FEniCS Form Compiler (FFC) Documentation

Release 2019.2.0.dev0

FEniCS Project

Mar 26, 2020

Contents

1 Documentation 3

Python Module Index 75

Index 77

FFC is a compiler for finite element variational forms. From a high-level description of the form, it generates efficient low-level C++ code that can be used to assemble the corresponding discrete operator (tensor). In particular, a bilinear form may be assembled into a matrix and a linear form may be assembled into a vector. FFC may be used either from the command line (by invoking the `ffc` command) or as a Python module (`import ffc`).

FFC is part of the FEniCS Project.

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

CHAPTER 1

Documentation

1.1 Installation

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

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

1.1.1 Requirements and dependencies

FFC requires Python version 3.5 or later and depends on the following Python packages:

- NumPy
- six

FFC also depends on the following FEniCS Python packages:

- FIAT
- UFL
- dijitso

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

TSFC requirements

To use experimental `tsfc` representation, additional dependencies are needed:

- TSFC[1]
- COFFEE[1]
- FInAT[1]

[1] These are forks of the original packages tested to be compatible with FFC and updated frequently from upstream.

and in turn their additional dependencies:

- singledispatch[2]
- networkx[2]
- PuLP[2]

Note: TSFC requirements are not installed in FEniCS Docker images by default yet but they can be easilly installed on demand:

```
docker pull quay.io/fenicsproject/stable
docker run -ti --rm quay.io/fenicsproject/stable
  pip3 install --prefix=${FENICS_PREFIX} --no-cache-dir \
  git+https://github.com/blechta/tsfc.git@2018.1.0 \
  git+https://github.com/blechta/COFFEE.git@2018.1.0 \
  git+https://github.com/blechta/FInAT.git@2018.1.0 \
  singledispatch networkx pulp && \
  sudo rm -rf /tmp/* /var/tmp/*
```

The first two commands (or their modification, or `fenicsproject` helper script) are to be run on a host, while the last command, to be run in the container, actually installs all the TSFC requirements. For further reading, see FEniCS Docker reference.

1.1.2 Installation instructions

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

```
pip3 install .
```

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

```
pip3 install --prefix=<some directory> .
```

1.2 User manual

Note: This page is work in progress.

1.3 ffc package

1.3.1 Subpackages

ffc.errorcontrol package

Submodules

[2] Pip-installable.

ffc.errorcontrol.errorcontrol module

This module provides compilation of forms required for goal-oriented error control

ffc.errorcontrol.errorcontrol.**compile_with_error_control**(*forms*, *object_names*, *reserved_objects*, *prefix*, *parameters*)

Compile forms and additionally generate and compile forms required for performing goal-oriented error control

For linear problems, the input forms should be a bilinear form (a) and a linear form (L) specifying the variational problem and additionally a linear form (M) specifying the goal functional.

For nonlinear problems, the input should be linear form (F) and a functional (M) specifying the goal functional.

Arguments

forms (tuple) Three (linear case) or two (nonlinear case) forms specifying the primal problem and the goal

object_names (dict) Map from object ids to object names

reserved_names (dict) Map from reserved object names to object ids

prefix (string) Basename of header file

parameters (dict) Parameters for form compilation

ffc.errorcontrol.errorcontrolgenerators module

This module provides an abstract ErrorControlGenerator class for generating forms required for goal-oriented error control and a realization of this: UFLErrorControlGenerator for handling pure UFL forms.

class ffc.errorcontrol.errorcontrolgenerators.**ErrorControlGenerator**(*module*, *F*, *M*, *u*)

Bases: object

cell_residual()
Generate and return (bilinear, linear) forms defining linear variational problem for the strong cell residual

dual_forms()
Generate and return (bilinear, linear) forms defining linear dual variational problem

error_estimate()
Generate and return functional defining error estimate

error_indicators()
Generate and return linear form defining error indicators

facet_residual()
Generate and return (bilinear, linear) forms defining linear variational problem for the strong facet residual(s)

generate_all_error_control_forms()
Utility function for generating all (8) forms required for error control in addition to the primal forms

initialize_data()
Initialize specific data

primal_forms()
Return primal forms in order (bilinear, linear, functional)

class ffc.errorcontrol.errorcontrolgenerators.**UFLErrorControlGenerator**(*F*, *M*, *u*)

Bases: *ffc.errorcontrol.errorcontrolgenerators.ErrorControlGenerator*

This class provides a realization of ErrorControlGenerator for use with pure UFL forms

initialize_data()
Extract required objects for defining error control forms. This will be stored, reused and in particular named.

Module contents

This module contains functionality for working with automated goal-oriented error control. In particular it offers the following function:

compile_with_error_control - Compile forms and generate error control forms

ffc.quadrature package

Submodules

ffc.quadrature.codesnippets module

Code snippets for code generation.

ffc.quadrature.cpp module

This module defines rules and algorithms for generating C++ code.

ffc.quadrature.cpp.**count_ops**(*code*)
Count the number of operations in code (multiply-add pairs).

ffc.quadrature.cpp.**indent**(*block*, *num_spaces*)
Indent each row of the given string block with n spaces.

ffc.quadrature.cpp.**make_classname**(*prefix*, *basename*, *signature*)

ffc.quadrature.cpp.**make_integral_classname**(*prefix*, *integral_type*, *form_id*, *subdomain_id*)

ffc.quadrature.cpp.**remove_unused**(*code*, *used_set=set()*)
Remove unused variables from a given C++ code. This is useful when generating code that will be compiled with gcc and parameters -Wall -Werror, in which case gcc returns an error when seeing a variable declaration for a variable that is never used.

Optionally, a set may be specified to indicate a set of variables names that are known to be used a priori.

ffc.quadrature.cpp.**set_exception_handling**(*convert_exceptions_to_warnings*)
Set handling of exceptions.

ffc.quadrature.cpp.**set_float_formatting**(*precision*)
Set floating point formatting based on precision.

ffc.quadrature.deprecation module

exception ffc.quadrature.deprecation.**QuadratureRepresentationDeprecationWarning**
Bases: DeprecationWarning

Warning about deprecation of quadrature representation

ffc.quadrature.deprecation.**issue_deprecation_warning**()
Issue warning about quadrature deprecation

ffc.quadrature.expr module

This file implements a base class to represent an expression.

class ffc.quadrature.expr.**Expr**
Bases: object

expand()
Expand the expression. (FloatValue and Symbol are expanded by construction).

get_unique_vars(*var_type*)
Get unique variables (Symbols) as a set.

get_var_occurrences()
Determine the number of times all variables occurs in the expression. Returns a dictionary of variables and the number of times they occur. Works for FloatValue and Symbol.

ops()
Return number of operations to compute the expression. This is always zero for a FloatValue.

reduce_ops()
Reduce number of operations to evaluate the expression. There is nothing to be done for FloatValue and Symbol.

reduce_var(*var*)
Reduce the expression by another variable by using division. This works for FloatValue, Symbol and Product.

reduce_vartype(*var_type*)
Reduce expression with given var_type. It returns a tuple (found, remain), where 'found' is an expression that only has variables of type == var_type. If no variables are found, found=(). The 'remain' part contains the leftover after division by 'found' such that: self = found*remain. Works for FloatValue and Symbol.

t

val

ffc.quadrature.floatvalue module

This file implements a class to represent a float.

class ffc.quadrature.floatvalue.**FloatValue**(*value*)
Bases: *ffc.quadrature.expr.Expr*

ffc.quadrature.fraction module

This file implements a class to represent a fraction.

class ffc.quadrature.fraction.**Fraction**(*numerator*, *denominator*)
Bases: *ffc.quadrature.expr.Expr*

denom

expand()
Expand the fraction expression.

get_unique_vars(*var_type*)
Get unique variables (Symbols) as a set.

get_var_occurrences()
Determine the number of minimum number of times all variables occurs in the expression simply by calling the function on the numerator.

num

ops()
Return number of operations needed to evaluate fraction.

reduce_ops()
Reduce number of operations to evaluate the expression. There is nothing to be done for FloatValue and Symbol.

reduce_var(*var*)
Reduce the fraction by another variable through division of numerator.

reduce_vartype(*var_type*)
Reduce expression with given var_type. It returns a tuple (found, remain), where 'found' is an expression that only has variables of type == var_type. If no variables are found, found=(). The 'remain' part contains the leftover after division by 'found' such that: self = found*remain.

ffc.quadrature.optimisedquadraturetransformer module

QuadratureTransformer (optimised) for quadrature code generation to translate UFL expressions.

class ffc.quadrature.optimisedquadraturetransformer.**QuadratureTransformerOpt**(**args*)
Bases: *ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase*

Transform UFL representation to quadrature code.

abs(*o*, **operands*)

binary_condition(*o*, **operands*)

cell_coordinate(*o*)

cell_facet_jacobian(*o*)

cell_facet_jacobian_determinant(*o*)

cell_facet_jacobian_inverse(*o*)

cell_facet_origin(*o*)

cell_normal(*o*)

cell_orientation(*o*)

cell_origin(*o*)

cell_volume(*o*)

circumradius(*o*)

conditional(*o*, **operands*)

create_argument(*ufl_argument*, *derivatives*, *component*, *local_comp*, *local_offset*, *ffc_element*, *transformation*, *multiindices*, *tdim*, *gdim*, *avg*)
Create code for basis functions, and update relevant tables of used basis.

create_function(*ufl_function*, *derivatives*, *component*, *local_comp*, *local_offset*, *ffc_element*, *is_quad_element*, *transformation*, *multiindices*, *tdim*, *gdim*, *avg*)
Create code for basis functions, and update relevant tables of used basis.

division(*o*, **operands*)

facet_area(*o*)

facet_coordinate(*o*)

facet_jacobian(*o*)

facet_jacobian_determinant(*o*)

facet_jacobian_inverse(*o*)

facet_normal(*o*)

facet_origin(*o*)

jacobian(*o*)

jacobian_determinant(*o*)

jacobian_inverse(*o*)

max_facet_edge_length(*o*)

max_value(*o*, **operands*)

min_facet_edge_length(*o*)

min_value(*o*, **operands*)

not_condition(*o*, **operands*)

power(*o*)

product(*o*, **operands*)

quadrature_weight(*o*)

sum(*o*, **operands*)

ffc.quadrature.optimisedquadraturetransformer.**firstkey**(*d*)

ffc.quadrature.parameters module

Quadrature representation class for UFL

ffc.quadrature.parameters.**default_optimize_parameters**()

ffc.quadrature.parameters.**parse_optimise_parameters**(*parameters*, *itg_data*)

ffc.quadrature.product module

This file implements a class to represent a product.

class ffc.quadrature.product.**Product**(*variables*)
Bases: *ffc.quadrature.expr.Expr*

expand()
Expand all members of the product.

get_unique_vars(*var_type*)
Get unique variables (Symbols) as a set.

get_var_occurrences()
Determine the number of times all variables occurs in the expression. Returns a dictionary of variables and the number of times they occur.

get_vrs()
Return all 'real' variables.

ops()
Get the number of operations to compute product.

reduce_ops()
Reduce the number of operations to evaluate the product.

reduce_vartype(*var_type*)
Reduce expression with given var_type. It returns a tuple (found, remain), where 'found' is an expression that only has variables of type == var_type. If no variables are found, found=(). The 'remain' part contains the leftover after division by 'found' such that: self = found*remain.

vrs

ffc.quadrature.quadraturegenerator module

Code generator for quadrature representation.

ffc.quadrature.quadraturegenerator.**generate_integral_code**(*ir*, *prefix*, *parameters*)
Generate code for integral from intermediate representation.

ffc.quadrature.quadratureoptimization module

ffc.quadrature.quadratureoptimization.**optimize_integral_ir**(*ir*, *parameters*)
Compute optimized intermediate representation of integral.

ffc.quadrature.quadraturerepresentation module

Quadrature representation class for UFL

ffc.quadrature.quadraturerepresentation.**compute_integral_ir**(*itg_data*, *form_data*, *form_id*, *element_numbers*, *classnames*, *parameters*)
Compute intermediate represention of integral.

ffc.quadrature.quadraturerepresentation.**sort_integrals**(*integrals*, *default_scheme*, *default_degree*)
Sort and accumulate integrals according to the number of quadrature points needed per axis.

All integrals should be over the same (sub)domain.

ffc.quadrature.quadraturetransformer module

QuadratureTransformer for quadrature code generation to translate UFL expressions.

class ffc.quadrature.quadraturetransformer.**QuadratureTransformer**(**args*)
Bases: *ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase*

Transform UFL representation to quadrature code.

abs(*o*, **operands*)

binary_condition(*o*, **operands*)

cell_coordinate(*o*)

cell_facet_jacobian(*o*)

cell_facet_jacobian_determinant(*o*)

cell_facet_jacobian_inverse(*o*)

cell_facet_origin(*o*)

cell_normal(*o*)

cell_orientation(*o*)

cell_origin(*o*)

cell_volume(*o*)

circumradius(*o*)

conditional(*o*, **operands*)

create_argument(*ufl_argument*, *derivatives*, *component*, *local_comp*, *local_offset*, *ffc_element*, *transformation*, *multiindices*, *tdim*, *gdim*, *avg*)
Create code for basis functions, and update relevant tables of used basis.

create_function(*ufl_function*, *derivatives*, *component*, *local_comp*, *local_offset*, *ffc_element*, *is_quad_element*, *transformation*, *multiindices*, *tdim*, *gdim*, *avg*)
Create code for basis functions, and update relevant tables of used basis.

division(*o*, **operands*)

facet_area(*o*)

facet_coordinate(*o*)

facet_jacobian(*o*)

facet_jacobian_determinant(*o*)

facet_jacobian_inverse(*o*)

facet_normal(*o*)

facet_origin(*o*)

jacobian(*o*)

jacobian_determinant(*o*)

jacobian_inverse(*o*)

max_facet_edge_length(*o*)

max_value(*o*, **operands*)

min_facet_edge_length(*o*)

min_value(*o*, **operands*)

not_condition(*o*, **operands*)

power(*o*)

product(*o*, **operands*)

quadrature_weight(*o*)

sum(*o*, **operands*)

ffc.quadrature.quadraturetransformer.**firstkey**(*d*)

ffc.quadrature.quadraturetransformerbase module

QuadratureTransformerBase, a common class for quadrature transformers to translate UFL expressions.

class ffc.quadrature.quadraturetransformerbase.**FFCMultiIndex**(*dims*)

Bases: object

A MultiIndex represents a list of indices and holds the following data:

rank - rank of multiindex dims - a list of dimensions indices - a list of all possible multiindex values

class ffc.quadrature.quadraturetransformerbase.**QuadratureTransformerBase**(*psi_tables*, *quad_weights*, *gdim*, *tdim*, *entity_type*, *function_replace_map*, *optimise_parameters*)

Bases: ufl.algorithms.transformer.Transformer

Transform UFL representation to quadrature code.

abs(*o*, **operands*)

acos(*o*, **operands*)

argument(*o*)

asin(*o*, **operands*)

atan(*o*, **operands*)

atan_2(*o*, **operands*)

atan_2_function(*o*)

bessel_function(*o*)

bessel_i(*o*, **operands*)

bessel_j(*o*, **operands*)

bessel_k(*o*, **operands*)

bessel_y(*o*, **operands*)

cell_avg(*o*)

cell_coordinate(*o*)

cell_facet_jacobian(*o*)

cell_facet_jacobian_determinant(*o*)

cell_facet_jacobian_inverse(*o*)

cell_facet_origin(*o*)

cell_normal(*o*)

cell_orientation(*o*)

cell_origin(*o*)

cell_volume(*o*)

circumradius(*o*)

coefficient(*o*)

component()
Return current component tuple.

component_tensor(*o*)

compound_tensor_operator(*o*)

condition(*o*)

constant_value(*o*)

cos(*o*, **operands*)

cosh(*o*, **operands*)

derivative(*o*, **operands*)

derivatives()
Return all derivatives tuple.

disp()

division(*o*, **operands*)

erf(*o*, **operands*)

exp(*o*, **operands*)

expr(*o*)
Trigger error.

facet_area(*o*)

facet_avg(*o*)

facet_coordinate(*o*)

facet_jacobian(*o*)

facet_jacobian_determinant(*o*)

facet_jacobian_inverse(*o*)

facet_normal(*o*)

facet_origin(*o*)

generate_terms(*integrand*, *integral_type*)
Generate terms for code generation.

geometric_quantity(*o*)

grad(*o*)

identity(*o*)

index_sum(*o*)

indexed(*o*)

jacobian(*o*)

jacobian_determinant(*o*)

jacobian_inverse(*o*)

label(*o*)

list_tensor(*o*)

ln(*o*, **operands*)

math_function(*o*)

max_facet_edge_length(*o*)

min_facet_edge_length(*o*)

multi_index(*o*)

negative_restricted(*o*)

positive_restricted(*o*)

power(*o*)

product(*o*, **operands*)

quadrature_weight(*o*)

restricted(*o*)

scalar_value(*o*)
ScalarValue covers IntValue and FloatValue

sin(*o*, **operands*)

sinh(*o*, **operands*)

spatial_coordinate(*o*)

sqrt(*o*, **operands*)

sum(*o*, **operands*)

tan(*o*, **operands*)

tanh(*o*, **operands*)

terminal(*o*)
Always reuse Expr (ignore children)

update_cell()

update_facets(*facet0*, *facet1*)

update_points(*points*)

update_vertex(*vertex*)

variable(*o*)

zero(*o*)

ffc.quadrature.quadratureutils module

Utility functions for quadrature representation.

ffc.quadrature.quadratureutils.**contains_zeros**(*tables*)
Checks if any tables contains only zeros.

ffc.quadrature.quadratureutils.**create_permutations**(*expr*)

ffc.quadrature.quadratureutils.**create_psi_tables**(*tables*, *eliminate_zeros*, *entity_type*)
Create names and maps for tables and non-zero entries if appropriate.

ffc.quadrature.quadratureutils.**flatten_psi_tables**(*tables*, *entity_type*)
Create a 'flat' dictionary of tables with unique names and a name map that maps number of quadrature points and element name to a unique element number.

Input tables on the format for scalar and non-scalar elements respectively:
tables[num_points][element][entity][derivs][ip][dof] tables[num_points][element][entity][derivs][ip][component][dof]

Planning to change this into: tables[num_points][element][avg][entity][derivs][ip][dof] tables[num_points][element][avg][entity][derivs][ip][component][dof]

Returns: element_map - { num_quad_points: {ufl_element: element_number} }. flat_tables - { unique_table_name: values[ip,dof] }.

ffc.quadrature.quadratureutils.**get_ones**(*tables*)
Return names of tables for which all values are 1.0.

ffc.quadrature.quadratureutils.**unique_psi_tables**(*tables*, *eliminate_zeros*)
Returns a name map and a dictionary of unique tables. The function checks if values in the tables are equal, if this is the case it creates a name mapping. It also create additional information (depending on which parameters are set) such as if the table contains all ones, or only zeros, and a list on non-zero columns. unique_tables - {name:values,}. name_map - {original_name:[new_name, non-zero-columns (list), is zero (bool), is ones (bool)],}.

ffc.quadrature.quadratureutils.**unique_tables**(*tables*)
Removes tables with redundant values and returns a name_map and a inverse_name_map. E.g.,

tables = {a:[0,1,2], b:[0,2,3], c:[0,1,2], d:[0,1,2]} results in: tables = {a:[0,1,2], b:[0,2,3]} name_map = {a:[c,d]} inverse_name_map = {a:a, b:b, c:a, d:a}.

ffc.quadrature.reduce_operations module

Some simple functions for manipulating expressions symbolically

ffc.quadrature.reduce_operations.**expand_operations**(*expression*, *format*)
This function expands an expression and returns the value. E.g., ((x + y)) –> x + y 2*(x + y) –> 2*x + 2*y (x + y)*(x + y) –> x*x + y*y + 2*x*y z*(x*(y + 3) + 2) + 1 –> 1 + 2*z + x*y*z + x*z*3 z*((y + 3)*x + 2) + 1 –> 1 + 2*z + x*y*z + x*z*3

ffc.quadrature.reduce_operations.**get_constants**(*expression*, *const_terms*, *format*, *constants=[]*)
This function returns a new expression where all geometry terms have been substituted with geometry declarations, these declarations are added to the const_terms dictionary.

ffc.quadrature.reduce_operations.**get_geo_terms**(*expression*, *geo_terms*, *offset*, *format*)
This function returns a new expression where all geometry terms have been substituted with geometry declarations, these declarations are added to the geo_terms dictionary.

ffc.quadrature.reduce_operations.**get_indices**(*variable*, *format*, *from_get_indices=False*)
This function returns the indices of a given variable. E.g., P[0][j], returns ['j'] P[ip][k], returns ['ip','k'] P[ip][nzc0[j] + 3], returns ['ip','j'] w[0][j + 2] , returns [j]

ffc.quadrature.reduce_operations.**get_simple_variables**(*expression*, *format*)

This function takes as argument an expression (preferably expanded): expression = "x*x + y*x + x*y*z"

returns a list of products and a dictionary:

prods = ["x*x", "y*x", "x*y*z"] variables = {variable: [num_occurences, [pos_in_prods]]} variables = {"x":[3, [0,1,2]], "y":[2, [1,2]], "z":[1, [2]]}

ffc.quadrature.reduce_operations.**get_variables**(*expression*, *variables*, *format*, *constants=[]*)
This function returns a new expression where all geometry terms have been substituted with geometry declarations, these declarations are added to the const_terms dictionary.

ffc.quadrature.reduce_operations.**group_vars**(*expr*, *format*)
Group variables in an expression, such that: "x + y + z + 2*y + 6*z" = "x + 3*y + 7*z" "x*x + x*x + 2*x + 3*x + 5" = "2.0*x*x + 5.0*x + 5" "x*y + y*x + 2*x*y + 3*x + 0*x + 5" = "5.0*x*y + 3.0*x + 5" "(y + z)*x + 5*(y + z)*x" = "6.0*(y + z)*x" "1/(x*x) + 2*1/(x*x) + std::sqrt(x) + 6*std::sqrt(x)" = "3*1/(x*x) + 7*std::sqrt(x)"

ffc.quadrature.reduce_operations.**is_constant**(*variable*, *format*, *constants=[]*, *from_is_constant=False*)
Determine if a variable is constant or not. The function accepts an optional list of variables (loop indices) that will be regarded as constants for the given variable. If none are supplied it is assumed that all array accesses will result in a non-constant variable.

v = 2.0, is constant v = Jinv_00*det, is constant v = w[0][1], is constant v = 2*w[0][1], is constant v = W0[ip], is constant if constants = ['ip'] else not v = P_t0[ip][j], is constant if constants = ['j','ip'] else not

ffc.quadrature.reduce_operations.**operation_count**(*expression*, *format*)
This function returns the number of double operations in an expression. We do split () but not [] as we only have unsigned integer operations in [].

ffc.quadrature.reduce_operations.**reduce_operations**(*expression*, *format*)
This function reduces the number of opertions needed to compute a given expression. It looks for the variable that appears the most and groups terms containing this variable inside parenthesis. The function is called recursively until no further reductions are possible.

"x + y + x" = 2*x + y "x*x + 2.0*x*y + y*y" = y*y + (2.0*y + x)*x, not (x + y)*(x + y) as it should be!! z*x*y + z*x*3 + 2*z + 1" = z*(x*(y + 3) + 2) + 1

ffc.quadrature.reduce_operations.**reduction_possible**(*variables*)
Find the variable that occurs in the most products, if more variables occur the same number of times and in the same products add them to list.

ffc.quadrature.reduce_operations.**split_expression**(*expression*, *format*, *operator*, *allow_split=False*)
Split the expression at the given operator, return list. Do not split () or [] unless told to split (). This is to enable easy count of double operations which can be in (), but in [] we only have integer operations.

ffc.quadrature.sumobj module

This file implements a class to represent a sum.

class ffc.quadrature.sumobj.**Sum**(*variables*)
Bases: *ffc.quadrature.expr.Expr*

expand()
Expand all members of the sum.

get_unique_vars(*var_type*)
Get unique variables (Symbols) as a set.

get_var_occurrences()
Determine the number of minimum number of times all variables occurs in the expression. Returns a dictionary of variables and the number of times they occur. x*x + x returns {x:1}, x + y returns {}.

ops()
Return number of operations to compute value of sum.

reduce_ops()
Reduce the number of operations needed to evaluate the sum.

reduce_vartype(*var_type*)
Reduce expression with given var_type. It returns a list of tuples [(found, remain)], where 'found' is an expression that only has variables of type == var_type. If no variables are found, found=(). The 'remain' part contains the leftover after division by 'found' such that: self = Sum([f*r for f,r in self.reduce_vartype(Type)]).

vrs

ffc.quadrature.symbol module

This file implements a class to represent a symbol.

class ffc.quadrature.symbol.**Symbol**(*variable*, *symbol_type*, *base_expr=None*, *base_op=0*)
Bases: *ffc.quadrature.expr.Expr*

base_expr

base_op

cond

exp

get_unique_vars(*var_type*)
Get unique variables (Symbols) as a set.

get_var_occurrences()
Determine the number of times all variables occurs in the expression. Returns a dictionary of variables and the number of times they occur.

ops()
Returning the number of floating point operation for symbol.

v

ffc.quadrature.symbolics module

This file contains functions to optimise the code generated for quadrature representation.

ffc.quadrature.symbolics.**create_float**(*val*)

ffc.quadrature.symbolics.**create_fraction**(*num*, *denom*)

ffc.quadrature.symbolics.**create_product**(*variables*)

ffc.quadrature.symbolics.**create_sum**(*variables*)

ffc.quadrature.symbolics.**create_symbol**(*variable*, *symbol_type*, *base_expr=None*, *base_op=0*)

ffc.quadrature.symbolics.**generate_aux_constants**(*constant_decl*, *name*, *var_type*, *print_ops=False*)

A helper tool to generate code for constant declarations.

ffc.quadrature.symbolics.**optimise_code**(*expr*, *ip_consts*, *geo_consts*, *trans_set*)

Optimise a given expression with respect to, basis functions, integration points variables and geometric constants. The function will update the dictionaries ip_const and geo_consts with new declarations and update the trans_set (used transformations).

ffc.quadrature.tabulate_basis module

Quadrature representation class.

ffc.quadrature.tabulate_basis.**tabulate_basis**(*sorted_integrals*, *form_data*, *itg_data*)

Tabulate the basisfunctions and derivatives.

Module contents

ffc.tsfc package

Submodules

ffc.tsfc.tsfcgenerator module

ffc.tsfc.tsfcoptimization module

ffc.tsfc.tsfcoptimization.**optimize_integral_ir**(*ir*, *parameters*)

ffc.tsfc.tsfcrepresentation module

ffc.tsfc.tsfcrepresentation.**compute_integral_ir**(*integral_data*, *form_data*, *form_id*, *element_numbers*, *classnames*, *parameters*)

Compute intermediate represention of integral.

Module contents

ffc.uflacs package

Subpackages

ffc.uflacs.analysis package

Submodules

ffc.uflacs.analysis.balancing module

Algorithms for the representation phase of the form compilation.

class ffc.uflacs.analysis.balancing.**BalanceModifiers**
Bases: ufl.corealg.multifunction.MultiFunction

cell_avg(*expr*, **ops*)

expr(*expr*, **ops*)
Trigger error for types with missing handlers.

facet_avg(*expr*, **ops*)

grad(*expr*, **ops*)

negative_restricted(*expr*, **ops*)

positive_restricted(*expr*, **ops*)

reference_grad(*expr*, **ops*)

reference_value(*expr*, **ops*)

terminal(*expr*)

ffc.uflacs.analysis.balancing.**balance_modified_terminal**(*expr*)

ffc.uflacs.analysis.balancing.**balance_modifiers**(*expr*)

ffc.uflacs.analysis.crsarray module

Compressed row storage 'matrix' (actually just a non-rectangular 2d array).

class ffc.uflacs.analysis.crsarray.**CRSArray**(*row_capacity*, *element_capacity*, *dtype*)
Bases: object

An array of variable length dense arrays.

Stored efficiently with simple compressed row storage. This CRS array variant doesn't have a sparsity pattern, as each row is simply a dense vector.

Values are stored in one flat array 'data[]', and 'row_offsets[i]' contains the index to the first element on row i for 0<=i<=num_rows. There is no column index.

classmethod from_rows(*rows*, *num_rows*, *num_elements*, *dtype*)
Construct a CRSArray from a list of row element lists.

num_elements

push_row(*elements*)

ffc.uflacs.analysis.crsarray.**sufficient_int**(*maxval*)

ffc.uflacs.analysis.dependencies module

Tools for analysing dependencies within expression graphs.

ffc.uflacs.analysis.dependencies.**compute_dependencies**(*e2i*, *V*, *ignore_terminal_modifiers=True*)

ffc.uflacs.analysis.dependencies.**mark_active**(*dependencies*, *targets*)
Return an array marking the recursive dependencies of targets.

Input: - dependencies - CRSArray of ints, a mapping from a symbol to the symbols of its dependencies. - targets - Sequence of symbols to mark the dependencies of.

Output: - active - Truth value for each symbol. - num_used - Number of true values in active array.

ffc.uflacs.analysis.dependencies.**mark_image**(*inverse_dependencies*, *sources*)
Return an array marking the set of symbols dependent on the sources.

Input: - dependencies - CRSArray of ints, a mapping from a symbol to the symbols of its dependencies. - sources - Sequence of symbols to mark the dependants of.

Output: - image - Truth value for each symbol. - num_used - Number of true values in active array.

ffc.uflacs.analysis.expr_shapes module

Tools for computing various shapes of ufl expressions.

The total shape is the regular shape tuple plus the index shape tuple. The index shape tuple is the tuple of index dimensions of the free indices of the expression, sorted by the count of the free indices.

The total shape of a tensor valued expression `A` and `A[*indices(len(A.ufl_shape))]` is therefore the same.

ffc.uflacs.analysis.expr_shapes.**compute_all_shapes**(*v*)
Compute the tensor-, index-, and total shape of an expr.

Returns (shape, size, index_shape, index_size, total_shape, total_size).

ffc.uflacs.analysis.expr_shapes.**compute_index_shape**(*v*)
Compute the 'index shape' of v.

ffc.uflacs.analysis.expr_shapes.**total_shape**(*v*)
Compute the total shape of an expr.

ffc.uflacs.analysis.factorization module

Algorithms for factorizing argument dependent monomials.

class ffc.uflacs.analysis.factorization.**Factors**
Bases: object

add(*expr*)

ffc.uflacs.analysis.factorization.**add_to_fv**(*expr*, *FV*, *e2fi*)
Add expression expr to factor vector FV and expr->FVindex mapping e2fi.

ffc.uflacs.analysis.factorization.**build_argument_dependencies**(*dependencies*, *arg_indices*)
Preliminary algorithm: build list of argument vertex indices each vertex (indirectly) depends on.

ffc.uflacs.analysis.factorization.**build_argument_indices**(*V*)
Build ordered list of indices to modified arguments.

ffc.uflacs.analysis.factorization.**compute_argument_factorization**(*SV*, *SV_deps*, *SV_targets*, *rank*)
Factorizes a scalar expression graph w.r.t. scalar Argument components.

The result is a triplet (AV, FV, IM):

- The scalar argument component subgraph:

  AV[ai] = v

  with the property

  SV[arg_indices] == AV[:]

- An expression graph vertex list with all non-argument factors:

  FV[fi] = f

  with the property that none of the expressions depend on Arguments.

- A dict representation of the final integrand of rank r:

  IM = { (ai1_1, ..., ai1_r): fi1, (ai2_1, ..., ai2_r): fi2, }

  This mapping represents the factorization of SV[-1] w.r.t. Arguments s.t.:

  SV[-1] := sum(FV[fik] * product(AV[ai] for ai in aik) for aik, fik in IM.items())

  where := means equivalence in the mathematical sense, of course in a different technical representation.

ffc.uflacs.analysis.factorization.**handle_conditional**(*v*, *si*, *deps*, *SV_factors*, *FV*, *sv2fv*, *e2fi*)

ffc.uflacs.analysis.factorization.**handle_division**(*v*, *si*, *deps*, *SV_factors*, *FV*, *sv2fv*, *e2fi*)

ffc.uflacs.analysis.factorization.**handle_operator**(*v*, *si*, *deps*, *SV_factors*, *FV*, *sv2fv*, *e2fi*)

ffc.uflacs.analysis.factorization.**handle_product**(*v*, *si*, *deps*, *SV_factors*, *FV*, *sv2fv*, *e2fi*)

ffc.uflacs.analysis.factorization.**handle_sum**(*v*, *si*, *deps*, *SV_factors*, *FV*, *sv2fv*, *e2fi*)

ffc.uflacs.analysis.factorization.**rebuild_scalar_graph_from_factorization**(*AV*, *FV*, *IM*)

ffc.uflacs.analysis.graph module

Linearized data structure for the computational graph.

class ffc.uflacs.analysis.graph.**Graph2**
Bases: object

ffc.uflacs.analysis.graph.**build_graph**(*expressions*, *DEBUG=False*)

ffc.uflacs.analysis.graph_rebuild module

Rebuilding UFL expressions from linearized representation of computational graph.

class ffc.uflacs.analysis.graph_rebuild.**ReconstructScalarSubexpressions**
Bases: ufl.corealg.multifunction.MultiFunction

abs(*o*, *ops*)

atan_2(*o*, *ops*)

bessel_function(*o*, *ops*)

component_tensor(*o*, **args*, ***kwargs*)

condition(*o*, *ops*)

conditional(*o*, *ops*)

division(*o*, *ops*)

expr(*o*, **args*, ***kwargs*)
Trigger error for types with missing handlers.

expr_list(*o*, **args*, ***kwargs*)

expr_mapping(*o*, **args*, ***kwargs*)

index_sum(*o*, *ops*)

label(*o*, **args*, ***kwargs*)

list_tensor(*o*, **args*, ***kwargs*)

math_function(*o*, *ops*)

max_value(*o*, *ops*)

min_value(*o*, *ops*)

multi_index(*o*, **args*, ***kwargs*)

power(*o*, *ops*)

product(*o*, *ops*)

scalar_nary(*o*, *ops*)

sum(*o*, *ops*)

terminal(*o*)

transposed(*o*, **args*, ***kwargs*)

unexpected(*o*, **args*, ***kwargs*)

utility_type(*o*, **args*, ***kwargs*)

variable(*o*, **args*, ***kwargs*)

ffc.uflacs.analysis.graph_rebuild.**rebuild_expression_from_graph**(*G*)
This is currently only used by tests.

ffc.uflacs.analysis.graph_rebuild.**rebuild_with_scalar_subexpressions**(*G*, *targets=None*)
Build a new expression2index mapping where each subexpression is scalar valued.

Input: - G.e2i - G.V - G.V_symbols - G.total_unique_symbols

Output: - NV - Array with reverse mapping from index to expression - nvs - Tuple of ne2i indices corresponding to the last vertex of G.V

Old output now no longer returned but possible to restore if needed: - ne2i - Mapping from scalar subexpressions to a contiguous unique index - W - Array with reconstructed scalar subexpressions for each original symbol

ffc.uflacs.analysis.graph_ssa module

Algorithms for working with computational graphs.

ffc.uflacs.analysis.graph_ssa.**allocate_registers**(*active*, *partitions*, *targets*, *scores*, *max_registers*, *score_threshold*)

FIXME: Cover with tests.

TODO: Allow reuse of registers, reducing memory usage.

TODO: Probably want to sort within partitions.

ffc.uflacs.analysis.graph_ssa.**compute_cache_scores**(*V*, *active*, *dependencies*, *inverse_dependencies*, *partitions*, *cache_score_policy=<function default_cache_score_policy>*)

FIXME: Cover with tests.

TODO: Experiment with heuristics later when we have functional code generation.

ffc.uflacs.analysis.graph_ssa.**compute_dependency_count**(*dependencies*)

FIXME: Test

ffc.uflacs.analysis.graph_ssa.**default_cache_score_policy**(*vtype*, *ndeps*, *ninvdeps*, *partition*)

ffc.uflacs.analysis.graph_ssa.**default_partition_seed**(*expr*, *rank*)

Partition 0: Piecewise constant on each cell (including Real and DG0 coefficients) Partition 1: Depends on x Partition 2: Depends on x and coefficients Partitions [3,3+rank): depend on argument with count partition-3

ffc.uflacs.analysis.graph_ssa.**invert_dependencies**(*dependencies*, *depcount*)

FIXME: Test

ffc.uflacs.analysis.graph_ssa.**mark_partitions**(*V*, *active*, *dependencies*, *rank*, *partition_seed=<function default_partition_seed>*, *partition_combiner=<built-in function max>*)

FIXME: Cover this with tests.

Input: - V - Array of expressions. - active - Boolish array. - dependencies - CRSArray with V dependencies. - partition_seed - Policy for determining the partition of a terminalish. - partition_combiner - Policy for determinging the partition of an operator.

Output: - partitions - Array of partition int ids.

ffc.uflacs.analysis.graph_symbols module

Assigning symbols to computational graph nodes.

ffc.uflacs.analysis.graph_symbols.**build_graph_symbols**(*V*, *e2i*, *DEBUG*)

Tabulate scalar value numbering of all nodes in a a list based representation of an expression graph.

Returns: V_shapes - CRSArray of the total shapes of nodes in V. V_symbols - CRSArray of symbols (value numbers) of each component of each node in V. total_unique_symbols - The number of symbol values assigned to unique scalar components of the nodes in V.

ffc.uflacs.analysis.graph_symbols.**build_node_shapes**(*V*)

Build total shapes for each node in list representation of expression graph.

V is an array of ufl expressions, possibly nonscalar and with free indices.

Returning a CRSArray where row i is the total shape of V[i].

ffc.uflacs.analysis.graph_symbols.**build_node_sizes**(*V_shapes*)

Compute all the products of a sequence of shapes.

ffc.uflacs.analysis.graph_symbols.**build_node_symbols**(*V*, *e2i*, *V_shapes*, *V_sizes*)
Tabulate scalar value numbering of all nodes in a a list based representation of an expression graph.

Returns: V_symbols - CRSArray of symbols (value numbers) of each component of each node in V. total_unique_symbols - The number of symbol values assigned to unique scalar components of the nodes in V.

ffc.uflacs.analysis.graph_vertices module

Algorithms for working with graphs.

ffc.uflacs.analysis.graph_vertices.**build_array_from_counts**(*e2i*)

ffc.uflacs.analysis.graph_vertices.**build_graph_vertices**(*expressions*)

ffc.uflacs.analysis.graph_vertices.**build_node_counts**(*expressions*)

ffc.uflacs.analysis.graph_vertices.**build_scalar_graph_vertices**(*expressions*)

ffc.uflacs.analysis.graph_vertices.**build_scalar_node_counts**(*expressions*)

ffc.uflacs.analysis.graph_vertices.**count_nodes_with_unique_post_traversal**(*expr*, *e2i=None*, *skip_terminal_modifiers*

Yields o for each node o in expr, child before parent. Never visits a node twice.

ffc.uflacs.analysis.indexing module

Algorithms for working with multiindices.

ffc.uflacs.analysis.indexing.**map_component_tensor_arg_components**(*tensor*)
Build integer list mapping between flattended components of tensor and its underlying indexed subexpression.

ffc.uflacs.analysis.indexing.**map_indexed_arg_components**(*indexed*)
Build integer list mapping between flattended components of indexed expression and its underlying tensor-valued subexpression.

ffc.uflacs.analysis.modified_terminals module

Definitions of 'modified terminals', a core concept in uflacs.

class ffc.uflacs.analysis.modified_terminals.**ModifiedTerminal**(*expr*, *terminal*, *reference_value*, *base_shape*, *base_symmetry*, *component*, *flat_component*, *global_derivatives*, *local_derivatives*, *averaged*, *restriction*)

Bases: object

A modified terminal expression is an object of a Terminal subtype, wrapped in terminal modifier types.

The variables of this class are:

expr - The original UFL expression terminal - the underlying Terminal object

global_derivatives - tuple of ints, each meaning derivative in that global direction local_derivatives - tuple of ints, each meaning derivative in that local direction reference_value - bool, whether this is represented in reference frame averaged - None, 'facet' or 'cell' restriction - None, '+' or '-'

component - tuple of ints, the global component of the Terminal flat_component - single int, flattened local component of the Terminal, considering symmetry

Possibly other component model: - global_component - reference_component - flat_component

argument_ordering_key()
Return a key for deterministic sorting of argument vertex indices based on the properties of the modified terminal. Used in factorization but moved here for closeness with ModifiedTerminal attributes.

as_tuple()
Return a tuple with hashable values that uniquely identifies this modified terminal.

Some of the derived variables can be omitted here as long as they are fully determined from the variables that are included here.

ffc.uflacs.analysis.modified_terminals.**analyse_modified_terminal**(*expr*)
Analyse a so-called 'modified terminal' expression.

Return its properties in more compact form as a ModifiedTerminal object.

A modified terminal expression is an object of a Terminal subtype, wrapped in terminal modifier types.

The wrapper types can include 0-* Grad or ReferenceGrad objects, and 0-1 ReferenceValue, 0-1 Restricted, 0-1 Indexed, and 0-1 FacetAvg or CellAvg objects.

ffc.uflacs.analysis.modified_terminals.**is_modified_terminal**(*v*)
Check if v is a terminal or a terminal wrapped in terminal modifier types.

ffc.uflacs.analysis.modified_terminals.**strip_modified_terminal**(*v*)
Extract core Terminal from a modified terminal or return None.

ffc.uflacs.analysis.valuenumbering module

Algorithms for value numbering within computational graphs.

class ffc.uflacs.analysis.valuenumbering.**ValueNumberer**(*e2i*, *V_sizes*, *V_symbols*)
Bases: ufl.corealg.multifunction.MultiFunction

An algorithm to map the scalar components of an expression node to unique value numbers, with fallthrough for types that can be mapped to the value numbers of their operands.

cell_avg(*v*, *i*)
Modifiers: terminal - the underlying Terminal object global_derivatives - tuple of ints, each meaning derivative in that global direction local_derivatives - tuple of ints, each meaning derivative in that local direction reference_value - bool, whether this is represented in reference frame averaged - None, 'facet' or 'cell' restriction - None, '+' or '-' component - tuple of ints, the global component of the Terminal flat_component - single int, flattened local component of the Terminal, considering symmetry

component_tensor(*A*, *i*)

expr(*v*, *i*)
Create new symbols for expressions that represent new values.

facet_avg(*v*, *i*)
Modifiers: terminal - the underlying Terminal object global_derivatives - tuple of ints, each meaning derivative in that global direction local_derivatives - tuple of ints, each meaning derivative in that local

direction reference_value - bool, whether this is represented in reference frame averaged - None, 'facet' or 'cell' restriction - None, '+' or '-' component - tuple of ints, the global component of the Terminal flat_component - single int, flattened local component of the Terminal, considering symmetry

form_argument(*v*, *i*)
Create new symbols for expressions that represent new values.

get_node_symbols(*expr*)

grad(*v*, *i*)
Modifiers: terminal - the underlying Terminal object global_derivatives - tuple of ints, each meaning derivative in that global direction local_derivatives - tuple of ints, each meaning derivative in that local direction reference_value - bool, whether this is represented in reference frame averaged - None, 'facet' or 'cell' restriction - None, '+' or '-' component - tuple of ints, the global component of the Terminal flat_component - single int, flattened local component of the Terminal, considering symmetry

indexed(*Aii*, *i*)

list_tensor(*v*, *i*)

new_symbol()
Generator for new symbols with a running counter.

new_symbols(*n*)
Generator for new symbols with a running counter.

reference_grad(*v*, *i*)
Modifiers: terminal - the underlying Terminal object global_derivatives - tuple of ints, each meaning derivative in that global direction local_derivatives - tuple of ints, each meaning derivative in that local direction reference_value - bool, whether this is represented in reference frame averaged - None, 'facet' or 'cell' restriction - None, '+' or '-' component - tuple of ints, the global component of the Terminal flat_component - single int, flattened local component of the Terminal, considering symmetry

reference_value(*v*, *i*)
Modifiers: terminal - the underlying Terminal object global_derivatives - tuple of ints, each meaning derivative in that global direction local_derivatives - tuple of ints, each meaning derivative in that local direction reference_value - bool, whether this is represented in reference frame averaged - None, 'facet' or 'cell' restriction - None, '+' or '-' component - tuple of ints, the global component of the Terminal flat_component - single int, flattened local component of the Terminal, considering symmetry

restricted(*v*, *i*)
Modifiers: terminal - the underlying Terminal object global_derivatives - tuple of ints, each meaning derivative in that global direction local_derivatives - tuple of ints, each meaning derivative in that local direction reference_value - bool, whether this is represented in reference frame averaged - None, 'facet' or 'cell' restriction - None, '+' or '-' component - tuple of ints, the global component of the Terminal flat_component - single int, flattened local component of the Terminal, considering symmetry

variable(*v*, *i*)
Direct reuse of all symbols.

Module contents

Algorithms for the analysis phase of the form compilation.

ffc.uflacs.backends package

Subpackages

ffc.uflacs.backends.ffc package

Submodules

ffc.uflacs.backends.ffc.access module

FFC/UFC specific variable access.

class ffc.uflacs.backends.ffc.access.**FFCBackendAccess**(*ir*, *language*, *symbols*, *parameters*)

Bases: ufl.corealg.multifunction.MultiFunction

FFC specific cpp formatter class.

cell_coordinate(*e*, *mt*, *tabledata*, *num_points*)

cell_edge_vectors(*e*, *mt*, *tabledata*, *num_points*)

cell_facet_jacobian(*e*, *mt*, *tabledata*, *num_points*)

cell_normal(*e*, *mt*, *tabledata*, *num_points*)

cell_orientation(*e*, *mt*, *tabledata*, *num_points*)

cell_vertices(*e*, *mt*, *tabledata*, *num_points*)

coefficient(*e*, *mt*, *tabledata*, *num_points*)

expr(*e*, *mt*, *tabledata*, *num_points*)
Trigger error for types with missing handlers.

facet_coordinate(*e*, *mt*, *tabledata*, *num_points*)

facet_edge_vectors(*e*, *mt*, *tabledata*, *num_points*)

facet_jacobian(*e*, *mt*, *tabledata*, *num_points*)

facet_jacobian_determinant(*e*, *mt*, *tabledata*, *num_points*)

facet_jacobian_inverse(*e*, *mt*, *tabledata*, *num_points*)

facet_normal(*e*, *mt*, *tabledata*, *num_points*)

facet_orientation(*e*, *mt*, *tabledata*, *num_points*)

float_value(*e*, *mt*, *tabledata*, *num_points*)

int_value(*e*, *mt*, *tabledata*, *num_points*)

jacobian(*e*, *mt*, *tabledata*, *num_points*)

jacobian_determinant(*e*, *mt*, *tabledata*, *num_points*)

jacobian_inverse(*e*, *mt*, *tabledata*, *num_points*)

reference_cell_edge_vectors(*e*, *mt*, *tabledata*, *num_points*)

reference_cell_volume(*e*, *mt*, *tabledata*, *access*)

reference_facet_edge_vectors(*e*, *mt*, *tabledata*, *num_points*)

reference_facet_volume(*e*, *mt*, *tabledata*, *access*)

reference_normal(*e*, *mt*, *tabledata*, *access*)

spatial_coordinate(*e*, *mt*, *tabledata*, *num_points*)

zero(*e*, *mt*, *tabledata*, *num_points*)

ffc.uflacs.backends.ffc.backend module

Collection of FFC specific pieces for the code generation phase.

class ffc.uflacs.backends.ffc.backend.**FFCBackend**(*ir*, *parameters*)
Bases: object

Class collecting all aspects of the FFC backend.

ffc.uflacs.backends.ffc.common module

FFC/UFC specific symbol naming.

ffc.uflacs.backends.ffc.common.**num_coordinate_component_dofs**(*coordinate_element*)
Get the number of dofs for a coordinate component for this degree.

This is a local hack that works for Lagrange 1-3, better would be to get this passed by ffc from fiat through the ir. The table data is to messy to figure out a clean design for that quickly.

ffc.uflacs.backends.ffc.common.**ufc_restriction_offset**(*restriction*, *length*)

ffc.uflacs.backends.ffc.definitions module

FFC/UFC specific variable definitions.

class ffc.uflacs.backends.ffc.definitions.**FFCBackendDefinitions**(*ir*, *language*, *symbols*, *parameters*)
Bases: ufl.corealg.multifunction.MultiFunction

FFC specific code definitions.

argument(*t*, *mt*, *tabledata*, *num_points*, *access*)
Arguments are accessed through element tables.

cell_coordinate(*e*, *mt*, *tabledata*, *num_points*, *access*)
Return definition code for the reference spatial coordinates.

If reference coordinates are given:

```
No definition needed.
```

If physical coordinates are given and domain is affine:

```
X = K*(x-x0)
```

This is inserted symbolically.

If physical coordinates are given and domain is non- affine:

```
Not currently supported.
```

cell_edge_vectors(*e*, *mt*, *tabledata*, *num_points*, *access*)
These quantities refer to coordinate_dofs

cell_facet_jacobian(*e*, *mt*, *tabledata*, *num_points*, *access*)
These quantities refer to constant tables defined in ufc_geometry.h.

cell_normal(*e*, *mt*, *tabledata*, *num_points*, *access*)
These quantities are expected to be replaced in symbolic preprocessing.

cell_orientation(*e*, *mt*, *tabledata*, *num_points*, *access*)

cell_vertices(*e*, *mt*, *tabledata*, *num_points*, *access*)
These quantities refer to coordinate_dofs

coefficient(*t*, *mt*, *tabledata*, *num_points*, *access*)
Return definition code for coefficients.

constant_value(*e*, *mt*, *tabledata*, *num_points*, *access*)
Constants simply use literals in the target language.

expr(*t*, *mt*, *tabledata*, *num_points*, *access*)
Trigger error for types with missing handlers.

facet_edge_vectors(*e*, *mt*, *tabledata*, *num_points*, *access*)
These quantities refer to coordinate_dofs

facet_jacobian(*e*, *mt*, *tabledata*, *num_points*, *access*)
These quantities are expected to be replaced in symbolic preprocessing.

facet_jacobian_determinant(*e*, *mt*, *tabledata*, *num_points*, *access*)
These quantities are expected to be replaced in symbolic preprocessing.

facet_jacobian_inverse(*e*, *mt*, *tabledata*, *num_points*, *access*)
These quantities are expected to be replaced in symbolic preprocessing.

facet_normal(*e*, *mt*, *tabledata*, *num_points*, *access*)
These quantities are expected to be replaced in symbolic preprocessing.

facet_orientation(*e*, *mt*, *tabledata*, *num_points*, *access*)
These quantities refer to constant tables defined in ufc_geometry.h.

jacobian(*e*, *mt*, *tabledata*, *num_points*, *access*)
Return definition code for the Jacobian of x(X).

J = sum_k xdof_k grad_X xphi_k(X)

jacobian_determinant(*e*, *mt*, *tabledata*, *num_points*, *access*)
These quantities are expected to be replaced in symbolic preprocessing.

jacobian_inverse(*e*, *mt*, *tabledata*, *num_points*, *access*)
These quantities are expected to be replaced in symbolic preprocessing.

reference_cell_edge_vectors(*e*, *mt*, *tabledata*, *num_points*, *access*)
These quantities refer to constant tables defined in ufc_geometry.h.

reference_cell_volume(*e*, *mt*, *tabledata*, *num_points*, *access*)
These quantities refer to constant tables defined in ufc_geometry.h.

reference_facet_edge_vectors(*e*, *mt*, *tabledata*, *num_points*, *access*)
These quantities refer to constant tables defined in ufc_geometry.h.

reference_facet_volume(*e*, *mt*, *tabledata*, *num_points*, *access*)
These quantities refer to constant tables defined in ufc_geometry.h.

reference_normal(*e*, *mt*, *tabledata*, *num_points*, *access*)
These quantities refer to constant tables defined in ufc_geometry.h.

spatial_coordinate(*e*, *mt*, *tabledata*, *num_points*, *access*)
Return definition code for the physical spatial coordinates.

If physical coordinates are given: No definition needed.

If reference coordinates are given: x = sum_k xdof_k xphi_k(X)

If reference facet coordinates are given: x = sum_k xdof_k xphi_k(Xf)

ffc.uflacs.backends.ffc.symbols module

FFC/UFC specific symbol naming.

class ffc.uflacs.backends.ffc.symbols.**FFCBackendSymbols**(*language*, *coefficient_numbering*)

Bases: object

FFC specific symbol definitions. Provides non-ufl symbols.

J_component(*mt*)
Jacobian component.

X_component(*mt*)
Reference coordinate component.

argument_loop_index(*iarg*)
Loop index for argument #iarg.

cell_orientation_argument(*restriction*)
Cell orientation argument in ufc. Not same as cell orientation in generated code.

cell_orientation_internal(*restriction*)
Internal value for cell orientation in generated code.

coefficient_dof_access(*coefficient*, *dof_number*)

coefficient_dof_sum_index()
Index for loops over coefficient dofs, assumed to never be used in two nested loops.

coefficient_value(*mt*)
Symbol for variable holding value or derivative component of coefficient.

custom_points_table()
Table for chunk of custom quadrature points (physical coordinates).

custom_quadrature_points()
Physical quadrature points, argument to custom integrals.

custom_quadrature_weights()
Quadrature weights including cell measure scaling, argument to custom integrals.

custom_weights_table()
Table for chunk of custom quadrature weights (including cell measure scaling).

domain_dof_access(*dof*, *component*, *gdim*, *num_scalar_dofs*, *restriction*)

domain_dofs_access(*gdim*, *num_scalar_dofs*, *restriction*)

element_table(*tabledata*, *entitytype*, *restriction*)

element_tensor()
Symbol for the element tensor itself.

entity(*entitytype*, *restriction*)
Entity index for lookup in element tables.

num_custom_quadrature_points()
Number of quadrature points, argument to custom integrals.

points_table(*num_points*)
Table of quadrature points (points on the reference integration entity).

quadrature_loop_index()
Reusing a single index name for all quadrature loops, assumed not to be nested.

weights_table(*num_points*)
Table of quadrature weights.

x_component(*mt*)
Physical coordinate component.

ffc.uflacs.backends.ffc.symbols.**format_mt_name**(*basename*, *mt*)
Format variable name for modified terminal.

ffc.uflacs.backends.ffc.symbols.**ufc_restriction_postfix**(*restriction*)

Module contents

The FFC specific backend to the UFLACS form compiler algorithms.

ffc.uflacs.backends.ufc package

Submodules

ffc.uflacs.backends.ufc.apply_mappings module

ffc.uflacs.backends.ufc.coordinate_mapping module

ffc.uflacs.backends.ufc.coordinate_mapping.**adj_expr_2x2**(*A*)

ffc.uflacs.backends.ufc.coordinate_mapping.**adj_expr_3x3**(*A*)

ffc.uflacs.backends.ufc.coordinate_mapping.**codet_nn**(*A*, *rows*, *cols*)

ffc.uflacs.backends.ufc.coordinate_mapping.**cross_expr**(*a*, *b*)

ffc.uflacs.backends.ufc.coordinate_mapping.**det_22**(*B*, *i*, *j*, *k*, *l*)

ffc.uflacs.backends.ufc.coordinate_mapping.**det_nn**(*A*, *n*)

ffc.uflacs.backends.ufc.coordinate_mapping.**generate_assign_inverse**(*L*, *K*, *J*, *detJ*, *gdim*, *tdim*)

ffc.uflacs.backends.ufc.coordinate_mapping.**generate_compute_ATA**(*L*, *ATA*, *A*, *m*, *n*, *index_prefix=''*)
Generate code to declare and compute ATA[i,j] = sum_k A[k,i]*A[k,j] with given A shaped (m,n).

ffc.uflacs.backends.ufc.coordinate_mapping.**generate_cross_decl**(*c*, *a*, *b*)

ffc.uflacs.backends.ufc.coordinate_mapping.**pdet_m1**(*L*, *A*, *m*)

class ffc.uflacs.backends.ufc.coordinate_mapping.**ufc_coordinate_mapping**
Bases: *ffc.uflacs.backends.ufc.generator.ufc_generator*

Each function maps to a keyword in the template. See documentation of ufc_generator.

cell_shape(*L*, *ir*)

compute_geometry(*L*, *ir*)

compute_jacobian_determinants(*L*, *ir*)

compute_jacobian_inverses(*L*, *ir*)

compute_jacobians(*L*, *ir*)

compute_midpoint_geometry(*L*, *ir*)

compute_physical_coordinates(*L*, *ir*)

compute_reference_coordinates(*L*, *ir*)

compute_reference_geometry(*L*, *ir*)

create_coordinate_dofmap(*L*, *ir*)

create_coordinate_finite_element(*L*, *ir*)

geometric_dimension(*L*, *geometric_dimension*)

topological_dimension(*L*, *topological_dimension*)

ffc.uflacs.backends.ufc.dofmap module

class ffc.uflacs.backends.ufc.dofmap.**ufc_dofmap**
Bases: *ffc.uflacs.backends.ufc.generator.ufc_generator*

Each function maps to a keyword in the template. See documentation of ufc_generator.

create_sub_dofmap(*L*, *ir*)

global_dimension(*L*, *ir*)

needs_mesh_entities(*L*, *needs_mesh_entities*)
needs_mesh_entities is a list of num dofs per entity.

num_element_dofs(*L*, *num_element_dofs*)

num_element_support_dofs(*L*, *num_element_support_dofs*)

num_entity_closure_dofs(*L*, *num_entity_closure_dofs*)

num_entity_dofs(*L*, *num_entity_dofs*)

num_facet_dofs(*L*, *num_facet_dofs*)

num_global_support_dofs(*L*, *num_global_support_dofs*)

num_sub_dofmaps(*L*, *num_sub_dofmaps*)

tabulate_dofs(*L*, *ir*)

tabulate_entity_closure_dofs(*L*, *ir*)

tabulate_entity_dofs(*L*, *ir*)

tabulate_facet_dofs(*L*, *ir*)

topological_dimension(*L*, *topological_dimension*)

ffc.uflacs.backends.ufc.evalderivs module

Work in progress translation of FFC evaluatebasis code to uflacs CNodes format.

ffc.uflacs.backends.ufc.evalderivs.**generate_evaluate_reference_basis_derivatives**(*L*, *data*, *parameters*)

ffc.uflacs.backends.ufc.evalderivs.**generate_tabulate_dmats**(*L*, *dofs_data*)
Tabulate the derivatives of the polynomial base

ffc.uflacs.backends.ufc.evaluatebasis module

Work in progress translation of FFC evaluatebasis code to uflacs CNodes format.

ffc.uflacs.backends.ufc.evaluatebasis.**generate_compute_basisvalues**(*L*, *dofs_data*, *element_cellname*, *tdim*, *X*, *ip*)

ffc.uflacs.backends.ufc.evaluatebasis.**generate_evaluate_reference_basis**(*L*, *data*, *parameters*)
Generate code to evaluate element basisfunctions at an arbitrary point on the reference element.

The value(s) of the basisfunction is/are computed as in FIAT as the dot product of the coefficients (computed at compile time) and basisvalues which are dependent on the coordinate and thus have to be computed at run time.

The function should work for all elements supported by FIAT, but it remains untested for tensor valued elements.

This code is adapted from code in FFC which computed the basis from physical coordinates, and also to use UFLACS utilities.

The FFC code has a comment "From FIAT_NEW.polynomial_set.tabulate()".

ffc.uflacs.backends.ufc.evaluatebasis.**generate_expansion_coefficients**(*L*, *dofs_data*)

ffc.uflacs.backends.ufc.evaluatebasis.**tabulate_coefficients**(*L*, *dof_data*)
This function tabulates the element coefficients that are generated by FIAT at compile time.

ffc.uflacs.backends.ufc.evaluatebasisderivatives module

Work in progress translation of FFC evaluatebasisderivatives code to uflacs CNodes format.

ffc.uflacs.backends.ufc.evaluatebasisderivatives.**generate_evaluate_basis_derivatives**(*L*, *data*)
Evaluate the derivatives of an element basisfunction at a point. The values are computed as in FIAT as the matrix

product of the coefficients (computed at compile time), basisvalues which are dependent on the coordinate and thus have to be computed at run time and combinations (depending on the order of derivative) of dmats tables which hold the derivatives of the expansion coefficients.

ffc.uflacs.backends.ufc.evaluatebasisderivatives.**generate_evaluate_basis_derivatives_all**(*L*, *da*

Like evaluate_basis, but return the values of all basis functions (dofs).

ffc.uflacs.backends.ufc.evaluatedof module

ffc.uflacs.backends.ufc.evaluatedof.**generate_evaluate_dof**(*L*, *ir*)
Generate code for evaluate_dof.

ffc.uflacs.backends.ufc.evaluatedof.**generate_evaluate_dofs**(*L*, *ir*)
Generate code for evaluate_dofs.

ffc.uflacs.backends.ufc.evaluatedof.**reference_to_physical_map**(*cellname*)
Returns a map from reference coordinates to physical element coordinates

ffc.uflacs.backends.ufc.finite_element module

ffc.uflacs.backends.ufc.finite_element.**compute_basis_values**(*L*, *data*, *dof_data*)

ffc.uflacs.backends.ufc.finite_element.**compute_values**(*L*, *data*, *dof_data*)
This function computes the value of the basisfunction as the dot product of the coefficients and basisvalues.

ffc.uflacs.backends.ufc.finite_element.**generate_element_mapping**(*mapping*, *i*, *num_reference_components*, *tdim*, *gdim*, *J*, *detJ*, *K*)

class ffc.uflacs.backends.ufc.finite_element.**ufc_finite_element**
Bases: *ffc.uflacs.backends.ufc.generator.ufc_generator*

Each function maps to a keyword in the template. See documentation of ufc_generator.

cell_shape(*L*, *cell_shape*)

create_sub_element(*L*, *ir*)

degree(*L*, *degree*)

evaluate_basis(*L*, *ir*, *parameters*)

evaluate_basis_all(*L*, *ir*, *parameters*)

evaluate_basis_derivatives(*L*, *ir*, *parameters*)

evaluate_basis_derivatives_all(*L*, *ir*, *parameters*)

evaluate_dof(*L*, *ir*, *parameters*)

evaluate_dofs(*L*, *ir*, *parameters*)
Generate code for evaluate_dofs.

evaluate_reference_basis(*L*, *ir*, *parameters*)

evaluate_reference_basis_derivatives(*L*, *ir*, *parameters*)

family(*L*, *family*)

geometric_dimension(*L*, *geometric_dimension*)

interpolate_vertex_values(*L*, *ir*, *parameters*)

num_sub_elements(*L*, *num_sub_elements*)

reference_value_dimension(*L*, *reference_value_shape*)

reference_value_rank(*L*, *reference_value_shape*)

reference_value_size(*L*, *reference_value_shape*)

space_dimension(*L*, *space_dimension*)

tabulate_dof_coordinates(*L*, *ir*, *parameters*)

tabulate_reference_dof_coordinates(*L*, *ir*, *parameters*)

topological_dimension(*L*, *topological_dimension*)

transform_reference_basis_derivatives(*L*, *ir*, *parameters*)

value_dimension(*L*, *value_shape*)

value_rank(*L*, *value_shape*)

value_size(*L*, *value_shape*)

ffc.uflacs.backends.ufc.form module

ffc.uflacs.backends.ufc.form.**add_ufc_form_integral_methods**(*cls*)

This function generates methods on the class it decorates, for each integral name template and for each integral type.

This allows implementing e.g. create_###_integrals once in the decorated class as '_create_foo_integrals', and this function will expand that implementation into 'create_cell_integrals', 'create_exterior_facet_integrals', etc.

Name templates are taken from 'integral_name_templates' and 'ufc_integral_types'.

ffc.uflacs.backends.ufc.form.**create_delegate**(*integral_type*, *declname*, *impl*)

class ffc.uflacs.backends.ufc.form.**ufc_form**

Bases: *ffc.uflacs.backends.ufc.generator.ufc_generator*

Each function maps to a keyword in the template. See documentation of ufc_generator.

The exceptions are functions on the form def _*_foo_*(self, L, ir, parameters, integral_type, declname)

which add_ufc_form_integral_methods will duplicate for foo = each integral type.

create_cell_integral(*L*, *ir*, *parameters*)

Return implementation of ufc::form::create_cell_integral().

create_coordinate_dofmap(*L*, *ir*)

create_coordinate_finite_element(*L*, *ir*)

create_coordinate_mapping(*L*, *ir*)

create_custom_integral(*L*, *ir*, *parameters*)

Return implementation of ufc::form::create_custom_integral().

create_cutcell_integral(*L*, *ir*, *parameters*)

Return implementation of ufc::form::create_cutcell_integral().

create_default_cell_integral(*L*, *ir*, *parameters*)

Return implementation of ufc::form::create_default_cell_integral().

create_default_custom_integral(*L*, *ir*, *parameters*)
Return implementation of ufc::form::create_default_custom_integral().

create_default_cutcell_integral(*L*, *ir*, *parameters*)
Return implementation of ufc::form::create_default_cutcell_integral().

create_default_exterior_facet_integral(*L*, *ir*, *parameters*)
Return implementation of ufc::form::create_default_exterior_facet_integral().

create_default_interface_integral(*L*, *ir*, *parameters*)
Return implementation of ufc::form::create_default_interface_integral().

create_default_interior_facet_integral(*L*, *ir*, *parameters*)
Return implementation of ufc::form::create_default_interior_facet_integral().

create_default_overlap_integral(*L*, *ir*, *parameters*)
Return implementation of ufc::form::create_default_overlap_integral().

create_default_vertex_integral(*L*, *ir*, *parameters*)
Return implementation of ufc::form::create_default_vertex_integral().

create_dofmap(*L*, *ir*)

create_exterior_facet_integral(*L*, *ir*, *parameters*)
Return implementation of ufc::form::create_exterior_facet_integral().

create_finite_element(*L*, *ir*)

create_interface_integral(*L*, *ir*, *parameters*)
Return implementation of ufc::form::create_interface_integral().

create_interior_facet_integral(*L*, *ir*, *parameters*)
Return implementation of ufc::form::create_interior_facet_integral().

create_overlap_integral(*L*, *ir*, *parameters*)
Return implementation of ufc::form::create_overlap_integral().

create_vertex_integral(*L*, *ir*, *parameters*)
Return implementation of ufc::form::create_vertex_integral().

has_cell_integrals(*L*, *ir*, *parameters*)
Return implementation of ufc::form::has_cell_integrals().

has_custom_integrals(*L*, *ir*, *parameters*)
Return implementation of ufc::form::has_custom_integrals().

has_cutcell_integrals(*L*, *ir*, *parameters*)
Return implementation of ufc::form::has_cutcell_integrals().

has_exterior_facet_integrals(*L*, *ir*, *parameters*)
Return implementation of ufc::form::has_exterior_facet_integrals().

has_interface_integrals(*L*, *ir*, *parameters*)
Return implementation of ufc::form::has_interface_integrals().

has_interior_facet_integrals(*L*, *ir*, *parameters*)
Return implementation of ufc::form::has_interior_facet_integrals().

has_overlap_integrals(*L*, *ir*, *parameters*)
Return implementation of ufc::form::has_overlap_integrals().

has_vertex_integrals(*L*, *ir*, *parameters*)
Return implementation of ufc::form::has_vertex_integrals().

max_cell_subdomain_id(*L*, *ir*, *parameters*)
Return implementation of ufc::form::max_cell_subdomain_id().

max_custom_subdomain_id(*L*, *ir*, *parameters*)
Return implementation of ufc::form::max_custom_subdomain_id().

max_cutcell_subdomain_id(*L*, *ir*, *parameters*)
Return implementation of ufc::form::max_cutcell_subdomain_id().

max_exterior_facet_subdomain_id(*L*, *ir*, *parameters*)
Return implementation of ufc::form::max_exterior_facet_subdomain_id().

max_interface_subdomain_id(*L*, *ir*, *parameters*)
Return implementation of ufc::form::max_interface_subdomain_id().

max_interior_facet_subdomain_id(*L*, *ir*, *parameters*)
Return implementation of ufc::form::max_interior_facet_subdomain_id().

max_overlap_subdomain_id(*L*, *ir*, *parameters*)
Return implementation of ufc::form::max_overlap_subdomain_id().

max_vertex_subdomain_id(*L*, *ir*, *parameters*)
Return implementation of ufc::form::max_vertex_subdomain_id().

num_coefficients(*L*, *num_coefficients*)

original_coefficient_position(*L*, *ir*)

rank(*L*, *rank*)

ffc.uflacs.backends.ufc.generator module

class ffc.uflacs.backends.ufc.generator.**ufc_generator**(*basename*)
Bases: object

Common functionality for code generators producing ufc classes.

The generate function is the driver for generating code for a class. It automatically extracts each template keyword %(foo)s and calls self.foo(...) to define the code snippet for that keyword.

The standard arguments to every self.foo(...) function are: - L (the language module reference) - ir (the full ir dict) - parameters (the parameters dict, can be omitted) If the second argument is not named "ir", it must be a valid key in ir, and the value of ir[keyname] is passed instead of the full ir dict. Invalid keynames result in attempts at informative errors, meaning errors will often be caught early when making changes.

classname(*L*, *ir*)
Return classname.

constructor(*L*, *ir*)
Return empty string. Override in classes that need constructor.

constructor_arguments(*L*, *ir*)
Return empty string. Override in classes that need constructor.

create(*L*, *ir*)
Default implementation of creating a new object of the same type.

destructor(*L*, *ir*)
Return empty string. Override in classes that need destructor.

generate(*L*, *ir*, *parameters=None*, *snippets=None*, *jit=False*)
Return composition of templates with generated snippets.

generate_snippets(*L*, *ir*, *parameters*)
Generate code snippets for each keyword found in templates.

initializer_list(*L*, *ir*)
Return empty string. Override in classes that need constructor.

members(*L*, *ir*)
Return empty string. Override in classes that need members.

signature(*L*, *ir*)
Default implementation of returning signature string fetched from ir.

ffc.uflacs.backends.ufc.generators module

ffc.uflacs.backends.ufc.integrals module

class ffc.uflacs.backends.ufc.integrals.**ufc_cell_integral**
Bases: *ffc.uflacs.backends.ufc.integrals.ufc_integral*

class ffc.uflacs.backends.ufc.integrals.**ufc_custom_integral**
Bases: *ffc.uflacs.backends.ufc.integrals.ufc_integral*

num_cells(*L*, *ir*)

class ffc.uflacs.backends.ufc.integrals.**ufc_cutcell_integral**
Bases: *ffc.uflacs.backends.ufc.integrals.ufc_integral*

class ffc.uflacs.backends.ufc.integrals.**ufc_exterior_facet_integral**
Bases: *ffc.uflacs.backends.ufc.integrals.ufc_integral*

class ffc.uflacs.backends.ufc.integrals.**ufc_integral**(*integral_type*)
Bases: *ffc.uflacs.backends.ufc.generator.ufc_generator*

Each function maps to a keyword in the template. See documentation of ufc_generator.

enabled_coefficients(*L*, *ir*)

tabulate_tensor(*L*, *ir*)

tabulate_tensor_comment(*L*, *ir*)
Generate comment for tabulate_tensor.

class ffc.uflacs.backends.ufc.integrals.**ufc_interface_integral**
Bases: *ffc.uflacs.backends.ufc.integrals.ufc_integral*

class ffc.uflacs.backends.ufc.integrals.**ufc_interior_facet_integral**
Bases: *ffc.uflacs.backends.ufc.integrals.ufc_integral*

class ffc.uflacs.backends.ufc.integrals.**ufc_overlap_integral**
Bases: *ffc.uflacs.backends.ufc.integrals.ufc_integral*

class ffc.uflacs.backends.ufc.integrals.**ufc_vertex_integral**
Bases: *ffc.uflacs.backends.ufc.integrals.ufc_integral*

ffc.uflacs.backends.ufc.jacobian module

ffc.uflacs.backends.ufc.jacobian.**fiat_coordinate_mapping**(*L*, *cellname*, *gdim*, *ref_coord_symbol='Y'*)

ffc.uflacs.backends.ufc.jacobian.**inverse_jacobian**(*L*, *gdim*, *tdim*, *element_cellname*)

ffc.uflacs.backends.ufc.jacobian.**jacobian**(*L*, *gdim*, *tdim*, *element_cellname*)

ffc.uflacs.backends.ufc.jacobian.**orientation**(*L*)

ffc.uflacs.backends.ufc.templates module

ffc.uflacs.backends.ufc.templates.**extract_keywords**(*template*)

ffc.uflacs.backends.ufc.utils module

ffc.uflacs.backends.ufc.utils.**generate_error**(*L*, *msg*, *emit_warning*)

ffc.uflacs.backends.ufc.utils.**generate_return_bool_switch**(*L*, *i*, *values*, *default*)

ffc.uflacs.backends.ufc.utils.**generate_return_int_switch**(*L*, *i*, *values*, *default*)

ffc.uflacs.backends.ufc.utils.**generate_return_literal_switch**(*L*, *i*, *values*, *default*, *literal_type*, *typename=None*)

ffc.uflacs.backends.ufc.utils.**generate_return_new**(*L*, *classname*, *factory*)

ffc.uflacs.backends.ufc.utils.**generate_return_new_switch**(*L*, *i*, *classnames*, *args=None*, *factory=False*)

ffc.uflacs.backends.ufc.utils.**generate_return_sizet_switch**(*L*, *i*, *values*, *default*)

Module contents

Backend for generating UFC code.

Module contents

Collection of backends to UFLACS form compiler algorithms.

In the end we want to specify a clean and mature API here.

We may or may not want to move e.g. the FFC backend to FFC instead of having it here.

ffc.uflacs.language package

Submodules

ffc.uflacs.language.cnodes module

class ffc.uflacs.language.cnodes.**Add**(*lhs*, *rhs*)

Bases: *ffc.uflacs.language.cnodes.BinOp*

op = '+'

precedence = 5

class ffc.uflacs.language.cnodes.**AddressOf**(*arg*)

Bases: *ffc.uflacs.language.cnodes.PrefixUnaryOp*

op = '&'

precedence = 3

class ffc.uflacs.language.cnodes.**And**(*lhs*, *rhs*)
Bases: *ffc.uflacs.language.cnodes.BinOp*

op = '&&'

precedence = 11

class ffc.uflacs.language.cnodes.**ArrayAccess**(*array*, *indices*)
Bases: *ffc.uflacs.language.cnodes.CExprOperator*

array

ce_format(*precision=None*)

indices

precedence = 2

class ffc.uflacs.language.cnodes.**ArrayDecl**(*typename*, *symbol*, *sizes=None*, *values=None*, *alignas=None*, *padlen=0*)
Bases: *ffc.uflacs.language.cnodes.CStatement*

A declaration or definition of an array.

Note that just setting values=0 is sufficient to initialize the entire array to zero.

Otherwise use nested lists of lists to represent multidimensional array values to initialize to.

alignas

cs_format(*precision=None*)
Return S: string | list(S) | Indented(S).

is_scoped = False

padlen

sizes

symbol

typename

values

class ffc.uflacs.language.cnodes.**Assign**(*lhs*, *rhs*)
Bases: *ffc.uflacs.language.cnodes.AssignOp*

op = '='

class ffc.uflacs.language.cnodes.**AssignAdd**(*lhs*, *rhs*)
Bases: *ffc.uflacs.language.cnodes.AssignOp*

op = '+='

class ffc.uflacs.language.cnodes.**AssignAnd**(*lhs*, *rhs*)
Bases: *ffc.uflacs.language.cnodes.AssignOp*

op = '&&='

class ffc.uflacs.language.cnodes.**AssignBitAnd**(*lhs*, *rhs*)
Bases: *ffc.uflacs.language.cnodes.AssignOp*

op = '&='

class ffc.uflacs.language.cnodes.**AssignBitOr**(*lhs*, *rhs*)
Bases: *ffc.uflacs.language.cnodes.AssignOp*

op = '|='

class ffc.uflacs.language.cnodes.**AssignBitXor**(*lhs*, *rhs*)
Bases: *ffc.uflacs.language.cnodes.AssignOp*

op = '^='

class ffc.uflacs.language.cnodes.**AssignDiv**(*lhs*, *rhs*)
Bases: *ffc.uflacs.language.cnodes.AssignOp*

op = '/='

class ffc.uflacs.language.cnodes.**AssignLShift**(*lhs*, *rhs*)
Bases: *ffc.uflacs.language.cnodes.AssignOp*

op = '<<='

class ffc.uflacs.language.cnodes.**AssignMod**(*lhs*, *rhs*)
Bases: *ffc.uflacs.language.cnodes.AssignOp*

op = '%='

class ffc.uflacs.language.cnodes.**AssignMul**(*lhs*, *rhs*)
Bases: *ffc.uflacs.language.cnodes.AssignOp*

op = '*='

class ffc.uflacs.language.cnodes.**AssignOp**(*lhs*, *rhs*)
Bases: *ffc.uflacs.language.cnodes.BinOp*

Base class for assignment operators.

precedence = 13

sideeffect = True

class ffc.uflacs.language.cnodes.**AssignOr**(*lhs*, *rhs*)
Bases: *ffc.uflacs.language.cnodes.AssignOp*

op = '||='

class ffc.uflacs.language.cnodes.**AssignRShift**(*lhs*, *rhs*)
Bases: *ffc.uflacs.language.cnodes.AssignOp*

op = '>>='

class ffc.uflacs.language.cnodes.**AssignSub**(*lhs*, *rhs*)
Bases: *ffc.uflacs.language.cnodes.AssignOp*

op = '-='

class ffc.uflacs.language.cnodes.**BinOp**(*lhs*, *rhs*)
Bases: *ffc.uflacs.language.cnodes.CExprOperator*

ce_format(*precision=None*)

lhs

rhs

class ffc.uflacs.language.cnodes.**BitAnd**(*lhs*, *rhs*)
Bases: *ffc.uflacs.language.cnodes.BinOp*

op = '&'

precedence = 8

class ffc.uflacs.language.cnodes.**BitNot**(*arg*)
Bases: *ffc.uflacs.language.cnodes.PrefixUnaryOp*

op = '~'

precedence = 3

class ffc.uflacs.language.cnodes.**BitOr**(*lhs*, *rhs*)
Bases: *ffc.uflacs.language.cnodes.BinOp*

op = '|'

precedence = 10

class ffc.uflacs.language.cnodes.**BitXor**(*lhs*, *rhs*)
Bases: *ffc.uflacs.language.cnodes.BinOp*

op = '^'

precedence = 9

class ffc.uflacs.language.cnodes.**Break**
Bases: *ffc.uflacs.language.cnodes.CStatement*

cs_format(*precision=None*)
Return S: string | list(S) | Indented(S).

is_scoped = True

class ffc.uflacs.language.cnodes.**CExpr**
Bases: *ffc.uflacs.language.cnodes.CNode*

Base class for all C expressions.

All subtypes should define a 'precedence' class attribute.

ce_format(*precision=None*)

class ffc.uflacs.language.cnodes.**CExprLiteral**
Bases: *ffc.uflacs.language.cnodes.CExprTerminal*

A float or int literal value.

precedence = 0

class ffc.uflacs.language.cnodes.**CExprOperator**
Bases: *ffc.uflacs.language.cnodes.CExpr*

Base class for all C expression operator.

sideeffect = False

class ffc.uflacs.language.cnodes.**CExprTerminal**
Bases: *ffc.uflacs.language.cnodes.CExpr*

Base class for all C expression terminals.

sideeffect = False

class ffc.uflacs.language.cnodes.**CNode**
Bases: object

Base class for all C AST nodes.

debug = False

class ffc.uflacs.language.cnodes.**CStatement**
Bases: *ffc.uflacs.language.cnodes.CNode*

Base class for all C statements.

Subtypes do _not_ define a 'precedence' class attribute.

cs_format(*precision=None*)
Return S: string | list(S) | Indented(S).

is_scoped = False

class ffc.uflacs.language.cnodes.**Call**(*function*, *arguments=None*)
Bases: *ffc.uflacs.language.cnodes.CExprOperator*

arguments

ce_format(*precision=None*)

function

precedence = 2

sideeffect = True

class ffc.uflacs.language.cnodes.**Case**(*value*)
Bases: *ffc.uflacs.language.cnodes.CStatement*

cs_format(*precision=None*)
Return S: string | list(S) | Indented(S).

is_scoped = False

value

class ffc.uflacs.language.cnodes.**Comment**(*comment*)
Bases: *ffc.uflacs.language.cnodes.CStatement*

Line comment(s) used for annotating the generated code with human readable remarks.

comment

cs_format(*precision=None*)
Return S: string | list(S) | Indented(S).

is_scoped = True

class ffc.uflacs.language.cnodes.**Conditional**(*condition*, *true*, *false*)
Bases: *ffc.uflacs.language.cnodes.CExprOperator*

ce_format(*precision=None*)

condition

false

precedence = 13

true

class ffc.uflacs.language.cnodes.**Continue**
Bases: *ffc.uflacs.language.cnodes.CStatement*

cs_format(*precision=None*)
Return S: string | list(S) | Indented(S).

is_scoped = True

class ffc.uflacs.language.cnodes.**Default**
Bases: *ffc.uflacs.language.cnodes.CStatement*

cs_format(*precision=None*)
Return S: string | list(S) | Indented(S).

is_scoped = False

class ffc.uflacs.language.cnodes.**Dereference**(*arg*)
Bases: *ffc.uflacs.language.cnodes.PrefixUnaryOp*

op = '*'

precedence = 3

class ffc.uflacs.language.cnodes.**Div**(*lhs*, *rhs*)
Bases: *ffc.uflacs.language.cnodes.BinOp*

op = '/'

precedence = 4

class ffc.uflacs.language.cnodes.**Do**(*condition*, *body*)
Bases: *ffc.uflacs.language.cnodes.CStatement*

body

condition

cs_format(*precision=None*)
Return S: string | list(S) | Indented(S).

is_scoped = True

class ffc.uflacs.language.cnodes.**EQ**(*lhs*, *rhs*)
Bases: *ffc.uflacs.language.cnodes.BinOp*

op = '=='

precedence = 7

class ffc.uflacs.language.cnodes.**Else**(*body*)
Bases: *ffc.uflacs.language.cnodes.CStatement*

body

cs_format(*precision=None*)
Return S: string | list(S) | Indented(S).

is_scoped = True

class ffc.uflacs.language.cnodes.**ElseIf**(*condition*, *body*)
Bases: *ffc.uflacs.language.cnodes.CStatement*

body

condition

cs_format(*precision=None*)
Return S: string | list(S) | Indented(S).

is_scoped = True

class ffc.uflacs.language.cnodes.**FlattenedArray**(*array*, *dummy=None*, *dims=None*, *strides=None*, *offset=None*)
Bases: object

Syntax carrying object only, will get translated on __getitem__ to ArrayAccess.

array

dims

offset

strides

class ffc.uflacs.language.cnodes.**For**(*init*, *check*, *update*, *body*, *pragma=None*)
Bases: *ffc.uflacs.language.cnodes.CStatement*

body

check

cs_format(*precision=None*)
Return S: string | list(S) | Indented(S).

init

is_scoped = True

pragma

update

class ffc.uflacs.language.cnodes.**ForRange**(*index*, *begin*, *end*, *body*, *index_type='int'*, *vectorize=None*)
Bases: *ffc.uflacs.language.cnodes.CStatement*

Slightly higher-level for loop assuming incrementing an index over a range.

begin

body

cs_format(*precision=None*)
Return S: string | list(S) | Indented(S).

end

index

index_type

is_scoped = True

pragma

ffc.uflacs.language.cnodes.**ForRanges**(**ranges*, ***kwargs*)

class ffc.uflacs.language.cnodes.**GE**(*lhs*, *rhs*)
Bases: *ffc.uflacs.language.cnodes.BinOp*

op = '>='

precedence = 6

class ffc.uflacs.language.cnodes.**GT**(*lhs*, *rhs*)
Bases: *ffc.uflacs.language.cnodes.BinOp*

op = '>'

precedence = 6

class ffc.uflacs.language.cnodes.**If**(*condition*, *body*)
Bases: *ffc.uflacs.language.cnodes.CStatement*

body

condition

cs_format(*precision=None*)
Return S: string | list(S) | Indented(S).

is_scoped = True

class ffc.uflacs.language.cnodes.**LE**(*lhs*, *rhs*)
Bases: *ffc.uflacs.language.cnodes.BinOp*

op = '<='

precedence = 6

class ffc.uflacs.language.cnodes.**LT**(*lhs*, *rhs*)
Bases: *ffc.uflacs.language.cnodes.BinOp*

op = '<'

precedence = 6

class ffc.uflacs.language.cnodes.**LiteralBool**(*value*)
Bases: *ffc.uflacs.language.cnodes.CExprLiteral*

A boolean literal value.

ce_format(*precision=None*)

precedence = 0

value

class ffc.uflacs.language.cnodes.**LiteralFloat**(*value*)
Bases: *ffc.uflacs.language.cnodes.CExprLiteral*

A floating point literal value.

ce_format(*precision=None*)

precedence = 0

value

class ffc.uflacs.language.cnodes.**LiteralInt**(*value*)
Bases: *ffc.uflacs.language.cnodes.CExprLiteral*

An integer literal value.

ce_format(*precision=None*)

precedence = 0

value

class ffc.uflacs.language.cnodes.**LiteralString**(*value*)
Bases: *ffc.uflacs.language.cnodes.CExprLiteral*

A boolean literal value.

ce_format(*precision=None*)

precedence = 0

value

ffc.uflacs.language.cnodes.**MemCopy**(*src*, *dst*, *size*)

ffc.uflacs.language.cnodes.**MemZero**(*name*, *size*)

ffc.uflacs.language.cnodes.**MemZeroRange**(*name*, *begin*, *end*)

class ffc.uflacs.language.cnodes.**Mod**(*lhs*, *rhs*)
Bases: *ffc.uflacs.language.cnodes.BinOp*

op = '%'

precedence = 4

class ffc.uflacs.language.cnodes.**Mul**(*lhs*, *rhs*)
Bases: *ffc.uflacs.language.cnodes.BinOp*

op = '*'

precedence = 4

class ffc.uflacs.language.cnodes.**NE**(*lhs*, *rhs*)
Bases: *ffc.uflacs.language.cnodes.BinOp*

op = '!='

precedence = 7

class ffc.uflacs.language.cnodes.**Namespace**(*name*, *body*)
Bases: *ffc.uflacs.language.cnodes.CStatement*

body

cs_format(*precision=None*)
Return S: string | list(S) | Indented(S).

is_scoped = True

name

class ffc.uflacs.language.cnodes.**NaryOp**(*args*)
Bases: *ffc.uflacs.language.cnodes.CExprOperator*

Base class for special n-ary operators.

args

ce_format(*precision=None*)

class ffc.uflacs.language.cnodes.**Neg**(*arg*)
Bases: *ffc.uflacs.language.cnodes.PrefixUnaryOp*

op = '-'

precedence = 3

class ffc.uflacs.language.cnodes.**New**(*typename*)
Bases: *ffc.uflacs.language.cnodes.CExpr*

ce_format(*precision=None*)

typename

ffc.uflacs.language.cnodes.**NoOp**()

class ffc.uflacs.language.cnodes.**Not**(*arg*)
Bases: *ffc.uflacs.language.cnodes.PrefixUnaryOp*

op = '!'

precedence = 3

class ffc.uflacs.language.cnodes.**Null**
Bases: *ffc.uflacs.language.cnodes.CExprLiteral*

A null pointer literal.

ce_format(*precision=None*)

precedence = 0

class ffc.uflacs.language.cnodes.**Or**(*lhs*, *rhs*)
Bases: *ffc.uflacs.language.cnodes.BinOp*

op = '||'

precedence = 12

class ffc.uflacs.language.cnodes.**Pos**(*arg*)
Bases: *ffc.uflacs.language.cnodes.PrefixUnaryOp*

op = '+'

precedence = 3

class ffc.uflacs.language.cnodes.**PostDecrement**(*arg*)
Bases: *ffc.uflacs.language.cnodes.PostfixUnaryOp*

op = '--'

precedence = 2

sideeffect = True

class ffc.uflacs.language.cnodes.**PostIncrement**(*arg*)
Bases: *ffc.uflacs.language.cnodes.PostfixUnaryOp*

op = '++'

precedence = 2

sideeffect = True

class ffc.uflacs.language.cnodes.**PostfixUnaryOp**(*arg*)
Bases: *ffc.uflacs.language.cnodes.UnaryOp*

Base class for postfix unary operators.

ce_format(*precision=None*)

class ffc.uflacs.language.cnodes.**Pragma**(*comment*)
Bases: *ffc.uflacs.language.cnodes.CStatement*

Pragma comments used for compiler-specific annotations.

comment

cs_format(*precision=None*)
Return S: string | list(S) | Indented(S).

is_scoped = True

class ffc.uflacs.language.cnodes.**PreDecrement**(*arg*)
Bases: *ffc.uflacs.language.cnodes.PrefixUnaryOp*

op = '--'

precedence = 3

sideeffect = True

class ffc.uflacs.language.cnodes.**PreIncrement**(*arg*)
Bases: *ffc.uflacs.language.cnodes.PrefixUnaryOp*

op = '++'

precedence = 3

sideeffect = True

class ffc.uflacs.language.cnodes.**PrefixUnaryOp**(*arg*)
Bases: *ffc.uflacs.language.cnodes.UnaryOp*

Base class for prefix unary operators.

ce_format(*precision=None*)

class ffc.uflacs.language.cnodes.**Product**(*args*)
Bases: *ffc.uflacs.language.cnodes.NaryOp*

Product of any number of operands.

op = '*'

precedence = 4

class ffc.uflacs.language.cnodes.**Return**(*value=None*)
Bases: *ffc.uflacs.language.cnodes.CStatement*

cs_format(*precision=None*)
Return S: string | list(S) | Indented(S).

is_scoped = True

value

class ffc.uflacs.language.cnodes.**Scope**(*body*)
Bases: *ffc.uflacs.language.cnodes.CStatement*

body

cs_format(*precision=None*)
Return S: string | list(S) | Indented(S).

is_scoped = True

class ffc.uflacs.language.cnodes.**SizeOf**(*arg*)
Bases: *ffc.uflacs.language.cnodes.PrefixUnaryOp*

op = 'sizeof'

precedence = 3

ffc.uflacs.language.cnodes.**Sqrt**(*x*)

class ffc.uflacs.language.cnodes.**Statement**(*expr*)
Bases: *ffc.uflacs.language.cnodes.CStatement*

Make an expression into a statement.

cs_format(*precision=None*)
Return S: string | list(S) | Indented(S).

expr

is_scoped = False

class ffc.uflacs.language.cnodes.**StatementList**(*statements*)
Bases: *ffc.uflacs.language.cnodes.CStatement*

A simple sequence of statements. No new scopes are introduced.

cs_format(*precision=None*)
Return S: string | list(S) | Indented(S).

is_scoped
bool(x) -> bool

Returns True when the argument x is true, False otherwise. The builtins True and False are the only two instances of the class bool. The class bool is a subclass of the class int, and cannot be subclassed.

statements

class ffc.uflacs.language.cnodes.**Sub**(*lhs*, *rhs*)
Bases: *ffc.uflacs.language.cnodes.BinOp*

op = '-'

precedence = 5

class ffc.uflacs.language.cnodes.**Sum**(*args*)
Bases: *ffc.uflacs.language.cnodes.NaryOp*

Sum of any number of operands.

op = '+'

precedence = 5

class ffc.uflacs.language.cnodes.**Switch**(*arg*, *cases*, *default=None*, *autobreak=True*, *autoscope=True*)
Bases: *ffc.uflacs.language.cnodes.CStatement*

arg

autobreak

autoscope

cases

cs_format(*precision=None*)
Return S: string | list(S) | Indented(S).

default

is_scoped = True

class ffc.uflacs.language.cnodes.**Symbol**(*name*)
Bases: *ffc.uflacs.language.cnodes.CExprTerminal*

A named symbol.

ce_format(*precision=None*)

name

precedence = 0

class ffc.uflacs.language.cnodes.**Throw**(*exception*, *message*)
Bases: *ffc.uflacs.language.cnodes.CStatement*

cs_format(*precision=None*)
Return S: string | list(S) | Indented(S).

exception

is_scoped = True

message

class ffc.uflacs.language.cnodes.**UnaryOp**(*arg*)
Bases: *ffc.uflacs.language.cnodes.CExprOperator*

Base class for unary operators.

arg

class ffc.uflacs.language.cnodes.**Using**(*name*)
Bases: *ffc.uflacs.language.cnodes.CStatement*

cs_format(*precision=None*)
Return S: string | list(S) | Indented(S).

is_scoped = True

name

class ffc.uflacs.language.cnodes.**VariableDecl**(*typename*, *symbol*, *value=None*)
Bases: *ffc.uflacs.language.cnodes.CStatement*

Declare a variable, optionally define initial value.

cs_format(*precision=None*)
Return S: string | list(S) | Indented(S).

is_scoped = False

symbol

typename

value

class ffc.uflacs.language.cnodes.**VerbatimExpr**(*codestring*)
Bases: *ffc.uflacs.language.cnodes.CExprTerminal*

A verbatim copy of an expression source string.

Handled as having the lowest precedence which will introduce parentheses around it most of the time.

ce_format(*precision=None*)

codestring

precedence = 15

class ffc.uflacs.language.cnodes.**VerbatimStatement**(*codestring*)
Bases: *ffc.uflacs.language.cnodes.CStatement*

Wraps a source code string to be pasted verbatim into the source code.

codestring

cs_format(*precision=None*)
Return S: string | list(S) | Indented(S).

is_scoped = False

class ffc.uflacs.language.cnodes.**While**(*condition*, *body*)
Bases: *ffc.uflacs.language.cnodes.CStatement*

body

condition

cs_format(*precision=None*)
Return S: string | list(S) | Indented(S).

is_scoped = True

ffc.uflacs.language.cnodes.**accumulate_loop**(*dst*, *src*, *ranges*)
Generate a nested loop over a list of ranges, adding src to dst in the innermost loop.

Ranges is a list on the format [(index, begin, end),...].

ffc.uflacs.language.cnodes.**as_cexpr**(*node*)
Typechecks and wraps an object as a valid CExpr.

Accepts CExpr nodes, treats int and float as literals, and treats a string as a symbol.

ffc.uflacs.language.cnodes.**as_cexpr_or_literal**(*node*)

ffc.uflacs.language.cnodes.**as_cexpr_or_string_symbol**(*node*)

ffc.uflacs.language.cnodes.**as_cexpr_or_verbatim**(*node*)

ffc.uflacs.language.cnodes.**as_cstatement**(*node*)
Perform type checking on node and wrap in a suitable statement type if necessary.

ffc.uflacs.language.cnodes.**as_pragma**(*pragma*)

ffc.uflacs.language.cnodes.**as_symbol**(*symbol*)

ffc.uflacs.language.cnodes.**assign_loop**(*dst*, *src*, *ranges*)
Generate a nested loop over a list of ranges, assigning src to dst in the innermost loop.

Ranges is a list on the format [(index, begin, end),...].

ffc.uflacs.language.cnodes.**build_1d_initializer_list**(*values*, *formatter*, *padlen=0*, *precision=None*)
Return a list containing a single line formatted like "{ 0.0, 1.0, 2.0 }"

ffc.uflacs.language.cnodes.**build_initializer_lists**(*values*, *sizes*, *level*, *formatter*, *padlen=0*, *precision=None*)
Return a list of lines with initializer lists for a multidimensional array.

Example output:

```
{ { 0.0, 0.1 },
  { 1.0, 1.1 } }
```

ffc.uflacs.language.cnodes.**commented_code_list**(*code*, *comments*)
Convenience wrapper for adding comment to code list if the list is not empty.

ffc.uflacs.language.cnodes.**flattened_indices**(*indices*, *shape*)
Given a tuple of indices and a shape tuple, return CNode expression for flattened indexing into multidimensional array.

Indices and shape entries can be int values, str symbol names, or CNode expressions.

ffc.uflacs.language.cnodes.**float_product**(*factors*)
Build product of float factors, simplifying ones and zeros and returning 1.0 if empty sequence.

ffc.uflacs.language.cnodes.**is_negative_one_cexpr**(*cexpr*)

ffc.uflacs.language.cnodes.**is_one_cexpr**(*cexpr*)

ffc.uflacs.language.cnodes.**is_simple_inner_loop**(*code*)

ffc.uflacs.language.cnodes.**is_zero_cexpr**(*cexpr*)

ffc.uflacs.language.cnodes.**leftover**(*size*, *padlen*)
Return minimum integer to add to size to make it divisible by padlen.

ffc.uflacs.language.cnodes.**pad_dim**(*dim*, *padlen*)
Make dim divisible by padlen.

ffc.uflacs.language.cnodes.**pad_innermost_dim**(*shape*, *padlen*)
Make the last dimension in shape divisible by padlen.

ffc.uflacs.language.cnodes.**scale_loop**(*dst*, *factor*, *ranges*)
Generate a nested loop over a list of ranges, multiplying dst with factor in the innermost loop.

Ranges is a list on the format [(index, begin, end),...].

ffc.uflacs.language.format_lines module

Tools for indentation-aware code string stitching.

When formatting an AST into a string, it's better to collect lists of snippets and then join them than adding the pieces continually, which gives O(n^2) behaviour w.r.t. AST size n.

class ffc.uflacs.language.format_lines.**Indented**(*body*)
Bases: object

Class to mark a collection of snippets for indentation.

This way nested indentations can be handled by adding the prefix spaces only once to each line instead of splitting and indenting substrings repeatedly.

body

ffc.uflacs.language.format_lines.**format_indented_lines**(*snippets*, *level=0*)
Format recursive sequences of indented lines as one string.

ffc.uflacs.language.format_lines.**iter_indented_lines**(*snippets*, *level=0*)
Iterate over indented string lines from a snippets data structure.

The snippets object can be built recursively using the following types:

- str: Split and yield as one line at a time indented to the appropriate level.
- Indented: Yield the lines within this object indented by one level.
- tuple,list: Yield lines from recursive application of this function to list items.

ffc.uflacs.language.format_value module

ffc.uflacs.language.format_value.**format_float**(*x*, *precision=None*)
Format a float value according to given precision.

ffc.uflacs.language.format_value.**format_int**(*x*, *precision=None*)

ffc.uflacs.language.format_value.**format_value**(*value*, *precision=None*)
Format a literal value as s tring.

- float: Formatted according to current precision configuration.
- int: Formatted as regular base 10 int literal.
- str: Wrapped in "quotes".

ffc.uflacs.language.precedence module

class ffc.uflacs.language.precedence.**PRECEDENCE**

Bases: object

An enum-like class for C operator precedence levels.

ADD = 5

ADDRESSOF = 3

AND = 11

ASSIGN = 13

BIT_AND = 8

BIT_NOT = 3

BIT_OR = 10

BIT_XOR = 9

CALL = 2

CONDITIONAL = 13

DEREFERENCE = 3

DIV = 4

EQ = 7

GE = 6

GT = 6

HIGHEST = 0

LE = 6

LITERAL = 0

LOWEST = 15

LT = 6

MOD = 4

MUL = 4

NE = 7

NEG = 3

NOT = 3

OR = 12

POS = 3

POST_DEC = 2

POST_INC = 2

PRE_DEC = 3

PRE_INC = 3

SIZEOF = 3

SUB = 5

SUBSCRIPT = 2

SYMBOL = 0

ffc.uflacs.language.ufl_to_cnodes module

Tools for C/C++ expression formatting.

class ffc.uflacs.language.ufl_to_cnodes.**RulesForC**
Bases: object

abs(*o*, *op*)

max_value(*o*, *a*, *b*)

min_value(*o*, *a*, *b*)

power(*o*, *a*, *b*)

class ffc.uflacs.language.ufl_to_cnodes.**RulesForCpp**
Bases: object

abs(*o*, *op*)

bessel_i(*o*, *n*, *v*)

bessel_j(*o*, *n*, *v*)

bessel_k(*o*, *n*, *v*)

bessel_y(*o*, *n*, *v*)

max_value(*o*, *a*, *b*)

min_value(*o*, *a*, *b*)

power(*o*, *a*, *b*)

class ffc.uflacs.language.ufl_to_cnodes.**UFL2CNodesMixin**(*language*)
Bases: object

Rules collection mixin for a UFL to CNodes translator class.

acos(*o*, *op*)

and_condition(*o*, *a*, *b*)

asin(*o*, *op*)

atan(*o*, *op*)

atan_2(*o*, *y*, *x*)

conditional(*o*, *c*, *t*, *f*)

cos(*o*, *op*)

cosh(*o*, *op*)

division(*o*, *a*, *b*)

eq(*o*, *a*, *b*)

erf(*o*, *op*)

exp(*o*, *op*)

expr(*o*)
Generic fallback with error message for missing rules.

float_value(*o*)

ge(*o*, *a*, *b*)

gt(*o*, *a*, *b*)

int_value(*o*)

le(*o*, *a*, *b*)

ln(*o*, *op*)

lt(*o*, *a*, *b*)

math_function(*o*, *op*)

ne(*o*, *a*, *b*)

not_condition(*o*, *a*)

or_condition(*o*, *a*, *b*)

product(*o*, *a*, *b*)

sin(*o*, *op*)

sinh(*o*, *op*)

sqrt(*o*, *op*)

sum(*o*, *a*, *b*)

tan(*o*, *op*)

tanh(*o*, *op*)

zero(*o*)

class ffc.uflacs.language.ufl_to_cnodes.**UFL2CNodesTranslatorC**(*language*)
Bases: ufl.corealg.multifunction.MultiFunction, *ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin*, *ffc.uflacs.language.ufl_to_cnodes.RulesForC*

UFL to CNodes translator class.

class ffc.uflacs.language.ufl_to_cnodes.**UFL2CNodesTranslatorCpp**(*language*)
Bases: ufl.corealg.multifunction.MultiFunction, *ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin*, *ffc.uflacs.language.ufl_to_cnodes.RulesForCpp*

UFL to CNodes translator class.

Module contents

Submodules

ffc.uflacs.build_uflacs_ir module

Main algorithm for building the uflacs intermediate representation.

ffc.uflacs.build_uflacs_ir.**analyse_dependencies**(*V*, *V_deps*, *V_targets*, *modified_terminal_indices*, *modified_terminals*, *mt_unique_table_reference*)

ffc.uflacs.build_uflacs_ir.**build_scalar_graph**(*expressions*)

Build list representation of expression graph covering the given expressions.

TODO: Renaming, refactoring and cleanup of the graph building algorithms used in here

ffc.uflacs.build_uflacs_ir.**build_uflacs_ir**(*cell*, *integral_type*, *entitytype*, *integrands*, *tensor_shape*, *coefficient_numbering*, *quadrature_rules*, *parameters*)

class ffc.uflacs.build_uflacs_ir.**common_block_data_t**(*block_mode*, *ttypes*, *factor_index*, *factor_is_piecewise*, *unames*, *restrictions*, *transposed*)

Bases: tuple

block_mode

Alias for field number 0

factor_index

Alias for field number 2

factor_is_piecewise

Alias for field number 3

restrictions

Alias for field number 5

transposed

Alias for field number 6

ttypes

Alias for field number 1

unames

Alias for field number 4

ffc.uflacs.build_uflacs_ir.**empty_expr_ir**()

class ffc.uflacs.build_uflacs_ir.**full_block_data_t**(*block_mode*, *ttypes*, *factor_index*, *factor_is_piecewise*, *unames*, *restrictions*, *transposed*, *ma_data*)

Bases: tuple

block_mode

Alias for field number 0

factor_index

Alias for field number 2

factor_is_piecewise

Alias for field number 3

ma_data

Alias for field number 7

restrictions

Alias for field number 5

transposed
Alias for field number 6

ttypes
Alias for field number 1

unames
Alias for field number 4

ffc.uflacs.build_uflacs_ir.**get_common_block_data**(*blockdata*)

ffc.uflacs.build_uflacs_ir.**integrate_block**(*weights*, *unames*, *ttypes*, *unique_tables*, *unique_table_num_dofs*)

ffc.uflacs.build_uflacs_ir.**integrate_block_interior_facets**(*weights*, *unames*, *ttypes*, *unique_tables*, *unique_table_num_dofs*)

class ffc.uflacs.build_uflacs_ir.**ma_data_t**(*ma_index*, *tabledata*)
Bases: tuple

ma_index
Alias for field number 0

tabledata
Alias for field number 1

ffc.uflacs.build_uflacs_ir.**multiply_block**(*point_index*, *unames*, *ttypes*, *unique_tables*, *unique_table_num_dofs*)

ffc.uflacs.build_uflacs_ir.**multiply_block_interior_facets**(*point_index*, *unames*, *ttypes*, *unique_tables*, *unique_table_num_dofs*)

ffc.uflacs.build_uflacs_ir.**parse_uflacs_optimization_parameters**(*parameters*, *integral_type*)
Following model from quadrature representation, extracting uflacs specific parameters from the global parameters dict.

class ffc.uflacs.build_uflacs_ir.**partial_block_data_t**(*block_mode*, *ttypes*, *factor_index*, *factor_is_piecewise*, *unames*, *restrictions*, *transposed*, *ma_data*, *piecewise_ma_index*)
Bases: tuple

block_mode
Alias for field number 0

factor_index
Alias for field number 2

factor_is_piecewise
Alias for field number 3

ma_data
Alias for field number 7

piecewise_ma_index
Alias for field number 8

restrictions
Alias for field number 5

transposed
Alias for field number 6

ttypes
Alias for field number 1

unames
Alias for field number 4

class ffc.uflacs.build_uflacs_ir.**preintegrated_block_data_t**(*block_mode*, *ttypes*, *factor_index*, *factor_is_piecewise*, *unames*, *restrictions*, *transposed*, *is_uniform*, *name*)

Bases: tuple

block_mode
Alias for field number 0

factor_index
Alias for field number 2

factor_is_piecewise
Alias for field number 3

is_uniform
Alias for field number 7

name
Alias for field number 8

restrictions
Alias for field number 5

transposed
Alias for field number 6

ttypes
Alias for field number 1

unames
Alias for field number 4

class ffc.uflacs.build_uflacs_ir.**premultiplied_block_data_t**(*block_mode*, *ttypes*, *factor_index*, *factor_is_piecewise*, *unames*, *restrictions*, *transposed*, *is_uniform*, *name*)

Bases: tuple

block_mode
Alias for field number 0

factor_index
Alias for field number 2

factor_is_piecewise
Alias for field number 3

is_uniform
Alias for field number 7

name
Alias for field number 8

restrictions
Alias for field number 5

transposed
Alias for field number 6

ttypes
Alias for field number 1

unames
Alias for field number 4

ffc.uflacs.build_uflacs_ir.**uflacs_default_parameters**(*optimize*)
Default parameters for tuning of uflacs code generation.

These are considered experimental and may change without deprecation mechanism at any time.

ffc.uflacs.elementtables module

Tools for precomputed tables of terminal values.

ffc.uflacs.elementtables.**analyse_table_type**(*table*, *rtol=1e-05*, *atol=1e-08*)

ffc.uflacs.elementtables.**analyse_table_types**(*unique_tables*, *rtol=1e-05*, *atol=1e-08*)

ffc.uflacs.elementtables.**build_element_tables**(*num_points*, *quadrature_rules*, *cell*, *integral_type*, *entitytype*, *modified_terminals*, *rtol=1e-05*, *atol=1e-08*)
Build the element tables needed for a list of modified terminals.

Input: entitytype - str modified_terminals - ordered sequence of unique modified terminals FIXME: Document

Output: tables - dict(name: table) mt_table_names - dict(ModifiedTerminal: name)

ffc.uflacs.elementtables.**build_optimized_tables**(*num_points*, *quadrature_rules*, *cell*, *integral_type*, *entitytype*, *modified_terminals*, *existing_tables*, *compress_zeros*, *rtol=1e-05*, *atol=1e-08*)

ffc.uflacs.elementtables.**build_unique_tables**(*tables*, *rtol=1e-05*, *atol=1e-08*)
Given a list or dict of tables, return a list of unique tables and a dict of unique table indices for each input table key.

ffc.uflacs.elementtables.**clamp_table_small_numbers**(*table*, *rtol=1e-05*, *atol=1e-08*, *numbers=(-1.0, -0.5, 0.0, 0.5, 1.0)*)
Clamp almost 0,1,-1 values to integers. Returns new table.

ffc.uflacs.elementtables.**equal_tables**(*a*, *b*, *rtol=1e-05*, *atol=1e-08*)

ffc.uflacs.elementtables.**generate_psi_table_name**(*num_points*, *element_counter*, *averaged*, *entitytype*, *derivative_counts*, *flat_component*)
Generate a name for the psi table of the form: FE#_C#_D###[_AC|_AF|][_F|V][_Q#], where '#' will be an integer value.

FE - is a simple counter to distinguish the various bases, it will be assigned in an arbitrary fashion.

C - is the component number if any (this does not yet take into account tensor valued functions)

D - is the number of derivatives in each spatial direction if any. If the element is defined in 3D, then D012 means d^3(*)/dydz^2.

AC - marks that the element values are averaged over the cell

AF - marks that the element values are averaged over the facet

F - marks that the first array dimension enumerates facets on the cell

V - marks that the first array dimension enumerates vertices on the cell

Q - number of quadrature points, to distinguish between tables in a mixed quadrature degree setting

ffc.uflacs.elementtables.**get_ffc_table_values**(*points*, *cell*, *integral_type*, *ufl_element*, *avg*, *entitytype*, *derivative_counts*, *flat_component*)

Extract values from ffc element table.

Returns a 3D numpy array with axes (entity number, quadrature point number, dof number)

ffc.uflacs.elementtables.**get_modified_terminal_element**(*mt*)

ffc.uflacs.elementtables.**is_ones_table**(*table*, *rtol=1e-05*, *atol=1e-08*)

ffc.uflacs.elementtables.**is_piecewise_table**(*table*, *rtol=1e-05*, *atol=1e-08*)

ffc.uflacs.elementtables.**is_quadrature_table**(*table*, *rtol=1e-05*, *atol=1e-08*)

ffc.uflacs.elementtables.**is_uniform_table**(*table*, *rtol=1e-05*, *atol=1e-08*)

ffc.uflacs.elementtables.**is_zeros_table**(*table*, *rtol=1e-05*, *atol=1e-08*)

ffc.uflacs.elementtables.**optimize_element_tables**(*tables*, *table_origins*, *compress_zeros*, *rtol=1e-05*, *atol=1e-08*)

Optimize tables and make unique set.

Steps taken:

- clamp values that are very close to -1, 0, +1 to those values
- remove dofs from beginning and end of tables where values are all zero
- for each modified terminal, provide the dof range that a given table corresponds to

Terminology: name - str, name used in input arguments here table - numpy array of float values stripped_table - numpy array of float values with zeroes

removed from each end of dofrange

Input: tables - { name: table } table_origins - FIXME

Output: unique_tables - { unique_name: stripped_table } unique_table_origins - FIXME

ffc.uflacs.elementtables.**strip_table_zeros**(*table*, *compress_zeros*, *rtol=1e-05*, *atol=1e-08*)

Strip zero columns from table. Returns column range (begin, end) and the new compact table.

ffc.uflacs.elementtables.**table_origin_t**
alias of ffc.uflacs.elementtables.table_origin

ffc.uflacs.elementtables.**unique_table_reference_t**
alias of ffc.uflacs.elementtables.unique_table_reference

ffc.uflacs.integralgenerator module

Controlling algorithm for building the tabulate_tensor source structure from factorized representation.

class ffc.uflacs.integralgenerator.**IntegralGenerator**(*ir*, *backend*, *precision*)
Bases: object

generate()
Generate entire tabulate_tensor body.

Assumes that the code returned from here will be wrapped in a context that matches a suitable version of the UFC tabulate_tensor signatures.

generate_block_parts(*num_points*, *blockmap*, *blockdata*)
Generate and return code parts for a given block.

Returns parts occuring before, inside, and after the quadrature loop identified by num_points.

Should be called with num_points=None for quadloop-independent blocks.

generate_copyout_statements()
Generate statements copying results to output array.

generate_dofblock_partition(*num_points*)

generate_element_tables()
Generate static tables with precomputed element basis function values in quadrature points.

generate_expr_copyout_statements()

generate_partition(*symbol*, *V*, *V_active*, *V_mts*, *mt_tabledata*, *num_points*)

generate_preintegrated_dofblock_partition()

generate_quadrature_loop(*num_points*)
Generate quadrature loop with for this num_points.

generate_quadrature_tables()
Generate static tables of quadrature points and weights.

generate_runtime_quadrature_loop()
Generate quadrature loop for custom integrals, with physical points given runtime.

generate_tensor_copyout_statements()

generate_tensor_reset()
Generate statements for resetting the element tensor to zero.

generate_tensor_value_initialization(*A_values*)

generate_unstructured_piecewise_partition()

generate_unstructured_varying_partition(*num_points*)

get_arg_factors(*blockdata*, *block_rank*, *num_points*, *iq*, *indices*)

get_entities(*blockdata*)

get_includes()
Return list of include statements needed to support generated code.

get_temp_symbol(*tempname*, *key*)

get_var(*num_points*, *v*)
“Lookup ufl expression v in variable scope dicts.

Scope is determined by num_points which identifies the quadrature loop scope or None if outside quadrature loops.

If v is not found in quadrature loop scope, the piecewise scope (None) is checked.

Returns the CNodes expression to access the value in the code.

has_var(*num_points*, *v*)
"Check if variable exists in variable scope dicts.

Return True if ufl expression v exists in the num_points scope.

NB! Does not fall back to piecewise scope.

init_scopes()
Initialize variable scope dicts.

new_temp_symbol(*basename*)
Create a new code symbol named basename + running counter.

set_var(*num_points*, *v*, *vaccess*)
"Set a new variable in variable scope dicts.

Scope is determined by num_points which identifies the quadrature loop scope or None if outside quadrature loops.

v is the ufl expression and vaccess is the CNodes expression to access the value in the code.

ffc.uflacs.params module

Collection of exposed parameters available to tune form compiler algorithms.

ffc.uflacs.params.**default_parameters**()

ffc.uflacs.tools module

ffc.uflacs.tools.**accumulate_integrals**(*itg_data*, *quadrature_rule_sizes*)
Group and accumulate integrals according to the number of quadrature points in their rules.

ffc.uflacs.tools.**collect_quadrature_rules**(*integrals*, *default_scheme*, *default_degree*)
Collect quadrature rules found in list of integrals.

ffc.uflacs.tools.**compute_quadrature_rules**(*rules*, *integral_type*, *cell*)
Compute points and weights for a set of quadrature rules.

ffc.uflacs.uflacsgenerator module

Controlling algorithm for building the tabulate_tensor source structure from factorized representation.

ffc.uflacs.uflacsgenerator.**generate_integral_code**(*ir*, *prefix*, *parameters*)
Generate code for integral from intermediate representation.

ffc.uflacs.uflacsoptimization module

ffc.uflacs.uflacsoptimization.**optimize_integral_ir**(*ir*, *parameters*)
Compute optimized intermediate representation of integral.

ffc.uflacs.uflacsrepresentation module

ffc.uflacs.uflacsrepresentation.**compute_integral_ir**(*itg_data*, *form_data*, *form_id*, *element_numbers*, *classnames*, *parameters*)

Compute intermediate represention of integral.

Module contents

This is UFLACS, the UFL Analyser and Compiler System.

1.3.2 Submodules

1.3.3 ffc.analysis module

Compiler stage 1: Analysis

This module implements the analysis/preprocessing of variational forms, including automatic selection of elements, degrees and form representation type.

ffc.analysis.**analyze_coordinate_mappings**(*coordinate_elements*, *parameters*)

ffc.analysis.**analyze_elements**(*elements*, *parameters*)

ffc.analysis.**analyze_forms**(*forms*, *parameters*)

Analyze form(s), returning

form_datas - a tuple of form_data objects unique_elements - a tuple of unique elements across all forms element_numbers - a mapping to unique numbers for all elements

ffc.analysis.**analyze_ufl_objects**(*ufl_objects*, *kind*, *parameters*)

Analyze ufl object(s), either forms, elements, or coordinate mappings, returning:

form_datas - a tuple of form_data objects unique_elements - a tuple of unique elements across all forms element_numbers - a mapping to unique numbers for all elements

1.3.4 ffc.classname module

This module defines some basics for generating C++ code.

ffc.classname.**make_classname**(*prefix*, *basename*, *signature*)

ffc.classname.**make_integral_classname**(*prefix*, *integral_type*, *form_id*, *subdomain_id*)

1.3.5 ffc.codegeneration module

Compiler stage 4: Code generation

This module implements the generation of C++ code for the body of each UFC function from an (optimized) intermediate representation (OIR).

ffc.codegeneration.**generate_code**(*ir*, *parameters*)

Generate code from intermediate representation.

1.3.6 ffc.compiler module

This is the compiler, acting as the main interface for compilation of forms and breaking the compilation into several sequential stages. The output of each stage is the input of the next stage.

Compiler stage 0: Language, parsing

Input: Python code or .ufl file Output: UFL form

This stage consists of parsing and expressing a form in the UFL form language.

This stage is completely handled by UFL.

Compiler stage 1: Analysis

Input: UFL form Output: Preprocessed UFL form and FormData (metadata)

This stage preprocesses the UFL form and extracts form metadata. It may also perform simplifications on the form.

Compiler stage 2: Code representation

Input: Preprocessed UFL form and FormData (metadata) Output: Intermediate Representation (IR)

This stage examines the input and generates all data needed for code generation. This includes generation of finite element basis functions, extraction of data for mapping of degrees of freedom and possible precomputation of integrals.

Most of the complexity of compilation is handled in this stage.

The IR is stored as a dictionary, mapping names of UFC functions to data needed for generation of the corresponding code.

Compiler stage 3: Optimization

Input: Intermediate Representation (IR) Output: Optimized Intermediate Representation (OIR)

This stage examines the IR and performs optimizations.

Optimization is currently disabled as a separate stage but is implemented as part of the code generation for quadrature representation.

Compiler stage 4: Code generation

Input: Optimized Intermediate Representation (OIR) Output: C++ code

This stage examines the OIR and generates the actual C++ code for the body of each UFC function.

The code is stored as a dictionary, mapping names of UFC functions to strings containing the C++ code of the body of each function.

Compiler stage 5: Code formatting

Input: C++ code Output: C++ code files

This stage examines the generated C++ code and formats it according to the UFC format, generating as output one or more .h/.cpp files conforming to the UFC format.

The main interface is defined by the following two functions:

compile_form compile_element

The compiler stages are implemented by the following functions:

analyze_forms or analyze_elements (stage 1) compute_ir (stage 2) optimize_ir (stage 3) generate_code (stage 4) format_code (stage 5)

ffc.compiler.**compile_form**(*forms*, *object_names=None*, *prefix='Form'*, *parameters=None*, *jit=False*)
This function generates UFC code for a given UFL form or list of UFL forms.

ffc.compiler.**compile_element**(*elements*, *object_names=None*, *prefix='Element'*, *parameters=None*, *jit=False*)
This function generates UFC code for a given UFL element or list of UFL elements.

1.3.7 ffc.fiatinterface module

class ffc.fiatinterface.**SpaceOfReals**
Bases: object

Constant over the entire domain, rather than just cellwise.

ffc.fiatinterface.**create_element**(*ufl_element*)

ffc.fiatinterface.**create_quadrature**(*shape*, *degree*, *scheme='default'*)
Generate quadrature rule (points, weights) for given shape that will integrate an polynomial of order 'degree' exactly.

ffc.fiatinterface.**map_facet_points**(*points*, *facet*, *cellname*)
Map points from the e (UFC) reference simplex of dimension d - 1 to a given facet on the (UFC) reference simplex of dimension d. This may be used to transform points tabulated for example on the 2D reference triangle to points on a given facet of the reference tetrahedron.

ffc.fiatinterface.**reference_cell**(*cellname*)
Return FIAT reference cell

ffc.fiatinterface.**reference_cell_vertices**(*cellname*)
Return dict of coordinates of reference cell vertices for this 'cellname'.

1.3.8 ffc.formatting module

Compiler stage 5: Code formatting

This module implements the formatting of UFC code from a given dictionary of generated C++ code for the body of each UFC function.

It relies on templates for UFC code available as part of the module ufc_utils.

ffc.formatting.**format_code**(*code*, *wrapper_code*, *prefix*, *parameters*, *jit=False*)
Format given code in UFC format. Returns two strings with header and source file contents.

ffc.formatting.**generate_factory_functions**(*prefix*, *kind*, *classname*)

ffc.formatting.**generate_jit_factory_functions**(*code*, *prefix*)

ffc.formatting.**write_code**(*code_h*, *code_c*, *prefix*, *parameters*)

1.3.9 ffc.git_commit_hash module

ffc.git_commit_hash.**git_commit_hash**()
Return git changeset hash (returns "unknown" if changeset is not known)

1.3.10 ffc.jitcompiler module

This module provides a just-in-time (JIT) form compiler. It uses dijitso to wrap the generated code into a Python module.

exception ffc.jitcompiler.**FFCError**
Bases: Exception

exception ffc.jitcompiler.**FFCJitError**
Bases: *ffc.jitcompiler.FFCError*

ffc.jitcompiler.**compute_jit_prefix**(*ufl_object*, *parameters*, *kind=None*)
Compute the prefix (module name) for jit modules.

ffc.jitcompiler.**jit**(*ufl_object*, *parameters=None*, *indirect=False*)
Just-in-time compile the given form or element

Parameters:

ufl_object : The UFL object to be compiled parameters : A set of parameters

ffc.jitcompiler.**jit_build**(*ufl_object*, *module_name*, *parameters*)
Wraps dijitso jit with some parameter conversion etc.

ffc.jitcompiler.**jit_generate**(*ufl_object*, *module_name*, *signature*, *parameters*)
Callback function passed to dijitso.jit: generate code and return as strings.

1.3.11 ffc.log module

This module provides functions used by the FFC implementation to output messages. These may be redirected by the user of FFC.

This module reuses the corresponding log.py module from UFL which is a wrapper for the standard Python logging module.

ffc.log.**add_indent**(**message*)

ffc.log.**add_logfile**(**message*)

ffc.log.**begin**(**message*)

ffc.log.**debug**(**message*)

ffc.log.**debug_code**(*code*, *name=''*)
Debug generated code.

ffc.log.**debug_dict**(*d*, *title=''*)
Pretty-print dictionary.

ffc.log.**debug_ir**(*ir*, *name=''*)
Debug intermediate representation.

ffc.log.**deprecate**(**message*)

ffc.log.**end**(**message*)

ffc.log.**error**(**message*)

ffc.log.**ffc_assert**(*condition*, **message*)
Assert that condition is true and otherwise issue an error with given message.

ffc.log.**get_handler**(**message*)

ffc.log.**get_logger**(**message*)

ffc.log.**info**(**message*)

ffc.log.**info_blue**(**message*)

ffc.log.**info_green**(**message*)

ffc.log.**info_red**(**message*)

ffc.log.**log**(**message*)

ffc.log.**pop_level**(**message*)

ffc.log.**push_level**(**message*)

ffc.log.**set_handler**(**message*)

ffc.log.**set_indent**(**message*)

ffc.log.**set_level**(**message*)

ffc.log.**set_prefix**(**message*)

ffc.log.**warning**(**message*)

ffc.log.**warning_blue**(**message*)

ffc.log.**warning_green**(**message*)

ffc.log.**warning_red**(**message*)

1.3.12 ffc.main module

This script is the command-line interface to FFC.

It parses command-line arguments and generates code from input UFL form files.

ffc.main.**compile_ufl_data**(*ufd*, *prefix*, *parameters*)

ffc.main.**info_usage**()
Print usage information.

ffc.main.**info_version**()
Print version number.

ffc.main.**main**(*args=None*)
This is the commandline tool for the python module ffc.

ffc.main.**print_error**(*msg*)
Print error message (cannot use log system at top level).

1.3.13 ffc.optimization module

Compiler stage 5: optimization

This module implements the optimization of an intermediate code representation.

ffc.optimization.**optimize_ir**(*ir*, *parameters*)
Optimize intermediate form representation.

1.3.14 ffc.parameters module

ffc.parameters.**compilation_relevant_parameters**(*parameters*)

ffc.parameters.**compute_jit_parameters_signature**(*parameters*)
Return parameters signature (some parameters must be ignored).

ffc.parameters.**default_jit_parameters**()

ffc.parameters.**default_parameters**()
Return (a copy of) the default parameter values for FFC.

ffc.parameters.**split_parameters**(*parameters*)
Split a parameters dict into groups based on what parameters are used for.

ffc.parameters.**validate_jit_parameters**(*parameters*)
Check parameters and add any missing parameters

ffc.parameters.**validate_parameters**(*parameters*)
Initial check of parameters.

1.3.15 ffc.plot module

This module provides functionality for plotting finite elements.

ffc.plot.**plot**(*element*, *rotate=True*)
Plot finite element.

1.3.16 ffc.representation module

Compiler stage 2: Code representation

This module computes intermediate representations of forms, elements and dofmaps. For each UFC function, we extract the data needed for code generation at a later stage.

The representation should conform strictly to the naming and order of functions in UFC. Thus, for code generation of the function "foo", one should only need to use the data stored in the intermediate representation under the key "foo".

ffc.representation.**all_elements**(*fiat_element*)

ffc.representation.**cell_midpoint**(*cell*)

ffc.representation.**compute_ir**(*analysis*, *prefix*, *parameters*, *jit=False*)
Compute intermediate representation.

ffc.representation.**make_all_element_classnames**(*prefix*, *elements*, *coordinate_elements*, *element_numbers*, *parameters*, *jit*)

ffc.representation.**make_coordinate_mapping_jit_classname**(*ufl_mesh*, *parameters*)

ffc.representation.**make_dofmap_jit_classname**(*ufl_element*, *parameters*)

ffc.representation.**make_finite_element_jit_classname**(*ufl_element*, *parameters*)

ffc.representation.**needs_oriented_jacobian**(*fiat_element*)

ffc.representation.**pick_representation**(*representation*)
Return one of the specialized code generation modules from a representation string.

ffc.representation.**uses_integral_moments**(*fiat_element*)
True if element uses integral moments for its degrees of freedom.

1.3.17 ffc.representationutils module

This module contains utility functions for some code shared between quadrature and tensor representation.

ffc.representationutils.**create_quadrature_points_and_weights**(*integral_type*, *cell*, *degree*, *rule*)
Create quadrature rule and return points and weights.

ffc.representationutils.**entity_type_from_integral_type**(*integral_type*)

ffc.representationutils.**generate_enabled_coefficients**(*enabled_coefficients*)

ffc.representationutils.**initialize_integral_code**(*ir*, *prefix*, *parameters*)
Representation independent default initialization of code dict for integral from intermediate representation.

ffc.representationutils.**initialize_integral_ir**(*representation*, *itg_data*, *form_data*, *form_id*)
Initialize a representation dict with common information that is expected independently of which representation is chosen.

ffc.representationutils.**integral_type_to_entity_dim**(*integral_type*, *tdim*)
Given integral_type and domain tdim, return the tdim of the integration entity.

ffc.representationutils.**map_integral_points**(*points*, *integral_type*, *cell*, *entity*)
Map points from reference entity to its parent reference cell.

ffc.representationutils.**needs_oriented_jacobian**(*form_data*)

ffc.representationutils.**transform_component**(*component*, *offset*, *ufl_element*)
This function accounts for the fact that if the geometrical and topological dimension does not match, then for native vector elements, in particular the Piola-mapped ones, the physical value dimensions and the reference value dimensions are not the same. This has certain consequences for mixed elements, aka 'fun with offsets'.

1.3.18 ffc.utils module

ffc.utils.**all_equal**(*sequence*)
Check that all items in list are equal.

ffc.utils.**compute_permutations**(*k*, *n*, *skip=[]*)
Compute all permutations of k elements from (0, n) in rising order. Any elements that are contained in the list skip are not included.

ffc.utils.**insert_nested_dict**(*root*, *keys*, *value*)
Set root[keys[0]][...][keys[-1]] = value, creating subdicts on the way if missing.

ffc.utils.**listcopy**(*sequence*)
Create a copy of the list, calling the copy constructor on each object in the list (problems when using copy.deepcopy).

ffc.utils.**pick_first**(*sequence*)
Check that all values are equal and return the value.

1.3.19 ffc.wrappers module

ffc.wrappers.**generate_wrapper_code**(*analysis*, *prefix*, *object_names*, *parameters*)
Generate code for additional wrappers.

1.3.20 Module contents

FEniCS Form Compiler (FFC)

FFC compiles finite element variational forms into C++ code.

The interface consists of the following functions:

compile_form - Compilation of forms compile_element - Compilation of finite elements jit - Just-In-Time compilation of forms and elements default_parameters - Default parameter values for FFC ufc_signature - Signature of UFC interface (SHA-1 hash of ufc.h)

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 2019.1.0

Summary of changes

- Support `NodalEnrichedElement`

Detailed changes

- Support `NodalEnrichedElement` see `demo/NodalMini.ufl`.

1.4.3 Changes in version 2018.1.0

FFC 2018.1.0 was released on 2018-06-14.

Summary of changes

- Remove python2 support.

1.4.4 Changes in version 2017.2.0

FFC 2017.2.0 was released on 2017-12-05.

Summary of changes

- Some fixes for ufc::eval for esoteric element combinations
- Reimplement code generation for all ufc classes with new class ufc::coordinate_mapping which can map between coordinates, compute jacobians, etc. for a coordinate mapping parameterized by a specific finite element.
- New functions in ufc::finite_element: - evaluate_reference_basis - evaluate_reference_basis_derivatives - transform_reference_basis_derivatives - tabulate_reference_dof_coordinates
- New functions in ufc::dofmap: - num_global_support_dofs - num_element_support_dofs
- Improved docstrings for parts of ufc.h
- FFC now accepts Q and DQ finite element families defined on quadrilaterals and hexahedrons
- Some fixes for ufc_geometry.h for quadrilateral and hexahedron cells

1.4.5 Changes in version 2017.1.0

FFC 2017.1.0.post2 was released on 2017-09-12.

Summary of changes

- Change PyPI package name to fenics-ffc.

1.4.6 Changes in version 2017.1.0

FFC 2017.1.0 was released on 2017-05-09.

Summary of changes

- Add experimental `tsfc` representation; for installation see *TSFC requirements*

Detailed changes

- Let ffc -O parameter take an optional integer level like -O2, -O0
- Implement blockwise optimizations in uflacs code generation
- Expose uflacs optimization parameters through parameter system

1.4.7 Changes in version 2016.2.0

FFC 2016.2.0 was released on 2016-11-30.

Summary of changes

- Generalize ufc interface to non-affine parameterized coordinates
- Add `ufc::coordinate_mapping` class
- Make ufc interface depend on C++11 features requiring gcc version >= 4.8
- Change the mapping `pullback as metric` to `double covariant piola` (this preserves tangential-tangential trace).
- Added Hellan-Herrmann-Johnson element as supported element
- Add mapping `double contravariant piola` (this preserves normal-normal trace).
- Include comment with effective representation and integral metadata to generated `tabulate_tensor` code

Detailed changes

- Jit compiler now compiles elements separately from forms to avoid duplicate work
- Add parameter max_signature_length to optionally shorten signatures in the jit cache
- Move uflacs module into ffc.uflacs
- Remove installation of pkg-config and CMake files (UFC path and compiler flags are available from ffc module)
- Add dependency on dijitso and remove dependency on instant
- Add experimental Bitbucket pipelines
- Tidy the repo after UFC and UFLACS merge, and general spring cleanup. This includes removal of instructions how to merge two repos, commit hash c8389032268041fe94682790cb773663bdf27286.

1.4.8 Changes in version 2016.1.0

FFC 2016.1.0 was released on 2016-06-23.

- Add function get_ufc_include to get path to ufc.h
- Merge UFLACS into FFC
- Generalize ufc interface to non-affine parameterized coordinates
- Add ufc::coordinate_mapping class
- Make ufc interface depend on C++11 features requiring gcc version >= 4.8
- Add function ufc_signature() to the form compiler interface

- Add function git_commit_hash()

1.4.9 Changes in version 1.6.0

FFC 1.6.0 was released on 2015-07-28.

- Rename and modify a number of UFC interface functions. See docstrings in ufc.h for details.
- Bump required SWIG version to 3.0.3
- Disable dual basis (`tabulate_coordinates` and `evaluate_dofs`) for enriched elements until correct implementation is brought up

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

- genindex
- modindex

Python Module Index

f

ffc, 71
ffc.analysis, 64
ffc.classname, 64
ffc.codegeneration, 64
ffc.compiler, 65
ffc.errorcontrol, 6
ffc.errorcontrol.errorcontrol, 5
ffc.errorcontrol.errorcontrolgenerators, 5
ffc.fiatinterface, 66
ffc.formatting, 66
ffc.git_commit_hash, 67
ffc.jitcompiler, 67
ffc.log, 67
ffc.main, 68
ffc.optimization, 69
ffc.parameters, 69
ffc.plot, 69
ffc.quadrature, 18
ffc.quadrature.codesnippets, 6
ffc.quadrature.cpp, 6
ffc.quadrature.deprecation, 7
ffc.quadrature.expr, 7
ffc.quadrature.floatvalue, 7
ffc.quadrature.fraction, 7
ffc.quadrature.optimisedquadraturetransformer, 8
ffc.quadrature.parameters, 9
ffc.quadrature.product, 9
ffc.quadrature.quadratureregenerator, 10
ffc.quadrature.quadratureoptimization, 10
ffc.quadrature.quadraturerepresentation, 10
ffc.quadrature.quadraturetransformer, 11
ffc.quadrature.quadraturetransformerbase, 12
ffc.quadrature.quadratureutils, 15
ffc.quadrature.reduce_operations, 15
ffc.quadrature.sumobj, 16
ffc.quadrature.symbol, 17
ffc.quadrature.symbolics, 17
ffc.quadrature.tabulate_basis, 18
ffc.representation, 69
ffc.representationutils, 70
ffc.tsfc.tsfcoptimization, 18
ffc.tsfc.tsfcrepresentation, 18
ffc.uflacs, 64
ffc.uflacs.analysis, 26
ffc.uflacs.analysis.balancing, 19
ffc.uflacs.analysis.crsarray, 19
ffc.uflacs.analysis.dependencies, 19
ffc.uflacs.analysis.expr_shapes, 20
ffc.uflacs.analysis.factorization, 20
ffc.uflacs.analysis.graph, 21
ffc.uflacs.analysis.graph_rebuild, 21
ffc.uflacs.analysis.graph_ssa, 22
ffc.uflacs.analysis.graph_symbols, 23
ffc.uflacs.analysis.graph_vertices, 24
ffc.uflacs.analysis.indexing, 24
ffc.uflacs.analysis.modified_terminals, 24
ffc.uflacs.analysis.valuenumbering, 25
ffc.uflacs.backends, 39
ffc.uflacs.backends.ffc, 31
ffc.uflacs.backends.ffc.access, 27
ffc.uflacs.backends.ffc.backend, 28
ffc.uflacs.backends.ffc.common, 28
ffc.uflacs.backends.ffc.definitions, 28
ffc.uflacs.backends.ffc.symbols, 30
ffc.uflacs.backends.ufc, 39
ffc.uflacs.backends.ufc.apply_mappings, 31
ffc.uflacs.backends.ufc.coordinate_mapping, 31
ffc.uflacs.backends.ufc.dofmap, 32
ffc.uflacs.backends.ufc.evalderivs, 33

ffc.uflacs.backends.ufc.evaluatebasis, 33
ffc.uflacs.backends.ufc.evaluatebasisderivatives, 33
ffc.uflacs.backends.ufc.evaluatedof, 34
ffc.uflacs.backends.ufc.finite_element, 34
ffc.uflacs.backends.ufc.form, 35
ffc.uflacs.backends.ufc.generator, 37
ffc.uflacs.backends.ufc.generators, 38
ffc.uflacs.backends.ufc.integrals, 38
ffc.uflacs.backends.ufc.jacobian, 38
ffc.uflacs.backends.ufc.templates, 39
ffc.uflacs.backends.ufc.utils, 39
ffc.uflacs.build_uflacs_ir, 56
ffc.uflacs.elementtables, 60
ffc.uflacs.integralgenerator, 62
ffc.uflacs.language, 56
ffc.uflacs.language.cnodes, 39
ffc.uflacs.language.format_lines, 53
ffc.uflacs.language.format_value, 53
ffc.uflacs.language.precedence, 54
ffc.uflacs.language.ufl_to_cnodes, 55
ffc.uflacs.params, 63
ffc.uflacs.tools, 63
ffc.uflacs.uflacsgenerator, 63
ffc.uflacs.uflacsoptimization, 63
ffc.uflacs.uflacsrepresentation, 64
ffc.utils, 70
ffc.wrappers, 71

Index

A

abs() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt method), 8
abs() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 11
abs() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 12
abs() (ffc.uflacs.analysis.graph_rebuild.ReconstructScalarSubexpressions method), 21
abs() (ffc.uflacs.language.ufl_to_cnodes.RulesForC method), 55
abs() (ffc.uflacs.language.ufl_to_cnodes.RulesForCpp method), 55
accumulate_integrals() (in module ffc.uflacs.tools), 63
accumulate_loop() (in module ffc.uflacs.language.cnodes), 52
acos() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 12
acos() (ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin method), 55
Add (class in ffc.uflacs.language.cnodes), 39
ADD (ffc.uflacs.language.precedence.PRECEDENCE attribute), 54
add() (ffc.uflacs.analysis.factorization.Factors method), 20
add_indent() (in module ffc.log), 67
add_logfile() (in module ffc.log), 67
add_to_fv() (in module ffc.uflacs.analysis.factorization), 20
add_ufc_form_integral_methods() (in module ffc.uflacs.backends.ufc.form), 35
AddressOf (class in ffc.uflacs.language.cnodes), 39
ADDRESSOF (ffc.uflacs.language.precedence.PRECEDENCE attribute), 54
adj_expr_2x2() (in module ffc.uflacs.backends.ufc.coordinate_mapping), 31
adj_expr_3x3() (in module ffc.uflacs.backends.ufc.coordinate_mapping), 31
alignas (ffc.uflacs.language.cnodes.ArrayDecl attribute), 40
all_elements() (in module ffc.representation), 69
all_equal() (in module ffc.utils), 70
allocate_registers() (in module ffc.uflacs.analysis.graph_ssa), 22
analyse_dependencies() (in module ffc.uflacs.build_uflacs_ir), 56
analyse_modified_terminal() (in module ffc.uflacs.analysis.modified_terminals), 25
analyse_table_type() (in module ffc.uflacs.elementtables), 60
analyse_table_types() (in module ffc.uflacs.elementtables), 60
analyze_coordinate_mappings() (in module ffc.analysis), 64
analyze_elements() (in module ffc.analysis), 64
analyze_forms() (in module ffc.analysis), 64
analyze_ufl_objects() (in module ffc.analysis), 64
And (class in ffc.uflacs.language.cnodes), 40
AND (ffc.uflacs.language.precedence.PRECEDENCE attribute), 54
and_condition() (ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin method), 55
arg (ffc.uflacs.language.cnodes.Switch attribute), 50
arg (ffc.uflacs.language.cnodes.UnaryOp attribute), 51
args (ffc.uflacs.language.cnodes.NaryOp attribute), 47
argument() (ffc.quadrature.quadraturetransformerbase.QuadratureTransform method), 12
argument() (ffc.uflacs.backends.ffc.definitions.FFCBackendDefinitions method), 28
argument_loop_index() (ffc.uflacs.backends.ffc.symbols.FFCBackendSymb method), 30
argument_ordering_key() (ffc.uflacs.analysis.modified_terminals.ModifiedTerminal method), 25
arguments (ffc.uflacs.language.cnodes.Call attribute), 43
array (ffc.uflacs.language.cnodes.ArrayAccess attribute), 40

array (ffc.uflacs.language.cnodes.FlattenedArray attribute), 44
ArrayAccess (class in ffc.uflacs.language.cnodes), 40
ArrayDecl (class in ffc.uflacs.language.cnodes), 40
as_cexpr() (in module ffc.uflacs.language.cnodes), 52
as_cexpr_or_literal() (in module ffc.uflacs.language.cnodes), 52
as_cexpr_or_string_symbol() (in module ffc.uflacs.language.cnodes), 52
as_cexpr_or_verbatim() (in module ffc.uflacs.language.cnodes), 52
as_cstatement() (in module ffc.uflacs.language.cnodes), 52
as_pragma() (in module ffc.uflacs.language.cnodes), 52
as_symbol() (in module ffc.uflacs.language.cnodes), 52
as_tuple() (ffc.uflacs.analysis.modified_terminals.ModifiedTerminal method), 25
asin() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 12
asin() (ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin method), 55
Assign (class in ffc.uflacs.language.cnodes), 40
ASSIGN (ffc.uflacs.language.precedence.PRECEDENCE attribute), 54
assign_loop() (in module ffc.uflacs.language.cnodes), 52
AssignAdd (class in ffc.uflacs.language.cnodes), 40
AssignAnd (class in ffc.uflacs.language.cnodes), 40
AssignBitAnd (class in ffc.uflacs.language.cnodes), 40
AssignBitOr (class in ffc.uflacs.language.cnodes), 40
AssignBitXor (class in ffc.uflacs.language.cnodes), 41
AssignDiv (class in ffc.uflacs.language.cnodes), 41
AssignLShift (class in ffc.uflacs.language.cnodes), 41
AssignMod (class in ffc.uflacs.language.cnodes), 41
AssignMul (class in ffc.uflacs.language.cnodes), 41
AssignOp (class in ffc.uflacs.language.cnodes), 41
AssignOr (class in ffc.uflacs.language.cnodes), 41
AssignRShift (class in ffc.uflacs.language.cnodes), 41
AssignSub (class in ffc.uflacs.language.cnodes), 41
atan() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 12
atan() (ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin method), 55
atan_2() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 12
atan_2() (ffc.uflacs.analysis.graph_rebuild.ReconstructScalarSubexpressions method), 21
atan_2() (ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin method), 55
atan_2_function() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 12
autobreak (ffc.uflacs.language.cnodes.Switch attribute), 50
autoscope (ffc.uflacs.language.cnodes.Switch attribute), 50

B

balance_modified_terminal() (in module ffc.uflacs.analysis.balancing), 19
balance_modifiers() (in module ffc.uflacs.analysis.balancing), 19
BalanceModifiers (class in ffc.uflacs.analysis.balancing), 19
base_expr (ffc.quadrature.symbol.Symbol attribute), 17
base_op (ffc.quadrature.symbol.Symbol attribute), 17
begin (ffc.uflacs.language.cnodes.ForRange attribute), 45
begin() (in module ffc.log), 67
bessel_function() (ffc.quadrature.quadraturetransformerbase.QuadratureTra method), 12
bessel_function() (ffc.uflacs.analysis.graph_rebuild.ReconstructScalarSubex method), 21
bessel_i() (ffc.quadrature.quadraturetransformerbase.QuadratureTransforme method), 12
bessel_i() (ffc.uflacs.language.ufl_to_cnodes.RulesForCpp method), 55
bessel_j() (ffc.quadrature.quadraturetransformerbase.QuadratureTransforme method), 12
bessel_j() (ffc.uflacs.language.ufl_to_cnodes.RulesForCpp method), 55
bessel_k() (ffc.quadrature.quadraturetransformerbase.QuadratureTransform method), 12
bessel_k() (ffc.uflacs.language.ufl_to_cnodes.RulesForCpp method), 55
bessel_y() (ffc.quadrature.quadraturetransformerbase.QuadratureTransform method), 12
bessel_y() (ffc.uflacs.language.ufl_to_cnodes.RulesForCpp method), 55
binary_condition() (ffc.quadrature.optimisedquadraturetransformer.Quadrat method), 8
binary_condition() (ffc.quadrature.quadraturetransformer.QuadratureTransf method), 11
BinOp (class in ffc.uflacs.language.cnodes), 41
BIT_AND (ffc.uflacs.language.precedence.PRECEDENCE attribute), 54
BIT_NOT (ffc.uflacs.language.precedence.PRECEDENCE attribute), 54
BIT_OR (ffc.uflacs.language.precedence.PRECEDENCE attribute), 54
BIT_XOR (ffc.uflacs.language.precedence.PRECEDENCE attribute), 54
BitAnd (class in ffc.uflacs.language.cnodes), 41
BitNot (class in ffc.uflacs.language.cnodes), 42
BitOr (class in ffc.uflacs.language.cnodes), 42
BitXor (class in ffc.uflacs.language.cnodes), 42
block_mode (ffc.uflacs.build_uflacs_ir.common_block_data_t attribute), 57
block_mode (ffc.uflacs.build_uflacs_ir.full_block_data_t attribute), 57
block_mode (ffc.uflacs.build_uflacs_ir.partial_block_data_t attribute), 58

block_mode (ffc.uflacs.build_uflacs_ir.preintegrated_block_data_t attribute), 59
block_mode (ffc.uflacs.build_uflacs_ir.premultiplied_block_data_t attribute), 59
body (ffc.uflacs.language.cnodes.Do attribute), 44
body (ffc.uflacs.language.cnodes.Else attribute), 44
body (ffc.uflacs.language.cnodes.ElseIf attribute), 44
body (ffc.uflacs.language.cnodes.For attribute), 45
body (ffc.uflacs.language.cnodes.ForRange attribute), 45
body (ffc.uflacs.language.cnodes.If attribute), 45
body (ffc.uflacs.language.cnodes.Namespace attribute), 47
body (ffc.uflacs.language.cnodes.Scope attribute), 49
body (ffc.uflacs.language.cnodes.While attribute), 51
body (ffc.uflacs.language.format_lines.Indented attribute), 53
Break (class in ffc.uflacs.language.cnodes), 42
build_1d_initializer_list() (in module ffc.uflacs.language.cnodes), 52
build_argument_dependencies() (in module ffc.uflacs.analysis.factorization), 20
build_argument_indices() (in module ffc.uflacs.analysis.factorization), 20
build_array_from_counts() (in module ffc.uflacs.analysis.graph_vertices), 24
build_element_tables() (in module ffc.uflacs.elementtables), 60
build_graph() (in module ffc.uflacs.analysis.graph), 21
build_graph_symbols() (in module ffc.uflacs.analysis.graph_symbols), 23
build_graph_vertices() (in module ffc.uflacs.analysis.graph_vertices), 24
build_initializer_lists() (in module ffc.uflacs.language.cnodes), 52
build_node_counts() (in module ffc.uflacs.analysis.graph_vertices), 24
build_node_shapes() (in module ffc.uflacs.analysis.graph_symbols), 23
build_node_sizes() (in module ffc.uflacs.analysis.graph_symbols), 23
build_node_symbols() (in module ffc.uflacs.analysis.graph_symbols), 23
build_optimized_tables() (in module ffc.uflacs.elementtables), 60
build_scalar_graph() (in module ffc.uflacs.build_uflacs_ir), 57
build_scalar_graph_vertices() (in module ffc.uflacs.analysis.graph_vertices), 24
build_scalar_node_counts() (in module ffc.uflacs.analysis.graph_vertices), 24
build_uflacs_ir() (in module ffc.uflacs.build_uflacs_ir), 57
build_unique_tables() (in module ffc.uflacs.elementtables), 60

C

Call (class in ffc.uflacs.language.cnodes), 43
CALL (ffc.uflacs.language.precedence.PRECEDENCE attribute), 54
Case (class in ffc.uflacs.language.cnodes), 43
cases (ffc.uflacs.language.cnodes.Switch attribute), 50
ce_format() (ffc.uflacs.language.cnodes.ArrayAccess method), 40
ce_format() (ffc.uflacs.language.cnodes.BinOp method), 41
ce_format() (ffc.uflacs.language.cnodes.Call method), 43
ce_format() (ffc.uflacs.language.cnodes.CExpr method), 42
ce_format() (ffc.uflacs.language.cnodes.Conditional method), 43
ce_format() (ffc.uflacs.language.cnodes.LiteralBool method), 46
ce_format() (ffc.uflacs.language.cnodes.LiteralFloat method), 46
ce_format() (ffc.uflacs.language.cnodes.LiteralInt method), 46
ce_format() (ffc.uflacs.language.cnodes.LiteralString method), 46
ce_format() (ffc.uflacs.language.cnodes.NaryOp method), 47
ce_format() (ffc.uflacs.language.cnodes.New method), 47
ce_format() (ffc.uflacs.language.cnodes.Null method), 48
ce_format() (ffc.uflacs.language.cnodes.PostfixUnaryOp method), 48
ce_format() (ffc.uflacs.language.cnodes.PrefixUnaryOp method), 49
ce_format() (ffc.uflacs.language.cnodes.Symbol method), 50
ce_format() (ffc.uflacs.language.cnodes.VerbatimExpr method), 51
cell_avg() (ffc.quadrature.quadraturetransformerbase.QuadratureTransforme method), 12
cell_avg() (ffc.uflacs.analysis.balancing.BalanceModifiers method), 19
cell_avg() (ffc.uflacs.analysis.valuenumbering.ValueNumberer method), 25
cell_coordinate() (ffc.quadrature.optimisedquadraturetransformer.Quadratur method), 8
cell_coordinate() (ffc.quadrature.quadraturetransformer.QuadratureTransfor method), 11
cell_coordinate() (ffc.quadrature.quadraturetransformerbase.QuadratureTra method), 13
cell_coordinate() (ffc.uflacs.backends.ffc.access.FFCBackendAccess method), 27
cell_coordinate() (ffc.uflacs.backends.ffc.definitions.FFCBackendDefinition method), 28
cell_edge_vectors() (ffc.uflacs.backends.ffc.access.FFCBackendAccess method), 27
cell_edge_vectors() (ffc.uflacs.backends.ffc.definitions.FFCBackendDefiniti

method), 28
cell_facet_jacobian() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt method), 8
cell_facet_jacobian() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 11
cell_facet_jacobian() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 13
cell_facet_jacobian() (ffc.uflacs.backends.ffc.access.FFCBackendAccess method), 27
cell_facet_jacobian() (ffc.uflacs.backends.ffc.definitions.FFCBackendDefinitions method), 28
cell_facet_jacobian_determinant() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt method), 8
cell_facet_jacobian_determinant() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 11
cell_facet_jacobian_determinant() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 13
cell_facet_jacobian_inverse() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt method), 8
cell_facet_jacobian_inverse() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 11
cell_facet_jacobian_inverse() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 13
cell_facet_origin() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt method), 8
cell_facet_origin() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 11
cell_facet_origin() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 13
cell_midpoint() (in module ffc.representation), 69
cell_normal() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt method), 8
cell_normal() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 11
cell_normal() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 13
cell_normal() (ffc.uflacs.backends.ffc.access.FFCBackendAccess method), 27
cell_normal() (ffc.uflacs.backends.ffc.definitions.FFCBackendDefinitions method), 29
cell_orientation() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt method), 8
cell_orientation() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 11
cell_orientation() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 13
cell_orientation() (ffc.uflacs.backends.ffc.access.FFCBackendAccess method), 27
cell_orientation() (ffc.uflacs.backends.ffc.definitions.FFCBackendDefinitions method), 29
cell_orientation() (ffc.uflacs.backends.ffc.symbols.FFCBackendSymbols method), 30
cell_orientation_argument() (ffc.uflacs.backends.ffc.symbols.FFCBackendSymbols method), 30
cell_origin() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt method), 8
cell_origin() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 11
cell_origin() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 13
cell_residual() (ffc.errorcontrol.errorcontrolgenerators.ErrorControlGenerator method), 5
cell_shape() (ffc.uflacs.backends.ufc.coordinate_mapping.ufc_coordinate_mapping method), 32
cell_shape() (ffc.uflacs.backends.ufc.finite_element.ufc_finite_element method), 34
cell_vertices() (ffc.uflacs.backends.ffc.access.FFCBackendAccess method), 27
cell_vertices() (ffc.uflacs.backends.ffc.definitions.FFCBackendDefinitions method), 29
cell_volume() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt method), 8
cell_volume() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 11
cell_volume() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 13
CExpr (class in ffc.uflacs.language.cnodes), 42
CExprLiteral (class in ffc.uflacs.language.cnodes), 42
CExprOperator (class in ffc.uflacs.language.cnodes), 42
CExprTerminal (class in ffc.uflacs.language.cnodes), 42
check (ffc.uflacs.language.cnodes.For attribute), 45
circumradius() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt method), 8
circumradius() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 11
circumradius() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 13
clamp_table_small_numbers() (in module ffc.uflacs.elementtables), 60
classname() (ffc.uflacs.backends.ufc.generator.ufc_generator method), 37
CNode (class in ffc.uflacs.language.cnodes), 42
code (ffc.uflacs.language.cnodes.VerbatimExpr attribute), 51
codestring (ffc.uflacs.language.cnodes.VerbatimStatement attribute), 51
CoordinateTransformerBase (in module ffc.uflacs.backends.ufc.coordinate_mapping), 31
coefficient() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase

method), 13
coefficient() (ffc.uflacs.backends.ffc.access.FFCBackendAccess method), 27
coefficient() (ffc.uflacs.backends.ffc.definitions.FFCBackendDefinitions method), 29
coefficient_dof_access() (ffc.uflacs.backends.ffc.symbols.FFCBackendSymbols method), 30
coefficient_dof_sum_index() (ffc.uflacs.backends.ffc.symbols.FFCBackendSymbols method), 30
coefficient_value() (ffc.uflacs.backends.ffc.symbols.FFCBackendSymbols method), 30
collect_quadrature_rules() (in module ffc.uflacs.tools), 63
Comment (class in ffc.uflacs.language.cnodes), 43
comment (ffc.uflacs.language.cnodes.Comment attribute), 43
comment (ffc.uflacs.language.cnodes.Pragma attribute), 48
commented_code_list() (in module ffc.uflacs.language.cnodes), 52
common_block_data_t (class in ffc.uflacs.build_uflacs_ir), 57
compilation_relevant_parameters() (in module ffc.parameters), 69
compile_element() (in module ffc.compiler), 66
compile_form() (in module ffc.compiler), 66
compile_ufl_data() (in module ffc.main), 68
compile_with_error_control() (in module ffc.errorcontrol.errorcontrol), 5
component() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 13
component_tensor() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 13
component_tensor() (ffc.uflacs.analysis.graph_rebuild.ReconstructScalarSubexpressions method), 21
component_tensor() (ffc.uflacs.analysis.valuenumbering.ValueNumberer method), 25
compound_tensor_operator() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 13
compute_all_shapes() (in module ffc.uflacs.analysis.expr_shapes), 20
compute_argument_factorization() (in module ffc.uflacs.analysis.factorization), 20
compute_basis_values() (in module ffc.uflacs.backends.ufc.finite_element), 34
compute_cache_scores() (in module ffc.uflacs.analysis.graph_ssa), 23
compute_dependencies() (in module ffc.uflacs.analysis.dependencies), 19
compute_dependency_count() (in module ffc.uflacs.analysis.graph_ssa), 23
compute_geometry() (ffc.uflacs.backends.ufc.coordinate_mapping.ufc_coordinate_mapping method), 32
compute_index_shape() (in module ffc.uflacs.analysis.expr_shapes), 20
compute_integral_ir() (in module ffc.quadrature.quadraturerepresentation), 10
compute_integral_ir() (in module ffc.tsfc.tsfcrepresentation), 18
compute_integral_ir() (in module ffc.uflacs.uflacsrepresentation), 64
compute_ir() (in module ffc.representation), 69
compute_jacobian_determinants() (ffc.uflacs.backends.ufc.coordinate_mapping.ufc_coordinate_mapping method), 32
compute_jacobian_inverses() (ffc.uflacs.backends.ufc.coordinate_mapping.ufc_coordinate_mapping method), 32
compute_jacobians() (ffc.uflacs.backends.ufc.coordinate_mapping.ufc_coordinate_mapping method), 32
compute_jit_parameters_signature() (in module ffc.parameters), 69
compute_jit_prefix() (in module ffc.jitcompiler), 67
compute_midpoint_geometry() (ffc.uflacs.backends.ufc.coordinate_mapping.ufc_coordinate_mapping method), 32
compute_permutations() (in module ffc.utils), 70
compute_physical_coordinates() (ffc.uflacs.backends.ufc.coordinate_mapping.ufc_coordinate_mapping method), 32
compute_quadrature_rules() (in module ffc.uflacs.tools), 63
compute_reference_coordinates() (ffc.uflacs.backends.ufc.coordinate_mapping.ufc_coordinate_mapping method), 32
compute_scalar_subexpressions() (ffc.uflacs.backends.ufc.coordinate_mapping.ufc_coordinate_mapping method), 32
compute_values() (in module ffc.uflacs.backends.ufc.finite_element), 34
conditional (ffc.uflacs.language.cnodes.Symbol attribute), 17
condition (ffc.uflacs.language.cnodes.Conditional attribute), 43
condition (ffc.uflacs.language.cnodes.Do attribute), 44
condition (ffc.uflacs.language.cnodes.ElseIf attribute), 44
condition (ffc.uflacs.language.cnodes.If attribute), 45
condition (ffc.uflacs.language.cnodes.While attribute), 51
condition() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 13
condition() (ffc.uflacs.analysis.graph_rebuild.ReconstructScalarSubexpressions method), 22
Conditional (class in ffc.uflacs.language.cnodes), 43
CONDITIONAL (ffc.uflacs.language.precedence.PRECEDENCE attribute), 54
conditional() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerBase method), 8

conditional() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 11
conditional() (ffc.uflacs.analysis.graph_rebuild.ReconstructScalarSubexpressions method), 22
conditional() (ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin method), 55
constant_value() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 13
constant_value() (ffc.uflacs.backends.ffc.definitions.FFCBackendDefinitions method), 29
constructor() (ffc.uflacs.backends.ufc.generator.ufc_generator method), 37
constructor_arguments() (ffc.uflacs.backends.ufc.generator.ufc_generator method), 37
contains_zeros() (in module ffc.quadrature.quadratureutils), 15
Continue (class in ffc.uflacs.language.cnodes), 43
cos() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 13
cos() (ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin method), 55
cosh() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 13
cosh() (ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin method), 55
count_nodes_with_unique_post_traversal() (in module ffc.uflacs.analysis.graph_vertices), 24
count_ops() (in module ffc.quadrature.cpp), 6
create() (ffc.uflacs.backends.ufc.generator.ufc_generator method), 37
create_argument() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt method), 9
create_argument() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 11
create_cell_integral() (ffc.uflacs.backends.ufc.form.ufc_form method), 35
create_coordinate_dofmap() (ffc.uflacs.backends.ufc.coordinate_mapping.ufc_coordinate_mapping method), 32
create_coordinate_dofmap() (ffc.uflacs.backends.ufc.form.ufc_form method), 35
create_coordinate_finite_element() (ffc.uflacs.backends.ufc.coordinate_mapping.ufc_coordinate_mapping method), 32
create_coordinate_finite_element() (ffc.uflacs.backends.ufc.form.ufc_form method), 35
create_coordinate_mapping() (ffc.uflacs.backends.ufc.form.ufc_form method), 35
create_custom_integral() (ffc.uflacs.backends.ufc.form.ufc_form method), 35
create_default_cell_integral() (ffc.uflacs.backends.ufc.form.ufc_form method), 35
create_default_exterior_facet_integral() (ffc.uflacs.backends.ufc.form.ufc_form method), 35
create_default_interface_integral() (ffc.uflacs.backends.ufc.form.ufc_form method), 35
create_default_custom_integral() (ffc.uflacs.backends.ufc.form.ufc_form method), 35
create_default_cutcell_integral() (ffc.uflacs.backends.ufc.form.ufc_form method), 36
create_default_exterior_facet_integral() (ffc.uflacs.backends.ufc.form.ufc_form method), 36
create_default_interface_integral() (ffc.uflacs.backends.ufc.form.ufc_form method), 36
create_default_interior_facet_integral() (ffc.uflacs.backends.ufc.form.ufc_form method), 36
create_default_overlap_integral() (ffc.uflacs.backends.ufc.form.ufc_form method), 36
create_default_vertex_integral() (ffc.uflacs.backends.ufc.form.ufc_form method), 36
create_delegate() (in module ffc.uflacs.backends.ufc.form), 35
create_dofmap() (ffc.uflacs.backends.ufc.form.ufc_form method), 36
create_element() (ffc.uflacs.backends.ufc.finite_element.ufc_finite_element method), 34
create_exterior_facet_integral() (ffc.uflacs.backends.ufc.form.ufc_form method), 36
create_finite_element() (ffc.uflacs.backends.ufc.form.ufc_form method), 36
create_float() (in module ffc.quadrature.symbolics), 17
create_fraction() (in module ffc.quadrature.symbolics), 17
create_function() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt method), 9
create_function() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 11
create_interface_integral() (ffc.uflacs.backends.ufc.form.ufc_form method), 36
create_interior_facet_integral() (ffc.uflacs.backends.ufc.form.ufc_form method), 36
create_overlap_integral() (ffc.uflacs.backends.ufc.form.ufc_form method), 36
create_permutations() (in module ffc.quadrature.quadratureutils), 15

create_product() (in module ffc.quadrature.symbolics), 17
create_psi_tables() (in module ffc.quadrature.quadratureutils), 15
create_quadrature() (in module ffc.fiatinterface), 66
create_quadrature_points_and_weights() (in module ffc.representationutils), 70
create_sub_dofmap() (ffc.uflacs.backends.ufc.dofmap.ufc_dofmap method), 32
create_sub_element() (ffc.uflacs.backends.ufc.finite_element.ufc_finite_element method), 34
create_sum() (in module ffc.quadrature.symbolics), 18
create_symbol() (in module ffc.quadrature.symbolics), 18
create_vertex_integral() (ffc.uflacs.backends.ufc.form.ufc_form method), 36
cross_expr() (in module ffc.uflacs.backends.ufc.coordinate_mapping), 31
CRSArray (class in ffc.uflacs.analysis.crsarray), 19
cs_format() (ffc.uflacs.language.cnodes.ArrayDecl method), 40
cs_format() (ffc.uflacs.language.cnodes.Break method), 42
cs_format() (ffc.uflacs.language.cnodes.Case method), 43
cs_format() (ffc.uflacs.language.cnodes.Comment method), 43
cs_format() (ffc.uflacs.language.cnodes.Continue method), 43
cs_format() (ffc.uflacs.language.cnodes.CStatement method), 43
cs_format() (ffc.uflacs.language.cnodes.Default method), 44
cs_format() (ffc.uflacs.language.cnodes.Do method), 44
cs_format() (ffc.uflacs.language.cnodes.Else method), 44
cs_format() (ffc.uflacs.language.cnodes.ElseIf method), 44
cs_format() (ffc.uflacs.language.cnodes.For method), 45
cs_format() (ffc.uflacs.language.cnodes.ForRange method), 45
cs_format() (ffc.uflacs.language.cnodes.If method), 46
cs_format() (ffc.uflacs.language.cnodes.Namespace method), 47
cs_format() (ffc.uflacs.language.cnodes.Pragma method), 48
cs_format() (ffc.uflacs.language.cnodes.Return method), 49
cs_format() (ffc.uflacs.language.cnodes.Scope method), 49
cs_format() (ffc.uflacs.language.cnodes.Statement method), 49
cs_format() (ffc.uflacs.language.cnodes.StatementList method), 50
cs_format() (ffc.uflacs.language.cnodes.Switch method), 50
cs_format() (ffc.uflacs.language.cnodes.Throw method), 50
cs_format() (ffc.uflacs.language.cnodes.Using method), 51
cs_format() (ffc.uflacs.language.cnodes.VariableDecl method), 51
cs_format() (ffc.uflacs.language.cnodes.VerbatimStatement method), 51
cs_format() (ffc.uflacs.language.cnodes.While method), 52
CStatement (class in ffc.uflacs.language.cnodes), 42
custom_points_table() (ffc.uflacs.backends.ffc.symbols.FFCBackendSymbo method), 30
custom_quadrature_points() (ffc.uflacs.backends.ffc.symbols.FFCBackendSymbols method), 30
custom_quadrature_weights() (ffc.uflacs.backends.ffc.symbols.FFCBackendSymbols method), 30
custom_weights_table() (ffc.uflacs.backends.ffc.symbols.FFCBackendSymb method), 30

D

debug (ffc.uflacs.language.cnodes.CNode attribute), 42
debug() (in module ffc.log), 67
debug_code() (in module ffc.log), 67
debug_dict() (in module ffc.log), 67
debug_ir() (in module ffc.log), 67
Default (class in ffc.uflacs.language.cnodes), 43
default (ffc.uflacs.language.cnodes.Switch attribute), 50
default_cache_score_policy() (in module ffc.uflacs.analysis.graph_ssa), 23
default_jit_parameters() (in module ffc.parameters), 69
default_optimize_parameters() (in module ffc.quadrature.parameters), 9
default_parameters() (in module ffc.parameters), 69
default_parameters() (in module ffc.uflacs.params), 63
default_partition_seed() (in module ffc.uflacs.analysis.graph_ssa), 23
degree() (ffc.uflacs.backends.ufc.finite_element.ufc_finite_element method), 34
denom (ffc.quadrature.fraction.Fraction attribute), 8
deprecate() (in module ffc.log), 68
Dereference (class in ffc.uflacs.language.cnodes), 44
DEREFERENCE (ffc.uflacs.language.precedence.PRECEDENCE attribute), 54
derivative() (ffc.quadrature.quadraturetransformerbase.QuadratureTransform method), 13
derivatives() (ffc.quadrature.quadraturetransformerbase.QuadratureTransfor method), 13
destructor() (ffc.uflacs.backends.ufc.generator.ufc_generator method), 37
det_22() (in module ffc.uflacs.backends.ufc.coordinate_mapping), 31

det_nn() (in module ffc.uflacs.backends.ufc.coordinate_mapping), 31
dims (ffc.uflacs.language.cnodes.FlattenedArray attribute), 45
disp() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 13
Div (class in ffc.uflacs.language.cnodes), 44
DIV (ffc.uflacs.language.precedence.PRECEDENCE attribute), 54
division() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt method), 9
division() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 11
division() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 13
division() (ffc.uflacs.analysis.graph_rebuild.ReconstructScalarSubexpressions method), 22
division() (ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin method), 55
Do (class in ffc.uflacs.language.cnodes), 44
domain_dof_access() (ffc.uflacs.backends.ffc.symbols.FFCBackendSymbols method), 30
domain_dofs_access() (ffc.uflacs.backends.ffc.symbols.FFCBackendSymbols method), 30
dual_forms() (ffc.errorcontrol.errorcontrolgenerators.ErrorControlGenerator method), 5

E

element_table() (ffc.uflacs.backends.ffc.symbols.FFCBackendSymbols method), 30
element_tensor() (ffc.uflacs.backends.ffc.symbols.FFCBackendSymbols method), 30
Else (class in ffc.uflacs.language.cnodes), 44
ElseIf (class in ffc.uflacs.language.cnodes), 44
empty_expr_ir() (in module ffc.uflacs.build_uflacs_ir), 57
enabled_coefficients() (ffc.uflacs.backends.ufc.integrals.ufc_integral method), 38
end (ffc.uflacs.language.cnodes.ForRange attribute), 45
end() (in module ffc.log), 68
entity() (ffc.uflacs.backends.ffc.symbols.FFCBackendSymbols method), 30
entity_type_from_integral_type() (in module ffc.representationutils), 70
EQ (class in ffc.uflacs.language.cnodes), 44
EQ (ffc.uflacs.language.precedence.PRECEDENCE attribute), 54
eq() (ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin method), 55
equal_tables() (in module ffc.uflacs.elementtables), 60
erf() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 13
erf() (ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin method), 55
error() (in module ffc.log), 68
error() (ffc.errorcontrol.errorcontrolgenerators.ErrorControlGenerator method), 5
error_indicators() (ffc.errorcontrol.errorcontrolgenerators.ErrorControlGenerator method), 5
ErrorControlGenerator (class in ffc.errorcontrol.errorcontrolgenerators), 5
evaluate_basis() (ffc.uflacs.backends.ufc.finite_element.ufc_finite_element method), 34
evaluate_basis_all() (ffc.uflacs.backends.ufc.finite_element.ufc_finite_element method), 34
evaluate_basis_derivatives() (ffc.uflacs.backends.ufc.finite_element.ufc_finite_element method), 34
evaluate_basis_derivatives_all() (ffc.uflacs.backends.ufc.finite_element.ufc_finite_element method), 34
evaluate_dof() (ffc.uflacs.backends.ufc.finite_element.ufc_finite_element method), 34
evaluate_dofs() (ffc.uflacs.backends.ufc.finite_element.ufc_finite_element method), 34
evaluate_reference_basis() (ffc.uflacs.backends.ufc.finite_element.ufc_finite_element method), 34
evaluate_reference_basis_derivatives() (ffc.uflacs.backends.ufc.finite_element.ufc_finite_element method), 34
exception (ffc.uflacs.language.cnodes.Throw attribute), 50
exp (ffc.quadrature.symbol.Symbol attribute), 17
exp() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 13
exp() (ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin method), 55
expand() (ffc.quadrature.expr.Expr method), 7
expand() (ffc.quadrature.fraction.Fraction method), 8
expand() (ffc.quadrature.product.Product method), 10
expand() (ffc.quadrature.sumobj.Sum method), 17
expand_operations() (in module ffc.quadrature.reduce_operations), 15
Expr (class in ffc.quadrature.expr), 7
expr (ffc.uflacs.language.cnodes.Statement attribute), 49
expr() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 13
expr() (ffc.uflacs.analysis.balancing.BalanceModifiers method), 19
expr() (ffc.uflacs.analysis.graph_rebuild.ReconstructScalarSubexpressions method), 22
expr() (ffc.uflacs.analysis.valuenumbering.ValueNumberer method), 25
expr() (ffc.uflacs.backends.ffc.access.FFCBackendAccess method), 27
expr() (ffc.uflacs.backends.ffc.definitions.FFCBackendDefinitions method), 29
expr() (ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin

method), 55
expr_list() (ffc.uflacs.analysis.graph_rebuild.ReconstructScalarSubexpressions method), 22
expr_mapping() (ffc.uflacs.analysis.graph_rebuild.ReconstructScalarSubexpressions method), 22
extract_keywords() (in module ffc.uflacs.backends.ufc.templates), 39

F

facet_area() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt method), 9
facet_area() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 11
facet_area() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 13
facet_avg() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 13
facet_avg() (ffc.uflacs.analysis.balancing.BalanceModifiers method), 19
facet_avg() (ffc.uflacs.analysis.valuenumbering.ValueNumberer method), 25
facet_coordinate() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt method), 9
facet_coordinate() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 11
facet_coordinate() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 13
facet_coordinate() (ffc.uflacs.backends.ffc.access.FFCBackendAccess method), 27
facet_edge_vectors() (ffc.uflacs.backends.ffc.access.FFCBackendAccess method), 27
facet_edge_vectors() (ffc.uflacs.backends.ffc.definitions.FFCBackendDefinitions method), 29
facet_jacobian() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt method), 9
facet_jacobian() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 11
facet_jacobian() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 13
facet_jacobian() (ffc.uflacs.backends.ffc.access.FFCBackendAccess method), 27
facet_jacobian() (ffc.uflacs.backends.ffc.definitions.FFCBackendDefinitions method), 29
facet_jacobian_determinant() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt method), 9
facet_jacobian_determinant() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 11
facet_jacobian_determinant() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 13
facet_jacobian_determinant() (ffc.uflacs.backends.ffc.access.FFCBackendAccess method), 27
facet_jacobian_determinant() (ffc.uflacs.backends.ffc.definitions.FFCBackendDefinitions method), 29
facet_jacobian_inverse() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt method), 9
facet_jacobian_inverse() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 11
facet_jacobian_inverse() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 13
facet_jacobian_inverse() (ffc.uflacs.backends.ffc.access.FFCBackendAccess method), 27
facet_jacobian_inverse() (ffc.uflacs.backends.ffc.definitions.FFCBackendDefinitions method), 29
facet_normal() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt method), 9
facet_normal() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 11
facet_normal() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 13
facet_normal() (ffc.uflacs.backends.ffc.access.FFCBackendAccess method), 27
facet_normal() (ffc.uflacs.backends.ffc.definitions.FFCBackendDefinitions method), 29
facet_orientation() (ffc.uflacs.backends.ffc.access.FFCBackendAccess method), 27
facet_orientation() (ffc.uflacs.backends.ffc.definitions.FFCBackendDefinitions method), 29
facet_origin() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt method), 9
facet_origin() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 11
facet_origin() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 13
facet_residual() (ffc.errorcontrol.errorcontrolgenerators.ErrorControlGenerator method), 5
factor_index (ffc.uflacs.build_uflacs_ir.common_block_data_t attribute), 57
factor_index (ffc.uflacs.build_uflacs_ir.full_block_data_t attribute), 57
factor_index (ffc.uflacs.build_uflacs_ir.partial_block_data_t attribute), 58
factor_index (ffc.uflacs.build_uflacs_ir.preintegrated_block_data_t attribute), 59
factor_index (ffc.uflacs.build_uflacs_ir.premultiplied_block_data_t attribute), 59
factor_is_piecewise (ffc.uflacs.build_uflacs_ir.common_block_data_t attribute), 57
factor_is_piecewise (ffc.uflacs.build_uflacs_ir.full_block_data_t attribute), 57
factor_is_piecewise (ffc.uflacs.build_uflacs_ir.partial_block_data_t attribute), 58
factor_is_piecewise (ffc.uflacs.build_uflacs_ir.preintegrated_block_data_t attribute), 59

factor_is_piecewise (ffc.uflacs.build_uflacs_ir.premultiplied_fo attribute), 59
Factors (class in ffc.uflacs.analysis.factorization), 20
false (ffc.uflacs.language.cnodes.Conditional attribute), 43
family() (ffc.uflacs.backends.ufc.finite_element.ufc_finite_elem method), 34
ffc (module), 71
ffc.analysis (module), 64
ffc.classname (module), 64
ffc.codegeneration (module), 64
ffc.compiler (module), 65
ffc.errorcontrol (module), 6
ffc.errorcontrol.errorcontrol (module), 5
ffc.errorcontrol.errorcontrolgenerators (module), 5
ffc.fiatinterface (module), 66
ffc.formatting (module), 66
ffc.git_commit_hash (module), 67
ffc.jitcompiler (module), 67
ffc.log (module), 67
ffc.main (module), 68
ffc.optimization (module), 69
ffc.parameters (module), 69
ffc.plot (module), 69
ffc.quadrature (module), 18
ffc.quadrature.codesnippets (module), 6
ffc.quadrature.cpp (module), 6
ffc.quadrature.deprecation (module), 7
ffc.quadrature.expr (module), 7
ffc.quadrature.floatvalue (module), 7
ffc.quadrature.fraction (module), 7
ffc.quadrature.optimisedquadraturetransformer (module), 8
ffc.quadrature.parameters (module), 9
ffc.quadrature.product (module), 9
ffc.quadrature.quadraturegenerator (module), 10
ffc.quadrature.quadratureoptimization (module), 10
ffc.quadrature.quadraturerepresentation (module), 10
ffc.quadrature.quadraturetransformer (module), 11
ffc.quadrature.quadraturetransformerbase (module), 12
ffc.quadrature.quadratureutils (module), 15
ffc.quadrature.reduce_operations (module), 15
ffc.quadrature.sumobj (module), 16
ffc.quadrature.symbol (module), 17
ffc.quadrature.symbolics (module), 17
ffc.quadrature.tabulate_basis (module), 18
ffc.representation (module), 69
ffc.representationutils (module), 70
ffc.tsfc.tsfcoptimization (module), 18
ffc.tsfc.tsfcrepresentation (module), 18
ffc.uflacs (module), 64
ffc.uflacs.analysis (module), 26
ffc.uflacs.analysis.balancing (module), 19
ffc.uflacs.analysis.crsarray (module), 19
ffc.uflacs.analysis.dependencies (module), 19
ffc.uflacs.analysis.expr_shapes (module), 20
ffc.uflacs.analysis.factorization (module), 20
ffc.uflacs.analysis.graph (module), 21
ffc.uflacs.analysis.graph_rebuild (module), 21
ffc.uflacs.analysis.graph_ssa (module), 22
ffc.uflacs.analysis.graph_symbols (module), 23
ffc.uflacs.analysis.graph_vertices (module), 24
ffc.uflacs.analysis.indexing (module), 24
ffc.uflacs.analysis.modified_terminals (module), 24
ffc.uflacs.analysis.valuenumbering (module), 25
ffc.uflacs.backends (module), 39
ffc.uflacs.backends.ffc (module), 31
ffc.uflacs.backends.ffc.access (module), 27
ffc.uflacs.backends.ffc.backend (module), 28
ffc.uflacs.backends.ffc.common (module), 28
ffc.uflacs.backends.ffc.definitions (module), 28
ffc.uflacs.backends.ffc.symbols (module), 30
ffc.uflacs.backends.ufc (module), 39
ffc.uflacs.backends.ufc.apply_mappings (module), 31
ffc.uflacs.backends.ufc.coordinate_mapping (module), 31
ffc.uflacs.backends.ufc.dofmap (module), 32
ffc.uflacs.backends.ufc.evalderivs (module), 33
ffc.uflacs.backends.ufc.evaluatebasis (module), 33
ffc.uflacs.backends.ufc.evaluatebasisderivatives (module), 33
ffc.uflacs.backends.ufc.evaluatedof (module), 34
ffc.uflacs.backends.ufc.finite_element (module), 34
ffc.uflacs.backends.ufc.form (module), 35
ffc.uflacs.backends.ufc.generator (module), 37
ffc.uflacs.backends.ufc.generators (module), 38
ffc.uflacs.backends.ufc.integrals (module), 38
ffc.uflacs.backends.ufc.jacobian (module), 38
ffc.uflacs.backends.ufc.templates (module), 39
ffc.uflacs.backends.ufc.utils (module), 39
ffc.uflacs.build_uflacs_ir (module), 56
ffc.uflacs.elementtables (module), 60
ffc.uflacs.integralgenerator (module), 62
ffc.uflacs.language (module), 56
ffc.uflacs.language.cnodes (module), 39
ffc.uflacs.language.format_lines (module), 53
ffc.uflacs.language.format_value (module), 53
ffc.uflacs.language.precedence (module), 54
ffc.uflacs.language.ufl_to_cnodes (module), 55
ffc.uflacs.params (module), 63
ffc.uflacs.tools (module), 63
ffc.uflacs.uflacsgenerator (module), 63
ffc.uflacs.uflacsoptimization (module), 63
ffc.uflacs.uflacsrepresentation (module), 64
ffc.utils (module), 70
ffc.wrappers (module), 71
ffc_assert() (in module ffc.log), 68
FFCBackend (class in ffc.uflacs.backends.ffc.backend), 28

FFCBackendAccess (class in ffc.uflacs.backends.ffc.access), 27
FFCBackendDefinitions (class in ffc.uflacs.backends.ffc.definitions), 28
FFCBackendSymbols (class in ffc.uflacs.backends.ffc.symbols), 30
FFCError, 67
FFCJitError, 67
FFCMultiIndex (class in ffc.quadrature.quadraturetransformerbase), 12
fiat_coordinate_mapping() (in module ffc.uflacs.backends.ufc.jacobian), 38
firstkey() (in module ffc.quadrature.optimisedquadraturetransformer), 9
firstkey() (in module ffc.quadrature.quadraturetransformer), 12
flatten_psi_tables() (in module ffc.quadrature.quadratureutils), 15
flattened_indices() (in module ffc.uflacs.language.cnodes), 52
FlattenedArray (class in ffc.uflacs.language.cnodes), 44
float_product() (in module ffc.uflacs.language.cnodes), 52
float_value() (ffc.uflacs.backends.ffc.access.FFCBackendAccess method), 27
float_value() (ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin method), 56
FloatValue (class in ffc.quadrature.floatvalue), 7
For (class in ffc.uflacs.language.cnodes), 45
form_argument() (ffc.uflacs.analysis.valuenumbering.ValueNumberer method), 26
format_code() (in module ffc.formatting), 66
format_float() (in module ffc.uflacs.language.format_value), 53
format_indented_lines() (in module ffc.uflacs.language.format_lines), 53
format_int() (in module ffc.uflacs.language.format_value), 53
format_mt_name() (in module ffc.uflacs.backends.ffc.symbols), 31
format_value() (in module ffc.uflacs.language.format_value), 53
ForRange (class in ffc.uflacs.language.cnodes), 45
ForRanges() (in module ffc.uflacs.language.cnodes), 45
Fraction (class in ffc.quadrature.fraction), 7
from_rows() (ffc.uflacs.analysis.crsarray.CRSArray class method), 19
full_block_data_t (class in ffc.uflacs.build_uflacs_ir), 57
function (ffc.uflacs.language.cnodes.Call attribute), 43

G

GE (class in ffc.uflacs.language.cnodes), 45
GE (ffc.uflacs.language.precedence.PRECEDENCE attribute), 54
ge() (ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin method), 56
generate() (ffc.uflacs.backends.ufc.generator.ufc_generator method), 37
generate() (ffc.uflacs.integralgenerator.IntegralGenerator method), 62
generate_all_error_control_forms() (ffc.errorcontrol.errorcontrolgenerators.ErrorControlGenerator method), 5
generate_assign_inverse() (in module ffc.uflacs.backends.ufc.coordinate_mapping), 31
generate_aux_constants() (in module ffc.quadrature.symbolics), 18
generate_block_parts() (ffc.uflacs.integralgenerator.IntegralGenerator method), 62
generate_code() (in module ffc.codegeneration), 64
generate_compute_ATA() (in module ffc.uflacs.backends.ufc.coordinate_mapping), 31
generate_compute_basisvalues() (in module ffc.uflacs.backends.ufc.evaluatebasis), 33
generate_copyout_statements() (ffc.uflacs.integralgenerator.IntegralGenerator method), 62
generate_cross_decl() (in module ffc.uflacs.backends.ufc.coordinate_mapping), 31
generate_dofblock_partition() (ffc.uflacs.integralgenerator.IntegralGenerator method), 62
generate_element_mapping() (in module ffc.uflacs.backends.ufc.finite_element), 34
generate_element_tables() (ffc.uflacs.integralgenerator.IntegralGenerator method), 62
generate_enabled_coefficients() (in module ffc.representationutils), 70
generate_error() (in module ffc.uflacs.backends.ufc.utils), 39
generate_evaluate_basis_derivatives() (in module ffc.uflacs.backends.ufc.evaluatebasisderivatives), 33
generate_evaluate_basis_derivatives_all() (in module ffc.uflacs.backends.ufc.evaluatebasisderivatives), 34
generate_evaluate_dof() (in module ffc.uflacs.backends.ufc.evaluatedof), 34
generate_evaluate_dofs() (in module ffc.uflacs.backends.ufc.evaluatedof), 34
generate_evaluate_reference_basis() (in module ffc.uflacs.backends.ufc.evaluatebasis), 33

generate_evaluate_reference_basis_derivatives() (in module ffc.uflacs.backends.ufc.evalderivs), 33
generate_expansion_coefficients() (in module ffc.uflacs.backends.ufc.evaluatebasis), 33
generate_expr_copyout_statements() (ffc.uflacs.integralgenerator.IntegralGenerator method), 62
generate_factory_functions() (in module ffc.formatting), 66
generate_integral_code() (in module ffc.quadrature.quadraturegenerator), 10
generate_integral_code() (in module ffc.uflacs.uflacsgenerator), 63
generate_jit_factory_functions() (in module ffc.formatting), 67
generate_partition() (ffc.uflacs.integralgenerator.IntegralGenerator method), 62
generate_preintegrated_dofblock_partition() (ffc.uflacs.integralgenerator.IntegralGenerator method), 62
generate_psi_table_name() (in module ffc.uflacs.elementtables), 60
generate_quadrature_loop() (ffc.uflacs.integralgenerator.IntegralGenerator method), 62
generate_quadrature_tables() (ffc.uflacs.integralgenerator.IntegralGenerator method), 62
generate_return_bool_switch() (in module ffc.uflacs.backends.ufc.utils), 39
generate_return_int_switch() (in module ffc.uflacs.backends.ufc.utils), 39
generate_return_literal_switch() (in module ffc.uflacs.backends.ufc.utils), 39
generate_return_new() (in module ffc.uflacs.backends.ufc.utils), 39
generate_return_new_switch() (in module ffc.uflacs.backends.ufc.utils), 39
generate_return_sizet_switch() (in module ffc.uflacs.backends.ufc.utils), 39
generate_runtime_quadrature_loop() (ffc.uflacs.integralgenerator.IntegralGenerator method), 62
generate_snippets() (ffc.uflacs.backends.ufc.generator.ufc_generator method), 37
generate_tabulate_dmats() (in module ffc.uflacs.backends.ufc.evalderivs), 33
generate_tensor_copyout_statements() (ffc.uflacs.integralgenerator.IntegralGenerator method), 62
generate_tensor_reset() (ffc.uflacs.integralgenerator.IntegralGenerator method), 62
generate_tensor_value_initialization() (ffc.uflacs.integralgenerator.IntegralGenerator method), 62
generate_terms() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 13
generate_unstructured_piecewise_partition() (ffc.uflacs.integralgenerator.IntegralGenerator method), 62
generate_unstructured_varying_partition() (ffc.uflacs.integralgenerator.IntegralGenerator method), 62
generate_wrapper_code() (in module ffc.wrappers), 71
geometric_dimension() (ffc.uflacs.backends.ufc.coordinate_mapping.ufc_coordinate_mapping method), 32
geometric_dimension() (ffc.uflacs.backends.ufc.finite_element.ufc_finite_element method), 34
geometric_quantity() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 14
get_arg_factors() (ffc.uflacs.integralgenerator.IntegralGenerator method), 62
get_common_block_data() (in module ffc.uflacs.build_uflacs_ir), 58
get_constants() (in module ffc.quadrature.reduce_operations), 15
get_entities() (ffc.uflacs.integralgenerator.IntegralGenerator method), 62
get_ffc_table_values() (in module ffc.uflacs.elementtables), 61
get_geo_terms() (in module ffc.quadrature.reduce_operations), 16
get_handler() (in module ffc.log), 68
get_includes() (ffc.uflacs.integralgenerator.IntegralGenerator method), 62
get_indices() (in module ffc.quadrature.reduce_operations), 16
get_logger() (in module ffc.log), 68
get_modified_terminal_element() (in module ffc.uflacs.elementtables), 61
get_node_symbols() (ffc.uflacs.analysis.valuenumbering.ValueNumberer method), 26
get_ones() (in module ffc.quadrature.quadratureutils), 15
get_simple_variables() (in module ffc.quadrature.reduce_operations), 16
get_temp_symbol() (ffc.uflacs.integralgenerator.IntegralGenerator method), 62
get_unique_vars() (ffc.quadrature.expr.Expr method), 7
get_unique_vars() (ffc.quadrature.fraction.Fraction method), 8
get_unique_vars() (ffc.quadrature.product.Product method), 10
get_unique_vars() (ffc.quadrature.sumobj.Sum method), 17
get_unique_vars() (ffc.quadrature.symbol.Symbol method), 17
get_var() (ffc.uflacs.integralgenerator.IntegralGenerator method), 62

get_var_occurrences() (ffc.quadrature.expr.Expr method), 7
get_var_occurrences() (ffc.quadrature.fraction.Fraction method), 8
get_var_occurrences() (ffc.quadrature.product.Product method), 10
get_var_occurrences() (ffc.quadrature.sumobj.Sum method), 17
get_var_occurrences() (ffc.quadrature.symbol.Symbol method), 17
get_variables() (in module ffc.quadrature.reduce_operations), 16
get_vrs() (ffc.quadrature.product.Product method), 10
git_commit_hash() (in module ffc.git_commit_hash), 67
global_dimension() (ffc.uflacs.backends.ufc.dofmap.ufc_dofmap method), 32
grad() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 14
grad() (ffc.uflacs.analysis.balancing.BalanceModifiers method), 19
grad() (ffc.uflacs.analysis.valuenumbering.ValueNumberer method), 26
Graph2 (class in ffc.uflacs.analysis.graph), 21
group_vars() (in module ffc.quadrature.reduce_operations), 16
GT (class in ffc.uflacs.language.cnodes), 45
GT (ffc.uflacs.language.precedence.PRECEDENCE attribute), 54
gt() (ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin method), 56

H

handle_conditional() (in module ffc.uflacs.analysis.factorization), 21
handle_division() (in module ffc.uflacs.analysis.factorization), 21
handle_operator() (in module ffc.uflacs.analysis.factorization), 21
handle_product() (in module ffc.uflacs.analysis.factorization), 21
handle_sum() (in module ffc.uflacs.analysis.factorization), 21
has_cell_integrals() (ffc.uflacs.backends.ufc.form.ufc_form method), 36
has_custom_integrals() (ffc.uflacs.backends.ufc.form.ufc_form method), 36
has_cutcell_integrals() (ffc.uflacs.backends.ufc.form.ufc_form method), 36
has_exterior_facet_integrals() (ffc.uflacs.backends.ufc.form.ufc_form method), 36
has_interface_integrals() (ffc.uflacs.backends.ufc.form.ufc_form method), 36
has_interior_facet_integrals() (ffc.uflacs.backends.ufc.form.ufc_form method), 36
has_overlap_integrals() (ffc.uflacs.backends.ufc.form.ufc_form method), 36
has_var() (ffc.uflacs.integralgenerator.IntegralGenerator method), 63
has_vertex_integrals() (ffc.uflacs.backends.ufc.form.ufc_form method), 36
HIGHEST (ffc.uflacs.language.precedence.PRECEDENCE attribute), 54

I

identity() (ffc.quadrature.quadraturetransformerbase.QuadratureTransforme method), 14
If (class in ffc.uflacs.language.cnodes), 45
indent() (in module ffc.quadrature.cpp), 6
Indented (class in ffc.uflacs.language.format_lines), 53
index (ffc.uflacs.language.cnodes.ForRange attribute), 45
index_sum() (ffc.quadrature.quadraturetransformerbase.QuadratureTransfor method), 14
index_sum() (ffc.uflacs.analysis.graph_rebuild.ReconstructScalarSubexpres method), 22
index_type (ffc.uflacs.language.cnodes.ForRange attribute), 45
indexed() (ffc.quadrature.quadraturetransformerbase.QuadratureTransforme method), 14
indexed() (ffc.uflacs.analysis.valuenumbering.ValueNumberer method), 26
indices (ffc.uflacs.language.cnodes.ArrayAccess attribute), 40
info() (in module ffc.log), 68
info_blue() (in module ffc.log), 68
info_green() (in module ffc.log), 68
info_red() (in module ffc.log), 68
info_usage() (in module ffc.main), 68
info_version() (in module ffc.main), 68
init (ffc.uflacs.language.cnodes.For attribute), 45
init_scopes() (ffc.uflacs.integralgenerator.IntegralGenerator method), 63
initialize_data() (ffc.errorcontrol.errorcontrolgenerators.ErrorControlGenera method), 5
initialize_data() (ffc.errorcontrol.errorcontrolgenerators.UFLErrorControlG method), 6
initialize_integral_code() (in module ffc.representationutils), 70
initialize_integral_ir() (in module ffc.representationutils), 70
initializer_list() (ffc.uflacs.backends.ufc.generator.ufc_generator method), 38
insert_nested_dict() (in module ffc.utils), 70
int_value() (ffc.uflacs.backends.ffc.access.FFCBackendAccess method), 27

int_value() (ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin method), 56
integral_type_to_entity_dim() (in module ffc.representationutils), 70
IntegralGenerator (class in ffc.uflacs.integralgenerator), 62
integrate_block() (in module ffc.uflacs.build_uflacs_ir), 58
integrate_block_interior_facets() (in module ffc.uflacs.build_uflacs_ir), 58
interpolate_vertex_values() (ffc.uflacs.backends.ufc.finite_element.ufc_finite_element method), 34
inverse_jacobian() (in module ffc.uflacs.backends.ufc.jacobian), 38
invert_dependencies() (in module ffc.uflacs.analysis.graph_ssa), 23
is_constant() (in module ffc.quadrature.reduce_operations), 16
is_modified_terminal() (in module ffc.uflacs.analysis.modified_terminals), 25
is_negative_one_cexpr() (in module ffc.uflacs.language.cnodes), 52
is_one_cexpr() (in module ffc.uflacs.language.cnodes), 52
is_ones_table() (in module ffc.uflacs.elementtables), 61
is_piecewise_table() (in module ffc.uflacs.elementtables), 61
is_quadrature_table() (in module ffc.uflacs.elementtables), 61
is_scoped (ffc.uflacs.language.cnodes.ArrayDecl attribute), 40
is_scoped (ffc.uflacs.language.cnodes.Break attribute), 42
is_scoped (ffc.uflacs.language.cnodes.Case attribute), 43
is_scoped (ffc.uflacs.language.cnodes.Comment attribute), 43
is_scoped (ffc.uflacs.language.cnodes.Continue attribute), 43
is_scoped (ffc.uflacs.language.cnodes.CStatement attribute), 43
is_scoped (ffc.uflacs.language.cnodes.Default attribute), 44
is_scoped (ffc.uflacs.language.cnodes.Do attribute), 44
is_scoped (ffc.uflacs.language.cnodes.Else attribute), 44
is_scoped (ffc.uflacs.language.cnodes.ElseIf attribute), 44
is_scoped (ffc.uflacs.language.cnodes.For attribute), 45
is_scoped (ffc.uflacs.language.cnodes.ForRange attribute), 45
is_scoped (ffc.uflacs.language.cnodes.If attribute), 46
is_scoped (ffc.uflacs.language.cnodes.Namespace attribute), 47
is_scoped (ffc.uflacs.language.cnodes.Pragma attribute), 48
is_scoped (ffc.uflacs.language.cnodes.Return attribute), 49
is_scoped (ffc.uflacs.language.cnodes.Scope attribute), 49
is_scoped (ffc.uflacs.language.cnodes.Statement attribute), 49
is_scoped (ffc.uflacs.language.cnodes.StatementList attribute), 50
is_scoped (ffc.uflacs.language.cnodes.Switch attribute), 50
is_scoped (ffc.uflacs.language.cnodes.Throw attribute), 51
is_scoped (ffc.uflacs.language.cnodes.Using attribute), 51
is_scoped (ffc.uflacs.language.cnodes.VariableDecl attribute), 51
is_scoped (ffc.uflacs.language.cnodes.VerbatimStatement attribute), 51
is_scoped (ffc.uflacs.language.cnodes.While attribute), 52
is_simple_inner_loop() (in module ffc.uflacs.language.cnodes), 52
is_uniform (ffc.uflacs.build_uflacs_ir.preintegrated_block_data_t attribute), 59
is_uniform (ffc.uflacs.build_uflacs_ir.premultiplied_block_data_t attribute), 59
is_uniform_table() (in module ffc.uflacs.elementtables), 61
is_zero_cexpr() (in module ffc.uflacs.language.cnodes), 52
is_zeros_table() (in module ffc.uflacs.elementtables), 61
issue_deprecation_warning() (in module ffc.quadrature.deprecation), 7
iter_indented_lines() (in module ffc.uflacs.language.format_lines), 53

J

J_component() (ffc.uflacs.backends.ffc.symbols.FFCBackendSymbols method), 30
jacobian() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTrans method), 9
jacobian() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 11
jacobian() (ffc.quadrature.quadraturetransformerbase.QuadratureTransforme method), 14
jacobian() (ffc.uflacs.backends.ffc.access.FFCBackendAccess method), 27
jacobian() (ffc.uflacs.backends.ffc.definitions.FFCBackendDefinitions method), 29
jacobian() (in module ffc.uflacs.backends.ufc.jacobian), 38
jacobian_determinant() (ffc.quadrature.optimisedquadraturetransformer.Qua method), 9
jacobian_determinant() (ffc.quadrature.quadraturetransformer.QuadratureTr method), 11

jacobian_determinant() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 14
jacobian_determinant() (ffc.uflacs.backends.ffc.access.FFCBackendAccess method), 27
jacobian_determinant() (ffc.uflacs.backends.ffc.definitions.FFCBackendDefinitions method), 29
jacobian_inverse() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt method), 9
jacobian_inverse() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 11
jacobian_inverse() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 14
jacobian_inverse() (ffc.uflacs.backends.ffc.access.FFCBackendAccess method), 27
jacobian_inverse() (ffc.uflacs.backends.ffc.definitions.FFCBackendDefinitions method), 29
jit() (in module ffc.jitcompiler), 67
jit_build() (in module ffc.jitcompiler), 67
jit_generate() (in module ffc.jitcompiler), 67

L

label() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 14
label() (ffc.uflacs.analysis.graph_rebuild.ReconstructScalarSubexpressions method), 22
LE (class in ffc.uflacs.language.cnodes), 46
LE (ffc.uflacs.language.precedence.PRECEDENCE attribute), 54
le() (ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin method), 56
leftover() (in module ffc.uflacs.language.cnodes), 53
lhs (ffc.uflacs.language.cnodes.BinOp attribute), 41
list_tensor() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 14
list_tensor() (ffc.uflacs.analysis.graph_rebuild.ReconstructScalarSubexpressions method), 22
list_tensor() (ffc.uflacs.analysis.valuenumbering.ValueNumberer method), 26
listcopy() (in module ffc.utils), 70
LITERAL (ffc.uflacs.language.precedence.PRECEDENCE attribute), 54
LiteralBool (class in ffc.uflacs.language.cnodes), 46
LiteralFloat (class in ffc.uflacs.language.cnodes), 46
LiteralInt (class in ffc.uflacs.language.cnodes), 46
LiteralString (class in ffc.uflacs.language.cnodes), 46
ln() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 14
ln() (ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin method), 56
log() (in module ffc.log), 68
LOWEST (ffc.uflacs.language.precedence.PRECEDENCE attribute), 54
LT (class in ffc.uflacs.language.cnodes), 46
LT (ffc.uflacs.language.precedence.PRECEDENCE attribute), 54
lt() (ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin method), 56

M

ma_data (ffc.uflacs.build_uflacs_ir.full_block_data_t attribute), 57
ma_data (ffc.uflacs.build_uflacs_ir.partial_block_data_t attribute), 58
ma_data_t (class in ffc.uflacs.build_uflacs_ir), 58
ma_index (ffc.uflacs.build_uflacs_ir.ma_data_t attribute), 58
main() (in module ffc.main), 68
make_all_element_classnames() (in module ffc.representation), 69
make_classname() (in module ffc.classname), 64
make_classname() (in module ffc.quadrature.cpp), 6
make_coordinate_mapping_jit_classname() (in module ffc.representation), 69
make_dofmap_jit_classname() (in module ffc.representation), 69
make_finite_element_jit_classname() (in module ffc.representation), 70
make_integral_classname() (in module ffc.classname), 64
make_integral_classname() (in module ffc.quadrature.cpp), 6
map_component_tensor_arg_components() (in module ffc.uflacs.analysis.indexing), 24
map_facet_points() (in module ffc.fiatinterface), 66
map_indexed_arg_components() (in module ffc.uflacs.analysis.indexing), 24
map_integral_points() (in module ffc.representationutils), 70
mark_active() (in module ffc.uflacs.analysis.dependencies), 19
mark_image() (in module ffc.uflacs.analysis.dependencies), 20
mark_partitions() (in module ffc.uflacs.analysis.graph_ssa), 23
math_function() (ffc.quadrature.quadraturetransformerbase.QuadratureTran method), 14
math_function() (ffc.uflacs.analysis.graph_rebuild.ReconstructScalarSubexp method), 22
math_function() (ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin method), 56
max_cell_subdomain_id() (ffc.uflacs.backends.ufc.form.ufc_form method), 36
max_custom_subdomain_id() (ffc.uflacs.backends.ufc.form.ufc_form method), 37
max_cutcell_subdomain_id() (ffc.uflacs.backends.ufc.form.ufc_form

method), 37
max_exterior_facet_subdomain_id() (ffc.uflacs.backends.ufc.form.ufc_form method), 37
max_facet_edge_length() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt method), 9
max_facet_edge_length() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 11
max_facet_edge_length() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 14
max_interface_subdomain_id() (ffc.uflacs.backends.ufc.form.ufc_form method), 37
max_interior_facet_subdomain_id() (ffc.uflacs.backends.ufc.form.ufc_form method), 37
max_overlap_subdomain_id() (ffc.uflacs.backends.ufc.form.ufc_form method), 37
max_value() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt method), 9
max_value() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 11
max_value() (ffc.uflacs.analysis.graph_rebuild.ReconstructScalarSubexpressions method), 22
max_value() (ffc.uflacs.language.ufl_to_cnodes.RulesForC method), 55
max_value() (ffc.uflacs.language.ufl_to_cnodes.RulesForCpp method), 55
max_vertex_subdomain_id() (ffc.uflacs.backends.ufc.form.ufc_form method), 37
members() (ffc.uflacs.backends.ufc.generator.ufc_generator method), 38
MemCopy() (in module ffc.uflacs.language.cnodes), 46
MemZero() (in module ffc.uflacs.language.cnodes), 46
MemZeroRange() (in module ffc.uflacs.language.cnodes), 46
message (ffc.uflacs.language.cnodes.Throw attribute), 51
min_facet_edge_length() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt method), 9
min_facet_edge_length() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 11
min_facet_edge_length() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 14
min_value() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt method), 9
min_value() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 12
min_value() (ffc.uflacs.analysis.graph_rebuild.ReconstructScalarSubexpress method), 22
min_value() (ffc.uflacs.language.ufl_to_cnodes.RulesForC method), 55
min_value() (ffc.uflacs.language.ufl_to_cnodes.RulesForCpp method), 55
Mod (class in ffc.uflacs.language.cnodes), 47
MOD (ffc.uflacs.language.precedence.PRECEDENCE attribute), 54
ModifiedTerminal (class in ffc.uflacs.analysis.modified_terminals), 24
Mul (class in ffc.uflacs.language.cnodes), 47
MUL (ffc.uflacs.language.precedence.PRECEDENCE attribute), 54
multi_index() (ffc.quadrature.quadraturetransformerbase.QuadratureTransfo method), 14
multi_index() (ffc.uflacs.analysis.graph_rebuild.ReconstructScalarSubexpre method), 22
multiply_block() (in module ffc.uflacs.build_uflacs_ir), 58
multiply_block_interior_facets() (in module ffc.uflacs.build_uflacs_ir), 58

N

name (ffc.uflacs.build_uflacs_ir.preintegrated_block_data_t attribute), 59
name (ffc.uflacs.build_uflacs_ir.premultiplied_block_data_t attribute), 59
name (ffc.uflacs.language.cnodes.Namespace attribute), 47
name (ffc.uflacs.language.cnodes.Symbol attribute), 50
name (ffc.uflacs.language.cnodes.Using attribute), 51
Namespace (class in ffc.uflacs.language.cnodes), 47
NaryOp (class in ffc.uflacs.language.cnodes), 47
NE (class in ffc.uflacs.language.cnodes), 47
NE (ffc.uflacs.language.precedence.PRECEDENCE attribute), 54
ne() (ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin method), 56
needs_mesh_entities() (ffc.uflacs.backends.ufc.dofmap.ufc_dofmap method), 32
needs_oriented_jacobian() (in module ffc.representation), 70
needs_oriented_jacobian() (in module ffc.representationutils), 70
Neg (class in ffc.uflacs.language.cnodes), 47
NEG (ffc.uflacs.language.precedence.PRECEDENCE attribute), 54
negative_restricted() (ffc.quadrature.quadraturetransformerbase.QuadratureT method), 14
negative_restricted() (ffc.uflacs.analysis.balancing.BalanceModifiers method), 19
New (class in ffc.uflacs.language.cnodes), 47

new_symbol() (ffc.uflacs.analysis.valuenumbering.ValueNumberer method), 26
new_symbols() (ffc.uflacs.analysis.valuenumbering.ValueNumberer method), 26
new_temp_symbol() (ffc.uflacs.integralgenerator.IntegralGenerator method), 63
NoOp() (in module ffc.uflacs.language.cnodes), 47
Not (class in ffc.uflacs.language.cnodes), 47
NOT (ffc.uflacs.language.precedence.PRECEDENCE attribute), 54
not_condition() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt method), 9
not_condition() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 12
not_condition() (ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin method), 56
Null (class in ffc.uflacs.language.cnodes), 47
num (ffc.quadrature.fraction.Fraction attribute), 8
num_cells() (ffc.uflacs.backends.ufc.integrals.ufc_custom_integral method), 38
num_coefficients() (ffc.uflacs.backends.ufc.form.ufc_form method), 37
num_coordinate_component_dofs() (in module ffc.uflacs.backends.ffc.common), 28
num_custom_quadrature_points() (ffc.uflacs.backends.ffc.symbols.FFCBackendSymbols method), 31
num_element_dofs() (ffc.uflacs.backends.ufc.dofmap.ufc_dofmap method), 32
num_element_support_dofs() (ffc.uflacs.backends.ufc.dofmap.ufc_dofmap method), 32
num_elements (ffc.uflacs.analysis.crsarray.CRSArray attribute), 19
num_entity_closure_dofs() (ffc.uflacs.backends.ufc.dofmap.ufc_dofmap method), 32
num_entity_dofs() (ffc.uflacs.backends.ufc.dofmap.ufc_dofmap method), 32
num_facet_dofs() (ffc.uflacs.backends.ufc.dofmap.ufc_dofmap method), 32
num_global_support_dofs() (ffc.uflacs.backends.ufc.dofmap.ufc_dofmap method), 32
num_sub_dofmaps() (ffc.uflacs.backends.ufc.dofmap.ufc_dofmap method), 32
num_sub_elements() (ffc.uflacs.backends.ufc.finite_element.ufc_finite_element method), 35

O

offset (ffc.uflacs.language.cnodes.FlattenedArray attribute), 45
op (ffc.uflacs.language.cnodes.Add attribute), 39
op (ffc.uflacs.language.cnodes.AddressOf attribute), 39
op (ffc.uflacs.language.cnodes.And attribute), 40
op (ffc.uflacs.language.cnodes.Assign attribute), 40
op (ffc.uflacs.language.cnodes.AssignAdd attribute), 40
op (ffc.uflacs.language.cnodes.AssignAnd attribute), 40
op (ffc.uflacs.language.cnodes.AssignBitAnd attribute), 40
op (ffc.uflacs.language.cnodes.AssignBitOr attribute), 41
op (ffc.uflacs.language.cnodes.AssignBitXor attribute), 41
op (ffc.uflacs.language.cnodes.AssignDiv attribute), 41
op (ffc.uflacs.language.cnodes.AssignLShift attribute), 41
op (ffc.uflacs.language.cnodes.AssignMod attribute), 41
op (ffc.uflacs.language.cnodes.AssignMul attribute), 41
op (ffc.uflacs.language.cnodes.AssignOr attribute), 41
op (ffc.uflacs.language.cnodes.AssignRShift attribute), 41
op (ffc.uflacs.language.cnodes.AssignSub attribute), 41
op (ffc.uflacs.language.cnodes.BitAnd attribute), 41
op (ffc.uflacs.language.cnodes.BitNot attribute), 42
op (ffc.uflacs.language.cnodes.BitOr attribute), 42
op (ffc.uflacs.language.cnodes.BitXor attribute), 42
op (ffc.uflacs.language.cnodes.Dereference attribute), 44
op (ffc.uflacs.language.cnodes.Div attribute), 44
op (ffc.uflacs.language.cnodes.EQ attribute), 44
op (ffc.uflacs.language.cnodes.GE attribute), 45
op (ffc.uflacs.language.cnodes.GT attribute), 45
op (ffc.uflacs.language.cnodes.LE attribute), 46
op (ffc.uflacs.language.cnodes.LT attribute), 46
op (ffc.uflacs.language.cnodes.Mod attribute), 47
op (ffc.uflacs.language.cnodes.Mul attribute), 47
op (ffc.uflacs.language.cnodes.NE attribute), 47
op (ffc.uflacs.language.cnodes.Neg attribute), 47
op (ffc.uflacs.language.cnodes.Not attribute), 47
op (ffc.uflacs.language.cnodes.Or attribute), 48
op (ffc.uflacs.language.cnodes.Pos attribute), 48
op (ffc.uflacs.language.cnodes.PostDecrement attribute), 48
op (ffc.uflacs.language.cnodes.PostIncrement attribute), 48
op (ffc.uflacs.language.cnodes.PreDecrement attribute), 48
op (ffc.uflacs.language.cnodes.PreIncrement attribute), 49
op (ffc.uflacs.language.cnodes.Product attribute), 49
op (ffc.uflacs.language.cnodes.SizeOf attribute), 49
op (ffc.uflacs.language.cnodes.Sub attribute), 50
op (ffc.uflacs.language.cnodes.Sum attribute), 50
operation_count() (in module ffc.quadrature.reduce_operations), 16
ops() (ffc.quadrature.expr.Expr method), 7
ops() (ffc.quadrature.fraction.Fraction method), 8
ops() (ffc.quadrature.product.Product method), 10
ops() (ffc.quadrature.sumobj.Sum method), 17
ops() (ffc.quadrature.symbol.Symbol method), 17

optimise_code() (in module ffc.quadrature.symbolics), 18
optimize_element_tables() (in module ffc.uflacs.elementtables), 61
optimize_integral_ir() (in module ffc.quadrature.quadratureoptimization), 10
optimize_integral_ir() (in module ffc.tsfc.tsfcoptimization), 18
optimize_integral_ir() (in module ffc.uflacs.uflacsoptimization), 63
optimize_ir() (in module ffc.optimization), 69
Or (class in ffc.uflacs.language.cnodes), 48
OR (ffc.uflacs.language.precedence.PRECEDENCE attribute), 54
or_condition() (ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin method), 56
orientation() (in module ffc.uflacs.backends.ufc.jacobian), 39
original_coefficient_position() (ffc.uflacs.backends.ufc.form.ufc_form method), 37

P

pad_dim() (in module ffc.uflacs.language.cnodes), 53
pad_innermost_dim() (in module ffc.uflacs.language.cnodes), 53
padlen (ffc.uflacs.language.cnodes.ArrayDecl attribute), 40
parse_optimise_parameters() (in module ffc.quadrature.parameters), 9
parse_uflacs_optimization_parameters() (in module ffc.uflacs.build_uflacs_ir), 58
partial_block_data_t (class in ffc.uflacs.build_uflacs_ir), 58
pdet_m1() (in module ffc.uflacs.backends.ufc.coordinate_mapping), 31
pick_first() (in module ffc.utils), 70
pick_representation() (in module ffc.representation), 70
piecewise_ma_index (ffc.uflacs.build_uflacs_ir.partial_block_data_t attribute), 58
plot() (in module ffc.plot), 69
points_table() (ffc.uflacs.backends.ffc.symbols.FFCBackendSymbols method), 31
pop_level() (in module ffc.log), 68
Pos (class in ffc.uflacs.language.cnodes), 48
POS (ffc.uflacs.language.precedence.PRECEDENCE attribute), 54
positive_restricted() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 14
positive_restricted() (ffc.uflacs.analysis.balancing.BalanceModifiers method), 19
POST_DEC (ffc.uflacs.language.precedence.PRECEDENCE attribute), 54
POST_INC (ffc.uflacs.language.precedence.PRECEDENCE attribute), 54
PostDecrement (class in ffc.uflacs.language.cnodes), 48
PostfixUnaryOp (class in ffc.uflacs.language.cnodes), 48
PostIncrement (class in ffc.uflacs.language.cnodes), 48
power() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransfo method), 9
power() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 12
power() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerB method), 14
power() (ffc.uflacs.analysis.graph_rebuild.ReconstructScalarSubexpressions method), 22
power() (ffc.uflacs.language.ufl_to_cnodes.RulesForC method), 55
power() (ffc.uflacs.language.ufl_to_cnodes.RulesForCpp method), 55
Pragma (class in ffc.uflacs.language.cnodes), 48
pragma (ffc.uflacs.language.cnodes.For attribute), 45
pragma (ffc.uflacs.language.cnodes.ForRange attribute), 45
PRE_DEC (ffc.uflacs.language.precedence.PRECEDENCE attribute), 54
PRE_INC (ffc.uflacs.language.precedence.PRECEDENCE attribute), 54
PRECEDENCE (class in ffc.uflacs.language.precedence), 54
precedence (ffc.uflacs.language.cnodes.Add attribute), 39
precedence (ffc.uflacs.language.cnodes.AddressOf attribute), 40
precedence (ffc.uflacs.language.cnodes.And attribute), 40
precedence (ffc.uflacs.language.cnodes.ArrayAccess attribute), 40
precedence (ffc.uflacs.language.cnodes.AssignOp attribute), 41
precedence (ffc.uflacs.language.cnodes.BitAnd attribute), 41
precedence (ffc.uflacs.language.cnodes.BitNot attribute), 42
precedence (ffc.uflacs.language.cnodes.BitOr attribute), 42
precedence (ffc.uflacs.language.cnodes.BitXor attribute), 42
precedence (ffc.uflacs.language.cnodes.Call attribute), 43
precedence (ffc.uflacs.language.cnodes.CExprLiteral attribute), 42
precedence (ffc.uflacs.language.cnodes.Conditional attribute), 43
precedence (ffc.uflacs.language.cnodes.Dereference attribute), 44
precedence (ffc.uflacs.language.cnodes.Div attribute), 44
precedence (ffc.uflacs.language.cnodes.EQ attribute), 44
precedence (ffc.uflacs.language.cnodes.GE attribute), 45
precedence (ffc.uflacs.language.cnodes.GT attribute), 45

precedence (ffc.uflacs.language.cnodes.LE attribute), 46
precedence (ffc.uflacs.language.cnodes.LiteralBool attribute), 46
precedence (ffc.uflacs.language.cnodes.LiteralFloat attribute), 46
precedence (ffc.uflacs.language.cnodes.LiteralInt attribute), 46
precedence (ffc.uflacs.language.cnodes.LiteralString attribute), 46
precedence (ffc.uflacs.language.cnodes.LT attribute), 46
precedence (ffc.uflacs.language.cnodes.Mod attribute), 47
precedence (ffc.uflacs.language.cnodes.Mul attribute), 47
precedence (ffc.uflacs.language.cnodes.NE attribute), 47
precedence (ffc.uflacs.language.cnodes.Neg attribute), 47
precedence (ffc.uflacs.language.cnodes.Not attribute), 47
precedence (ffc.uflacs.language.cnodes.Null attribute), 48
precedence (ffc.uflacs.language.cnodes.Or attribute), 48
precedence (ffc.uflacs.language.cnodes.Pos attribute), 48
precedence (ffc.uflacs.language.cnodes.PostDecrement attribute), 48
precedence (ffc.uflacs.language.cnodes.PostIncrement attribute), 48
precedence (ffc.uflacs.language.cnodes.PreDecrement attribute), 48
precedence (ffc.uflacs.language.cnodes.PreIncrement attribute), 49
precedence (ffc.uflacs.language.cnodes.Product attribute), 49
precedence (ffc.uflacs.language.cnodes.SizeOf attribute), 49
precedence (ffc.uflacs.language.cnodes.Sub attribute), 50
precedence (ffc.uflacs.language.cnodes.Sum attribute), 50
precedence (ffc.uflacs.language.cnodes.Symbol attribute), 50
precedence (ffc.uflacs.language.cnodes.VerbatimExpr attribute), 51
PreDecrement (class in ffc.uflacs.language.cnodes), 48
PrefixUnaryOp (class in ffc.uflacs.language.cnodes), 49
PreIncrement (class in ffc.uflacs.language.cnodes), 48
preintegrated_block_data_t (class in ffc.uflacs.build_uflacs_ir), 59
premultiplied_block_data_t (class in ffc.uflacs.build_uflacs_ir), 59
primal_forms() (ffc.errorcontrol.errorcontrolgenerators.ErrorControlGenerator method), 5
print_error() (in module ffc.main), 68
Product (class in ffc.quadrature.product), 9
Product (class in ffc.uflacs.language.cnodes), 49
product() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt method), 9
product() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 12
product() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 14
product() (ffc.uflacs.analysis.graph_rebuild.ReconstructScalarSubexpressio method), 22
product() (ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin method), 56
push_level() (in module ffc.log), 68
push_row() (ffc.uflacs.analysis.crsarray.CRSArray method), 19

Q

quadrature_loop_index() (ffc.uflacs.backends.ffc.symbols.FFCBackendSym method), 31
quadrature_weight() (ffc.quadrature.optimisedquadraturetransformer.Quadra method), 9
quadrature_weight() (ffc.quadrature.quadraturetransformer.QuadratureTrans method), 12
quadrature_weight() (ffc.quadrature.quadraturetransformerbase.QuadratureT method), 14
QuadratureRepresentationDeprecationWarning, 7
QuadratureTransformer (class in ffc.quadrature.quadraturetransformer), 11
QuadratureTransformerBase (class in ffc.quadrature.quadraturetransformerbase), 12
QuadratureTransformerOpt (class in ffc.quadrature.optimisedquadraturetransformer), 8

R

rank() (ffc.uflacs.backends.ufc.form.ufc_form method), 37
rebuild_expression_from_graph() (in module ffc.uflacs.analysis.graph_rebuild), 22
rebuild_scalar_graph_from_factorization() (in module ffc.uflacs.analysis.factorization), 21
rebuild_with_scalar_subexpressions() (in module ffc.uflacs.analysis.graph_rebuild), 22
ReconstructScalarSubexpressions (class in ffc.uflacs.analysis.graph_rebuild), 21
reduce_operations() (in module ffc.quadrature.reduce_operations), 16
reduce_ops() (ffc.quadrature.expr.Expr method), 7
reduce_ops() (ffc.quadrature.fraction.Fraction method), 8
reduce_ops() (ffc.quadrature.product.Product method), 10
reduce_ops() (ffc.quadrature.sumobj.Sum method), 17
reduce_var() (ffc.quadrature.expr.Expr method), 7
reduce_var() (ffc.quadrature.fraction.Fraction method), 8
reduce_vartype() (ffc.quadrature.expr.Expr method), 7
reduce_vartype() (ffc.quadrature.fraction.Fraction method), 8
reduce_vartype() (ffc.quadrature.product.Product method), 10

reduce_vartype() (ffc.quadrature.sumobj.Sum method), 17
reduction_possible() (in module ffc.quadrature.reduce_operations), 16
reference_cell() (in module ffc.fiatinterface), 66
reference_cell_edge_vectors() (ffc.uflacs.backends.ffc.access.FFCBackendAccess method), 27
reference_cell_edge_vectors() (ffc.uflacs.backends.ffc.definitions.FFCBackendDefinitions method), 29
reference_cell_vertices() (in module ffc.fiatinterface), 66
reference_cell_volume() (ffc.uflacs.backends.ffc.access.FFCBackendAccess method), 27
reference_cell_volume() (ffc.uflacs.backends.ffc.definitions.FFCBackendDefinitions method), 29
reference_facet_edge_vectors() (ffc.uflacs.backends.ffc.access.FFCBackendAccess method), 27
reference_facet_edge_vectors() (ffc.uflacs.backends.ffc.definitions.FFCBackendDefinitions method), 29
reference_facet_volume() (ffc.uflacs.backends.ffc.access.FFCBackendAccess method), 27
reference_facet_volume() (ffc.uflacs.backends.ffc.definitions.FFCBackendDefinitions method), 29
reference_grad() (ffc.uflacs.analysis.balancing.BalanceModifiers method), 19
reference_grad() (ffc.uflacs.analysis.valuenumbering.ValueNumberer method), 26
reference_normal() (ffc.uflacs.backends.ffc.access.FFCBackendAccess method), 27
reference_normal() (ffc.uflacs.backends.ffc.definitions.FFCBackendDefinitions method), 29
reference_to_physical_map() (in module ffc.uflacs.backends.ufc.evaluatedof), 34
reference_value() (ffc.uflacs.analysis.balancing.BalanceModifiers method), 19
reference_value() (ffc.uflacs.analysis.valuenumbering.ValueNumberer method), 26
reference_value_dimension() (ffc.uflacs.backends.ufc.finite_element.ufc_finite_element method), 35
reference_value_rank() (ffc.uflacs.backends.ufc.finite_element.ufc_finite_element method), 35
reference_value_size() (ffc.uflacs.backends.ufc.finite_element.ufc_finite_element method), 35
remove_unused() (in module ffc.quadrature.cpp), 6
restricted() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 14
restricted() (ffc.uflacs.analysis.valuenumbering.ValueNumberer method), 26
restrictions (ffc.uflacs.build_uflacs_ir.common_block_data_t attribute), 57
restrictions (ffc.uflacs.build_uflacs_ir.full_block_data_t attribute), 57
restrictions (ffc.uflacs.build_uflacs_ir.partial_block_data_t attribute), 58
restrictions (ffc.uflacs.build_uflacs_ir.preintegrated_block_data_t attribute), 59
restrictions (ffc.uflacs.build_uflacs_ir.premultiplied_block_data_t attribute), 60
Return (class in ffc.uflacs.language.cnodes), 49
rhs (ffc.uflacs.language.cnodes.BinOp attribute), 41
RuleAccess (class in ffc.uflacs.language.ufl_to_cnodes), 55
RuleLoop (class in ffc.uflacs.language.ufl_to_cnodes), 55

S

scalar_nary() (ffc.uflacs.analysis.graph_rebuild.ReconstructScalarSubexpres method), 22
scalar_value() (ffc.quadrature.quadraturetransformerbase.QuadratureTransfo method), 14
scale_loop() (in module ffc.uflacs.language.cnodes), 53
Scope (class in ffc.uflacs.language.cnodes), 49
set_exception_handling() (in module ffc.quadrature.cpp), 6
set_float_formatting() (in module ffc.quadrature.cpp), 6
set_handler() (in module ffc.log), 68
set_indent() (in module ffc.log), 68
set_level() (in module ffc.log), 68
set_prefix() (in module ffc.log), 68
set_var() (ffc.uflacs.integralgenerator.IntegralGenerator method), 63
sideeffect (ffc.uflacs.language.cnodes.AssignOp attribute), 41
sideeffect (ffc.uflacs.language.cnodes.Call attribute), 43
sideeffect (ffc.uflacs.language.cnodes.CExprOperator attribute), 42
sideeffect (ffc.uflacs.language.cnodes.CExprTerminal attribute), 42
sideeffect (ffc.uflacs.language.cnodes.PostDecrement attribute), 48
sideeffect (ffc.uflacs.language.cnodes.PostIncrement attribute), 48
sideeffect (ffc.uflacs.language.cnodes.PreDecrement attribute), 48
sideeffect (ffc.uflacs.language.cnodes.PreIncrement attribute), 48
signature() (ffc.uflacs.backends.ufc.generator.ufc_generator method), 38
sin() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 14
sin() (ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin method), 56

sinh() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 14
sinh() (ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin method), 56
SizeOf (class in ffc.uflacs.language.cnodes), 49
SIZEOF (ffc.uflacs.language.precedence.PRECEDENCE attribute), 54
sizes (ffc.uflacs.language.cnodes.ArrayDecl attribute), 40
sort_integrals() (in module ffc.quadrature.quadraturerepresentation), 10
space_dimension() (ffc.uflacs.backends.ufc.finite_element.ufc_finite_element method), 35
SpaceOfReals (class in ffc.fiatinterface), 66
spatial_coordinate() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 14
spatial_coordinate() (ffc.uflacs.backends.ffc.access.FFCBackendAccess method), 27
spatial_coordinate() (ffc.uflacs.backends.ffc.definitions.FFCBackendDefinitions method), 29
split_expression() (in module ffc.quadrature.reduce_operations), 16
split_parameters() (in module ffc.parameters), 69
sqrt() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 14
sqrt() (ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin method), 56
Sqrt() (in module ffc.uflacs.language.cnodes), 49
Statement (class in ffc.uflacs.language.cnodes), 49
StatementList (class in ffc.uflacs.language.cnodes), 49
statements (ffc.uflacs.language.cnodes.StatementList attribute), 50
strides (ffc.uflacs.language.cnodes.FlattenedArray attribute), 45
strip_modified_terminal() (in module ffc.uflacs.analysis.modified_terminals), 25
strip_table_zeros() (in module ffc.uflacs.elementtables), 61
Sub (class in ffc.uflacs.language.cnodes), 50
SUB (ffc.uflacs.language.precedence.PRECEDENCE attribute), 54
SUBSCRIPT (ffc.uflacs.language.precedence.PRECEDENCE attribute), 55
sufficient_int() (in module ffc.uflacs.analysis.crsarray), 19
Sum (class in ffc.quadrature.sumobj), 16
Sum (class in ffc.uflacs.language.cnodes), 50
sum() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt method), 9
sum() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 12
sum() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 14
sum() (ffc.uflacs.analysis.graph_rebuild.ReconstructScalarSubexpressions method), 22
sum() (ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin method), 56
Switch (class in ffc.uflacs.language.cnodes), 50
Symbol (class in ffc.quadrature.symbol), 17
Symbol (class in ffc.uflacs.language.cnodes), 50
symbol (ffc.uflacs.language.cnodes.ArrayDecl attribute), 40
symbol (ffc.uflacs.language.cnodes.VariableDecl attribute), 51
SYMBOL (ffc.uflacs.language.precedence.PRECEDENCE attribute), 55

T

t (ffc.quadrature.expr.Expr attribute), 7
table_origin_t (in module ffc.uflacs.elementtables), 61
tabledata (ffc.uflacs.build_uflacs_ir.ma_data_t attribute), 58
tabulate_basis() (in module ffc.quadrature.tabulate_basis), 18
tabulate_coefficients() (in module ffc.uflacs.backends.ufc.evaluatebasis), 33
tabulate_dof_coordinates() (ffc.uflacs.backends.ufc.finite_element.ufc_finite_element method), 35
tabulate_dofs() (ffc.uflacs.backends.ufc.dofmap.ufc_dofmap method), 32
tabulate_entity_closure_dofs() (ffc.uflacs.backends.ufc.dofmap.ufc_dofmap method), 32
tabulate_entity_dofs() (ffc.uflacs.backends.ufc.dofmap.ufc_dofmap method), 32
tabulate_facet_dofs() (ffc.uflacs.backends.ufc.dofmap.ufc_dofmap method), 32
tabulate_reference_dof_coordinates() (ffc.uflacs.backends.ufc.finite_element.ufc_finite_element method), 35
tabulate_tensor() (ffc.uflacs.backends.ufc.integrals.ufc_integral method), 38
tabulate_tensor_comment() (ffc.uflacs.backends.ufc.integrals.ufc_integral method), 38
tan() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 14
tan() (ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin method), 56
tanh() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 14
tanh() (ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin method), 56
terminal() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 14
terminal() (ffc.uflacs.analysis.balancing.BalanceModifiers method), 19

terminal() (ffc.uflacs.analysis.graph_rebuild.ReconstructScalarSubexpressions method), 22
Throw (class in ffc.uflacs.language.cnodes), 50
topological_dimension() (ffc.uflacs.backends.ufc.coordinate_mapping.ufc_coordinate_mapping method), 32
topological_dimension() (ffc.uflacs.backends.ufc.dofmap.ufc_dofmap method), 32
topological_dimension() (ffc.uflacs.backends.ufc.finite_element.ufc_finite_element method), 35
total_shape() (in module ffc.uflacs.analysis.expr_shapes), 20
transform_component() (in module ffc.representationutils), 70
transform_reference_basis_derivatives() (ffc.uflacs.backends.ufc.finite_element.ufc_finite_element method), 35
transposed (ffc.uflacs.build_uflacs_ir.common_block_data_t attribute), 57
transposed (ffc.uflacs.build_uflacs_ir.full_block_data_t attribute), 57
transposed (ffc.uflacs.build_uflacs_ir.partial_block_data_t attribute), 58
transposed (ffc.uflacs.build_uflacs_ir.preintegrated_block_data_t attribute), 59
transposed (ffc.uflacs.build_uflacs_ir.premultiplied_block_data_t attribute), 60
transposed() (ffc.uflacs.analysis.graph_rebuild.ReconstructScalarSubexpressions method), 22
true (ffc.uflacs.language.cnodes.Conditional attribute), 43
ttypes (ffc.uflacs.build_uflacs_ir.common_block_data_t attribute), 57
ttypes (ffc.uflacs.build_uflacs_ir.full_block_data_t attribute), 58
ttypes (ffc.uflacs.build_uflacs_ir.partial_block_data_t attribute), 59
ttypes (ffc.uflacs.build_uflacs_ir.preintegrated_block_data_t attribute), 59
ttypes (ffc.uflacs.build_uflacs_ir.premultiplied_block_data_t attribute), 60
typename (ffc.uflacs.language.cnodes.ArrayDecl attribute), 40
typename (ffc.uflacs.language.cnodes.New attribute), 47
typename (ffc.uflacs.language.cnodes.VariableDecl attribute), 51

U

ufc_cell_integral (class in ffc.uflacs.backends.ufc.integrals), 38
ufc_coordinate_mapping (class in ffc.uflacs.backends.ufc.coordinate_mapping), 32
ufc_custom_integral (class in ffc.uflacs.backends.ufc.integrals), 38
ufc_dofmap (class in ffc.uflacs.backends.ufc.dofmap), 32
ufc_exterior_facet_integral (class in ffc.uflacs.backends.ufc.integrals), 38
ufc_finite_element (class in ffc.uflacs.backends.ufc.finite_element), 34
ufc_form (class in ffc.uflacs.backends.ufc.form), 35
ufc_generator (class in ffc.uflacs.backends.ufc.generator), 37
ufc_integral (class in ffc.uflacs.backends.ufc.integrals), 38
ufc_interface_integral (class in ffc.uflacs.backends.ufc.integrals), 38
ufc_interior_facet_integral (class in ffc.uflacs.backends.ufc.integrals), 38
ufc_overlap_integral (class in ffc.uflacs.backends.ufc.integrals), 38
ufc_restriction_offset() (in module ffc.uflacs.backends.ffc.common), 28
ufc_restriction_postfix() (in module ffc.uflacs.backends.ffc.symbols), 31
ufc_vertex_integral (class in ffc.uflacs.backends.ufc.integrals), 38
UFL2CNodesMixin (class in ffc.uflacs.language.ufl_to_cnodes), 55
UFL2CNodesTranslatorC (class in ffc.uflacs.language.ufl_to_cnodes), 56
UFL2CNodesTranslatorCpp (class in ffc.uflacs.language.ufl_to_cnodes), 56
uflacs_default_parameters() (in module ffc.uflacs.build_uflacs_ir), 60
UFLErrorControlGenerator (class in ffc.errorcontrol.errorcontrolgenerators), 5
unames (ffc.uflacs.build_uflacs_ir.common_block_data_t attribute), 57
unames (ffc.uflacs.build_uflacs_ir.full_block_data_t attribute), 58
unames (ffc.uflacs.build_uflacs_ir.partial_block_data_t attribute), 59
unames (ffc.uflacs.build_uflacs_ir.preintegrated_block_data_t attribute), 59
unames (ffc.uflacs.build_uflacs_ir.premultiplied_block_data_t attribute), 60
UnaryOp (class in ffc.uflacs.language.cnodes), 51
unexpected() (ffc.uflacs.analysis.graph_rebuild.ReconstructScalarSubexpressions method), 22
unique_psi_tables() (in module ffc.quadrature.quadratureutils), 15
unique_table_reference_t (in module ffc.uflacs.elementtables), 61
unique_tables() (in module ffc.quadrature.quadratureutils), 15
update (ffc.uflacs.language.cnodes.For attribute), 45

update_cell() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 14
update_facets() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 14
update_points() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 14
update_vertex() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 14
uses_integral_moments() (in module ffc.representation), 70
Using (class in ffc.uflacs.language.cnodes), 51
utility_type() (ffc.uflacs.analysis.graph_rebuild.ReconstructScalarSubexpressions method), 22

V

v (ffc.quadrature.symbol.Symbol attribute), 17
val (ffc.quadrature.expr.Expr attribute), 7
validate_jit_parameters() (in module ffc.parameters), 69
validate_parameters() (in module ffc.parameters), 69
value (ffc.uflacs.language.cnodes.Case attribute), 43
value (ffc.uflacs.language.cnodes.LiteralBool attribute), 46
value (ffc.uflacs.language.cnodes.LiteralFloat attribute), 46
value (ffc.uflacs.language.cnodes.LiteralInt attribute), 46
value (ffc.uflacs.language.cnodes.LiteralString attribute), 46
value (ffc.uflacs.language.cnodes.Return attribute), 49
value (ffc.uflacs.language.cnodes.VariableDecl attribute), 51
value_dimension() (ffc.uflacs.backends.ufc.finite_element.ufc_finite_element method), 35
value_rank() (ffc.uflacs.backends.ufc.finite_element.ufc_finite_element method), 35
value_size() (ffc.uflacs.backends.ufc.finite_element.ufc_finite_element method), 35
ValueNumberer (class in ffc.uflacs.analysis.valuenumbering), 25
values (ffc.uflacs.language.cnodes.ArrayDecl attribute), 40
variable() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 15
variable() (ffc.uflacs.analysis.graph_rebuild.ReconstructScalarSubexpressions method), 22
variable() (ffc.uflacs.analysis.valuenumbering.ValueNumberer method), 26
VariableDecl (class in ffc.uflacs.language.cnodes), 51
VerbatimExpr (class in ffc.uflacs.language.cnodes), 51
VerbatimStatement (class in ffc.uflacs.language.cnodes), 51
vrs (ffc.quadrature.product.Product attribute), 10
vrs (ffc.quadrature.sumobj.Sum attribute), 17

W

warning() (in module ffc.log), 68
warning_blue() (in module ffc.log), 68
warning_green() (in module ffc.log), 68
warning_red() (in module ffc.log), 68
weights_table() (ffc.uflacs.backends.ffc.symbols.FFCBackendSymbols method), 31
While (class in ffc.uflacs.language.cnodes), 51
write_code() (in module ffc.formatting), 67

X

X_component() (ffc.uflacs.backends.ffc.symbols.FFCBackendSymbols method), 30
x_component() (ffc.uflacs.backends.ffc.symbols.FFCBackendSymbols method), 31

Z

zero() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 15
zero() (ffc.uflacs.backends.ffc.access.FFCBackendAccess method), 28
zero() (ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin method), 56