_numpy**(*c_code*, ***kwargs*)

This is a short wrapper around the build_module function in instant.

It creates a module given that the input is a valid C function. It is only possible to inline one C function each time. The difference between this function and the inline function is that C-arrays can be used. The following example illustrates that.

Usage:

```
>>> import numpy
>>> import time
>>> from instant import inline_with_numpy
>>> c_code = """
    double sum (int n1, double* array1){
        double tmp = 0.0;
        for (int i=0; i<n1; i++) {
            tmp += array1[i];
        }
        return tmp;
    }
    """
>>> sum_func = inline_with_numpy(c_code,  arrays = [['n1', 'array1']])
>>> a = numpy.arange(10000000); a = numpy.sin(a)
>>> sum_func(a)
```

1.3.7 instant.locking module

File locking for the cache system, to avoid problems when multiple processes work with the same module. Only works on UNIX systems.

Two Python libraries can be used:

flufl.lock : A nfs safe which can be downloaded from:

https://launchpad.net/flufl.lock

fcntl [A builtin Python module which only works on posix machines] and it is does unfortunately not work on nfs

instant.locking.**get_lock**(*cache_dir*, *module_name*)
Get a new file lock.

instant.locking.**release_lock**(*lock*)
Release a lock currently held by Instant.

class instant.locking.**file_lock**(*cache_dir*, *module_name*)
Bases: object

File lock using with statement

1.3.8 instant.output module

This module contains internal logging utilities.

instant.output.**get_log_handler**()

instant.output.**get_logger**()

instant.output.**get_status_output**(*cmd*, *input=None*, *cwd=None*, *env=None*)

instant.output.**instant_assert**(*condition*, **message*)

instant.output.**instant_debug**(**message*)

instant.output.**instant_error**(**message*)

instant.output.**instant_info**(**message*)

instant.output.**instant_warning**(**message*)

instant.output.**set_log_handler**(*handler*)

instant.output.**set_log_level**(*level*)

instant.output.**set_logging_level**(*level*)

instant.output.**write_file**(*filename*, *text*, *mode='w'*)
Write text to a file and close it.

1.3.9 instant.paths module

This module contains helper functions for working with temp and cache directories.

instant.paths.**delete_temp_dir**()
Delete the temporary directory created by get_temp_dir().

instant.paths.**get_default_cache_dir**()
Return the default cache directory.

instant.paths.**get_default_error_dir**()
Return the default error directory.

instant.paths.**get_instant_dir**()
Return the default instant directory, creating it if necessary.

instant.paths.**get_temp_dir**()
Return a temporary directory for the duration of this process.

Multiple calls in the same process returns the same directory. Remember to call delete_temp_dir() before exiting.

instant.paths.**makedirs**(*path*)
Creates a directory (tree). If directory already excists it does nothing.

instant.paths.**validate_cache_dir**(*cache_dir*)

1.3.10 instant.signatures module

This module contains helper functions for working with checksums.

instant.signatures.**compute_checksum**(*text=''*, *filenames=[]*)
Get the checksum value of filename modified based on Python24ToolsScriptsmd5.py

1.3.11 Module contents

Instant allows compiled C/C++ modules to be created at runtime in your Python application, using SWIG to wrap the C/C++ code.

A simple example:

```
>>> from instant import inline
>>> add_func = inline("double add(double a, double b){ return a+b; }")
>>> print "The sum of 3 and 4.5 is ", add_func(3, 4.5)
```

The main functions are C{build_module}, C{write_code}, and C{inline*} see their documentation for more details.

For more examples, see the tests/ directory in the Instant distribution.

Questions, bugs and patches should be sent to fenics-dev@googlegroups.com.

1.4 Release notes

1.4.1 Changes in the next release

Summary of changes

Note: Developers should use this page to track and list changes during development. At the time of release, this page should be published (and renamed) to list the most important changes in the new release.

Detailed changes

Note: At the time of release, make a verbatim copy of the ChangeLog here (and remove this note).

1.4.2 Changes in version 2017.2.0

Instant 2017.2.0 was released on 2017-12-05.

1.4.3 Changes in version 2017.1.0.post1

Instant 2017.1.0.post1 was released on 2017-09-12.

- Change PyPI package name to fenics-instant.

1.4.4 Changes in version 2017.1.0

Instant 2017.1.0 was released on 2017-05-09.

- Minor fixes

1.4.5 Changes in version 2016.2.0

Instant 2016.2.0 was released on 2016-11-30.

- Add Python version string to hash signature input
- Add pipelines testing, with py2 and py3 coverage
- Remove commands module (removed from Py3)
- Switch unit tests to pytest

1.4.6 Changes in version 2016.1.0

Instant 2016.1.0 was released on 2016-06-23.

- Minor fixes

1.4.7 Changes in version 1.6.0

Instant 1.6.0 was released on 2015-07-28.

- Minor bug fixes

[FIXME: These links don't belong here, should go under API reference somehow.]

- genindex
- modindex

Python Module Index

i

Index

M

P

R

S

U

V

W