

---

# **FEniCS Form Compiler (FFC) Documentation**

*Release 2017.1.0*

**FEniCS Project**

**May 12, 2017**



---

## Contents

---

<b>1 Documentation</b>	<b>3</b>
<b>Python Module Index</b>	<b>81</b>



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>



## 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.

## Requirements and dependencies

FFC requires Python version 2.7 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](#)<sup>1</sup>
- [COFFEE](#)<sup>1</sup>
- [FIAT](#)<sup>1</sup>

---

<sup>1</sup> 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<sup>2</sup>
- networkx<sup>2</sup>
- PuLP<sup>2, 4</sup>
- GLPK<sup>3, 4</sup>

---

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

```
docker pull quay.io/fenicsproject/stable:2017.1.0
docker run -ti --rm quay.io/fenicsproject/stable:2017.1.0
sudo apt-get update && sudo apt-get -y install glpk-utils && \
  sudo pip2 install --prefix=${FENICS_PREFIX} --no-cache-dir \
  git+https://github.com/blechta/tsfc.git@2017.1.0 \
  git+https://github.com/blechta/COFFEE.git@2017.1.0 \
  git+https://github.com/blechta/FInAT.git@2017.1.0 \
  singledispatch networkx pulp && \
  sudo pip3 install --prefix=${FENICS_PREFIX} --no-cache-dir \
  git+https://github.com/blechta/tsfc.git@2017.1.0 \
  git+https://github.com/blechta/COFFEE.git@2017.1.0 \
  git+https://github.com/blechta/FInAT.git@2017.1.0 \
  singledispatch networkx pulp && \
  sudo apt-get clean && \
  sudo rm -rf /var/lib/apt/lists/* /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](#).

---

## Installation instructions

To install FFC, download the source code from the [FFC Bitbucket repository](#), and run the following command:

```
pip install .
```

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

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

## User manual

---

**Note:** This page is work in progress.

---

---

<sup>2</sup> Pip-installable.

<sup>4</sup> Needed for certain COFFEE optimizations.

<sup>3</sup> Binary package; `glpsol` executable needed. Version GLPSOL: GLPK LP/MIP Solver, v4.57 from Ubuntu 16.04 `glpk-utils` package is known to produce the same references as our test system.



## ffc package

### Subpackages

#### ffc.backends package

#### Subpackages

#### ffc.backends.dolfin package

#### Submodules

#### ffc.backends.dolfin.capsules module

```
class ffc.backends.dolfin.capsules.UFCElementNames (name,
                                                    ufc_finite_element_classnames,
                                                    ufc_dofmap_classnames)
```

Bases: `object`

Encapsulation of the names related to a generated UFC element.

```
class ffc.backends.dolfin.capsules.UFCFormNames (name, coefficient_names,
                                                  ufc_form_classname,
                                                  ufc_finite_element_classnames,
                                                  ufc_dofmap_classnames, superclass-
                                                  name='Form')
```

Bases: `object`

Encapsulation of the names related to a generated UFC form.

#### ffc.backends.dolfin.form module

```
ffc.backends.dolfin.form.generate_form (form, classname, error_control)
```

Generate dolfin wrapper code associated with a form including code for function spaces used in form and typedefs

**@param form:** A UFCFormNames instance

**@param classname** Name of Form class.

#### ffc.backends.dolfin.functionspace module

```
ffc.backends.dolfin.functionspace.apply_function_space_template (name, element_name,
                                                                  dofmap_name)
```

```
ffc.backends.dolfin.functionspace.apply_multimesh_function_space_template (name,
                                                                              single_name,
                                                                              element_name,
                                                                              dofmap_name)
```

`ffc.backends.dolfin.functionspace.extract_coefficient_spaces` (*forms*)

Extract a list of tuples

(*classname*, *finite\_element\_classname*, *dofmap\_classname*)

for the coefficient spaces in the set of given forms. This can then be used for input to the function space template.

`ffc.backends.dolfin.functionspace.generate_typedefs` (*form*, *classname*, *error\_control*)

Generate typedefs for test, trial and coefficient spaces relative to a function space.

### **ffc.backends.dolfin.goalfunctional module**

`ffc.backends.dolfin.goalfunctional.generate_update_ec` (*form*)

### **ffc.backends.dolfin.includes module**

### **ffc.backends.dolfin.wrappers module**

`ffc.backends.dolfin.wrappers.generate_dolfin_code` (*prefix*, *header*, *forms*, *common\_function\_space=False*,  
*add\_guards=False*, *error\_control=False*)

Generate complete dolfin wrapper code with given generated names.

@**param prefix**: String, prefix for all form names.

@**param header**: Code that will be inserted at the top of the file.

@**param forms**: List of UFCFormNames instances or single UFCElementNames.

@**param common\_function\_space**: True if common function space, otherwise False

@**param add\_guards**: True iff guards (ifdefs) should be added

@**param error\_control**: True iff adaptivity typedefs (ifdefs) should be added

## **Module contents**

### **ffc.backends.ufc package**

#### **Submodules**

#### **ffc.backends.ufc.coordinate\_mapping module**

#### **ffc.backends.ufc.dofmap module**

#### **ffc.backends.ufc.finite\_element module**

#### **ffc.backends.ufc.form module**

#### **ffc.backends.ufc.function module**

## ffc.backends.ufc.integrals module

### Module contents

Code generation format strings for UFC (Unified Form-assembly Code) version 2017.1.0

Five format strings are defined for each of the following UFC classes:

cell\_integral exterior\_facet\_integral interior\_facet\_integral custom\_integral cutcell\_integral inter-  
face\_integral overlap\_integral function

finite\_element dofmap coordinate\_mapping form

The strings are named:

'<classname>\_header'      '<classname>\_implementation'      '<classname>\_combined'      '<class-  
name>\_jit\_header' '<classname>\_jit\_implementation'

The header and implementation contain the definition and declaration of the class respectively, and are meant to be placed in .h and .cpp files, while the combined version is for an implementation within a single .h header. The \_jit\_ versions are used in the jit compiler and contains some additional factory functions exported as extern "C" to allow construction of compiled objects through ctypes without dealing with C++ ABI and name mangling issues.

Each string has at least the following format variables: 'classname', 'members', 'constructor', 'destructor', plus one for each interface function with name equal to the function name.

For more information about UFC and the FEniCS Project, visit

<http://www.fenicsproject.org> <https://bitbucket.org/fenics-project/ffc>

`ffc.backends.ufc.all_ufc_classnames()`

Build list of all classnames.

`ffc.backends.ufc.get_include_path()`

Return location of UFC header files

`ffc.backends.ufc.get_ufc_cxx_flags()`

Return C++ flags for compiling UFC C++11 code.

Return type is a list of strings.

Used internally in some tests.

`ffc.backends.ufc.get_ufc_signature()`

Return SHA-1 hash of the contents of ufc.h and ufc\_geometry.h.

In this implementation, the value is computed on import.

`ffc.backends.ufc.get_ufc_templates_signature()`

Return SHA-1 hash of the ufc code templates.

In this implementation, the value is computed on import.

`ffc.backends.ufc.ufc_signature()`

Return SHA-1 hash of the contents of ufc.h and ufc\_geometry.h.

In this implementation, the value is computed on import.

### Module contents

#### ffc.errorcontrol package

## Submodules

### 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.UFL_ErrorControlGenerator(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.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_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,      ele-
                                                             ment_numbers,
                                                             classnames, param-
                                                             eters)
```

Compute intermediate representation 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.QuadratureTransformerBase(psi_tables,
                                                                           quad_weights,
                                                                           gdim,
                                                                           tdim,
                                                                           en-
                                                                           tity_type,
                                                                           func-
                                                                           tion_replace_map,
                                                                           opti-
                                                                           mise_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*)  
**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)
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]` `ta-`

**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.,  $((x + y)) \rightarrow x + y$ ,  $2*(x + y) \rightarrow 2*x + 2*y$ ,  $(x + y)*(x + y) \rightarrow x*x + y*y + 2*x*y$ ,  $z*(x*(y + 3) + 2) + 1 \rightarrow 1 + 2*z + x*y*z + x*z*3$ ,  $z*((y + 3)*x + 2) + 1 \rightarrow 1 + 2*z + x*y*z + x*z*3$

`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_occurrences, [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 operations 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.tensor package

#### Submodules

#### ffc.tensor.costestimation module

```
ffc.tensor.costestimation.estimate_cost(integral, function_replace_map)
```

Estimate cost of tensor representation for integral. The cost is computed as the sum of the number of coefficients and derivatives, if the integrand can be represented as a monomial, and -1 if not.

#### ffc.tensor.geometrytensor module

```
class ffc.tensor.geometrytensor.GeometryTensor(monomial)
```

Bases: `object`

This class represents the geometry tensor for a monomial term of a multilinear form.



**ffc.tensor.monomialextraction module**

Extraction of monomial representations of UFL forms.

**class** `ffc.tensor.monomialextraction.Monomial` (*arg=None*)

Bases: `object`

This class represents a product of monomial factors.

**apply\_derivative** (*indices*)

**apply\_indices** (*indices*)

**apply\_restriction** (*restriction*)

**apply\_tensor** (*indices*)

**exception** `ffc.tensor.monomialextraction.MonomialException` (*\*args, \*\*kwargs*)

Bases: `Exception`

Exception raised when monomial extraction fails.

**class** `ffc.tensor.monomialextraction.MonomialFactor` (*arg=None*)

Bases: `object`

This class represents a monomial factor, that is, a derivative of a scalar component of a basis function.

**apply\_derivative** (*indices*)

**apply\_restriction** (*restriction*)

**element** ()

**index** ()

**replace\_indices** (*old\_indices, new\_indices*)

**class** `ffc.tensor.monomialextraction.MonomialForm`

Bases: `object`

This class represents a monomial form, that is, a sum of integrals, each represented as a `MonomialSum`.

**append** (*integrand*)

**class** `ffc.tensor.monomialextraction.MonomialSum` (*arg=None*)

Bases: `object`

This class represents a sum of monomials.

**apply\_derivative** (*indices*)

**apply\_indices** (*indices*)

**apply\_restriction** (*restriction*)

**apply\_tensor** (*indices*)

**class** `ffc.tensor.monomialextraction.MonomialTransformer` (*function\_replace\_map=None*)

Bases: `ufl.algorithms.transformer.ReuseTransformer`

This class defines the transformation rules for extraction of a monomial form represented as a `MonomialSum` from a UFL integral.

**argument** (*v*)

**coefficient** (*v*)

**component\_tensor** (*o, s, indices*)

**division** (*o*)  
**expr** (*o*, \**ops*)  
**grad** (*o*, *s*)  
**index\_sum** (*o*, *s*, *index*)  
**indexed** (*o*, *s*, *indices*)  
**multi\_index** (*multi\_index*)  
**negative\_restricted** (*o*, *s*)  
**positive\_restricted** (*o*, *s*)  
**power** (*o*, *s*, *ignored\_exponent\_expressed\_as\_sum*)  
**product** (*o*, *s0*, *s1*)  
**scalar\_value** (*x*)  
**sum** (*o*, *s0*, *s1*)  
**terminal** (*o*)  
**variable** (*o*)

`ffc.tensor.monomialextraction.extract_monomial_form` (*integrands*, *function\_replace\_map*)

Extract monomial representation of form (if possible). When successful, the form is represented as a sum of products of scalar components of basis functions or derivatives of basis functions. If unsuccessful, `MonomialException` is raised.

`ffc.tensor.monomialextraction.extract_monomial_integrand` (*integrand*, *function\_replace\_map*)

Extract monomial integrand (if possible).

## ffc.tensor.monomialintegration module

This module implements efficient integration of monomial forms.

`ffc.tensor.monomialintegration.integrate` (*monomial*, *integral\_type*, *facet0*, *facet1*, *quadrature\_degree*, *quadrature\_rule*, *cell*)

Compute the reference tensor for a given monomial term of a multilinear form

## ffc.tensor.monomialtransformation module

Transformation of monomial representations of UFL forms.

**class** `ffc.tensor.monomialtransformation.MonomialArgument` (*element*, *index*, *components*, *derivatives*, *restriction*)

Bases: `object`

This class represents a monomial argument, that is, a derivative of a scalar component of a basis function on the reference element.

**class** `ffc.tensor.monomialtransformation.MonomialCoefficient` (*index*, *number*)

Bases: `object`

This class represents a coefficient in a monomial.

**class** `ffc.tensor.monomialtransformation.MonomialDeterminant` (*power=None, restriction=None*)

Bases: `object`

This class represents a determinant factor in a monomial.

**class** `ffc.tensor.monomialtransformation.MonomialIndex` (*index=None, index\_type=None, index\_range=None, index\_id=None*)

Bases: `object`

This class represents a monomial index. Each index has a type, a range and a unique id. Valid index types are listed below.

**EXTERNAL** = 'external'

**FIXED** = 'fixed'

**INTERNAL** = 'internal'

**PRIMARY** = 'primary'

**SECONDARY** = 'secondary'

**class** `ffc.tensor.monomialtransformation.MonomialTransform` (*index0, index1, transform\_type, restriction, offset*)

Bases: `object`

This class represents a transform (mapping derivative) in a form.

**J** = 'J'

**JINV** = 'JINV'

**class** `ffc.tensor.monomialtransformation.TransformedMonomial` (*monomial*)

Bases: `object`

This class represents a monomial form after transformation to the reference element.

**extract\_unique\_indices** (*index\_type=None*)

Return all unique indices for monomial w.r.t. type and id (not range).

`ffc.tensor.monomialtransformation.transform_monomial_form` (*monomial\_form*)

Transform monomial form to reference element.

## ffc.tensor.multiindex module

**class** `ffc.tensor.multiindex.MultiIndex` (*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

`ffc.tensor.multiindex.build_indices` (*dims*)

Create a list of all index combinations.

`ffc.tensor.multiindex.create_multiindex` (*indices*)

Create multiindex for given list of indices.

`ffc.tensor.multiindex.outer_join` (*a, b*)

Let a be a list of lists and b a list. We append each element of b to each list in a and return the resulting list of lists.

## ffc.tensor.referencetensor module

**class** `ffc.tensor.referencetensor.ReferenceTensor` (*monomial*, *integral\_type*, *facet0*, *facet1*,  
*quadrature\_order*, *quadrature\_rule*,  
*cell*)

Bases: `object`

This class represents the reference tensor for a monomial term of a multilinear form.

## ffc.tensor.tensorgenerator module

Code generator for tensor representation

`ffc.tensor.tensorgenerator.generate_integral_code` (*ir*, *prefix*, *parameters*)  
Generate code for integral from intermediate representation.

## ffc.tensor.tensoroptimization module

`ffc.tensor.tensoroptimization.optimize_integral_ir` (*ir*, *parameters*)  
Compute optimized intermediate representation of integral.

## ffc.tensor.tensorreordering module

Reordering of entries in reference tensor for interior facets.

`ffc.tensor.tensorreordering.reorder_entries` (*terms*)  
Reorder entries to compute the reference tensor for an interior facet from the the reduced reference tensor.

## ffc.tensor.tensorrepresentation module

This module implements the representation of a multilinear form as a sum of tensor contractions.

The following possible optimizations are currently not implemented but might be (re-)implemented in a future version of FFC

1. Factorization of common reference tensors

`ffc.tensor.tensorrepresentation.compute_integral_ir` (*itg\_data*, *form\_data*, *form\_id*, *el-*  
*ement\_numbers*, *classnames*, *pa-*  
*rameters*)  
Compute intermediate representation of integral.

## Module contents

### ffc.tsfc package

### Submodules

### ffc.tsfc.tsfcgenerator module

## ffc.tsfc.tsfcoptimization module

## ffc.tsfc.tsfcrepresentation module

## 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*)

**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.CRSAarray` (*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*->*FV*index 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*:

$IM = \{ (ai1\_1, \dots, ai1\_r): fi1, (ai2\_1, \dots, ai2\_r): fi2, \}$

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

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

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)
```

```
    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, nindeps, 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 determining 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 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 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 flattened components of tensor and its underlying indexed subexpression.

`ffc.uflacs.analysis.indexing.map_indexed_arg_components(indexed)`

Build integer list mapping between flattened 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,    restric-
                                                             tion)

```

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*)

**coefficient** (*e*, *mt*, *tabledata*, *num\_points*)

**expr** (*e*, *mt*, *tabledata*, *num\_points*)

**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_volume (e, mt, tabledata, access)  
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 constant tables defined in ufc\_geometry.h.

**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*)

**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*)

**facet\_edge\_vectors** (*e, mt, tabledata, num\_points, access*)

These quantities refer to constant tables defined in ufc\_geometry.h.

**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  $x(X)$ .

$J = \sum_k x_{dof_k} \text{grad}_X \phi_k(X)$

**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\_volume** (*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:**  $x = \sum_k x_{dof\_k} x_{phi\_k}(X)$

**If reference facet coordinates are given:**  $x = \sum_k x_{dof\_k} x_{phi\_k}(X_f)$

## 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, interleaved\_components*)

**domain\_dofs\_access** (*gdim, num\_scalar\_dofs, restriction, interleaved\_components*)

**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.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_accumulation_loop (dst,  
expr,  
in-  
dices)`

`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, in-
                                                                dex_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*

**cell\_shape** (L, ir)

**compute\_geometry** (L, ir)

**compute\_jacobian\_determinants** (L, ir)

**compute\_jacobian\_inverses** (L, ir)

**compute\_jacobians** (L, ir)

**compute\_physical\_coordinates** (L, ir)

**compute\_reference\_coordinates** (L, ir)

**create\_coordinate\_dofmap** (L, ir)

**create\_coordinate\_finite\_element** (L, ir)

**geometric\_dimension** (L, ir)

Default implementation of returning geometric dimension fetched from ir.

**topological\_dimension** (L, ir)

Default implementation of returning topological dimension fetched from ir.

## ffc.uflacs.backends.ufc.dofmap module

```
class ffc.uflacs.backends.ufc.dofmap.ufc_dofmap
```

Bases: *ffc.uflacs.backends.ufc.generator.ufc\_generator*

**create\_sub\_dofmap** (L, ir)

**global\_dimension** (L, ir)

**needs\_mesh\_entities** (L, ir)

**num\_element\_dofs** (L, ir)

**num\_entity\_closure\_dofs** (L, ir)

**num\_entity\_dofs** (L, ir)

**num\_facet\_dofs** (L, ir)

**num\_sub\_dofmaps** (L, ir)

**tabulate\_dofs** (L, ir)

**tabulate\_entity\_closure\_dofs** (L, ir)

**tabulate\_entity\_dofs** (L, ir)

**tabulate\_facet\_dofs** (L, ir)

**topological\_dimension** (L, ir)

Default implementation of returning topological dimension fetched from ir.

**ffc.uflacs.backends.ufc.evaluatebasis module**

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

`ffc.uflacs.backends.ufc.evaluatebasis.generate_evaluate_reference_basis` (*L*, *data*)

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.finite\_element module**

`ffc.uflacs.backends.ufc.finite_element.affine_weights` (*dim*)

Compute coefficients for mapping from reference to physical element

**class** `ffc.uflacs.backends.ufc.finite_element.ufc_finite_element`

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

`cell_shape` (*L*, *ir*)

`create_sub_element` (*L*, *ir*)

`degree` (*L*, *ir*)

`evaluate_basis` (*L*, *ir*)

`evaluate_basis_all` (*L*, *ir*)

`evaluate_basis_derivatives` (*L*, *ir*)

`evaluate_basis_derivatives_all` (*L*, *ir*)

`evaluate_dof` (*L*, *ir*)

`evaluate_dofs` (*L*, *ir*)

Generate code for evaluate\_dofs.

`evaluate_reference_basis` (*L*, *ir*)

`evaluate_reference_basis_derivatives` (*L*, *ir*)

`family` (*L*, *ir*)

`geometric_dimension` (*L*, *ir*)

`interpolate_vertex_values` (*L*, *ir*)

`num_sub_elements` (*L*, *ir*)

`reference_value_dimension` (*L*, *ir*)

`reference_value_rank` (*L*, *ir*)

`reference_value_size` (*L*, *ir*)

`space_dimension` (*L*, *ir*)

`tabulate_dof_coordinates` (*L*, *ir*)

```
topological_dimension (L, ir)
value_dimension (L, ir)
value_rank (L, ir)
value_size (L, ir)
```

## 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'`.

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

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

```
create_cell_integral (L, ir, integral_type='cell', declname='create_cell_integral',
                      impl=<function ufc_form._create_foo_integral>)
    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, integral_type='custom', declname='create_custom_integral',
                        impl=<function ufc_form._create_foo_integral>)
    Return implementation of ufc::form::create_custom_integral().
```

```
create_cutcell_integral (L, ir, integral_type='cutcell', declname='create_cutcell_integral',
                        impl=<function ufc_form._create_foo_integral>)
    Return implementation of ufc::form::create_cutcell_integral().
```

```
create_default_cell_integral (L, ir, integral_type='cell', declname='create_default_cell_integral',
                             impl=<function ufc_form._create_default_foo_integral>)
    Return implementation of ufc::form::create_default_cell_integral().
```

```
create_default_custom_integral (L, ir, integral_type='custom', declname='create_default_custom_integral',
                                impl=<function ufc_form._create_default_foo_integral>)
    Return implementation of ufc::form::create_default_custom_integral().
```

```
create_default_cutcell_integral (L, ir, integral_type='cutcell', declname='create_default_cutcell_integral',
                                 impl=<function ufc_form._create_default_foo_integral>)
    Return implementation of ufc::form::create_default_cutcell_integral().
```

```
create_default_exterior_facet_integral (L, ir, integral_type='exterior_facet', declname='create_default_exterior_facet_integral',
                                         impl=<function ufc_form._create_default_foo_integral>)
    Return implementation of ufc::form::create_default_exterior_facet_integral().
```

**create\_default\_interface\_integral** (*L*, *ir*, *integral\_type*='interface', *declname*='create\_default\_interface\_integral', *impl*=<function *ufc\_form.\_create\_default\_foo\_integral*>)  
 Return implementation of *ufc::form::create\_default\_interface\_integral*().

**create\_default\_interior\_facet\_integral** (*L*, *ir*, *integral\_type*='interior\_facet', *declname*='create\_default\_interior\_facet\_integral', *impl*=<function *ufc\_form.\_create\_default\_foo\_integral*>)  
 Return implementation of *ufc::form::create\_default\_interior\_facet\_integral*().

**create\_default\_overlap\_integral** (*L*, *ir*, *integral\_type*='overlap', *declname*='create\_default\_overlap\_integral', *impl*=<function *ufc\_form.\_create\_default\_foo\_integral*>)  
 Return implementation of *ufc::form::create\_default\_overlap\_integral*().

**create\_default\_vertex\_integral** (*L*, *ir*, *integral\_type*='vertex', *declname*='create\_default\_vertex\_integral', *impl*=<function *ufc\_form.\_create\_default\_foo\_integral*>)  
 Return implementation of *ufc::form::create\_default\_vertex\_integral*().

**create\_dofmap** (*L*, *ir*)

**create\_exterior\_facet\_integral** (*L*, *ir*, *integral\_type*='exterior\_facet', *declname*='create\_exterior\_facet\_integral', *impl*=<function *ufc\_form.\_create\_foo\_integral*>)  
 Return implementation of *ufc::form::create\_exterior\_facet\_integral*().

**create\_finite\_element** (*L*, *ir*)

**create\_interface\_integral** (*L*, *ir*, *integral\_type*='interface', *declname*='create\_interface\_integral', *impl*=<function *ufc\_form.\_create\_foo\_integral*>)  
 Return implementation of *ufc::form::create\_interface\_integral*().

**create\_interior\_facet\_integral** (*L*, *ir*, *integral\_type*='interior\_facet', *declname*='create\_interior\_facet\_integral', *impl*=<function *ufc\_form.\_create\_foo\_integral*>)  
 Return implementation of *ufc::form::create\_interior\_facet\_integral*().

**create\_overlap\_integral** (*L*, *ir*, *integral\_type*='overlap', *declname*='create\_overlap\_integral', *impl*=<function *ufc\_form.\_create\_foo\_integral*>)  
 Return implementation of *ufc::form::create\_overlap\_integral*().

**create\_vertex\_integral** (*L*, *ir*, *integral\_type*='vertex', *declname*='create\_vertex\_integral', *impl*=<function *ufc\_form.\_create\_foo\_integral*>)  
 Return implementation of *ufc::form::create\_vertex\_integral*().

**geometric\_dimension** (*L*, *ir*)  
 Default implementation of returning geometric dimension fetched from *ir*.

**has\_cell\_integrals** (*L*, *ir*, *integral\_type*='cell', *declname*='has\_cell\_integrals', *impl*=<function *ufc\_form.\_has\_foo\_integrals*>)  
 Return implementation of *ufc::form::has\_cell\_integrals*().

**has\_custom\_integrals** (*L*, *ir*, *integral\_type*='custom', *declname*='has\_custom\_integrals', *impl*=<function *ufc\_form.\_has\_foo\_integrals*>)  
 Return implementation of *ufc::form::has\_custom\_integrals*().

**has\_cutcell\_integrals** (*L*, *ir*, *integral\_type*='cutcell', *declname*='has\_cutcell\_integrals', *impl*=<function *ufc\_form.\_has\_foo\_integrals*>)  
 Return implementation of *ufc::form::has\_cutcell\_integrals*().

**has\_exterior\_facet\_integrals** (*L*, *ir*, *integral\_type*='exterior\_facet', *declname*='has\_exterior\_facet\_integrals', *impl*=<function *ufc\_form.has\_foo\_integrals*>)

Return implementation of `ufc::form::has_exterior_facet_integrals()`.

**has\_interface\_integrals** (*L*, *ir*, *integral\_type*='interface', *declname*='has\_interface\_integrals', *impl*=<function *ufc\_form.has\_foo\_integrals*>)

Return implementation of `ufc::form::has_interface_integrals()`.

**has\_interior\_facet\_integrals** (*L*, *ir*, *integral\_type*='interior\_facet', *declname*='has\_interior\_facet\_integrals', *impl*=<function *ufc\_form.has\_foo\_integrals*>)

Return implementation of `ufc::form::has_interior_facet_integrals()`.

**has\_overlap\_integrals** (*L*, *ir*, *integral\_type*='overlap', *declname*='has\_overlap\_integrals', *impl*=<function *ufc\_form.has\_foo\_integrals*>)

Return implementation of `ufc::form::has_overlap_integrals()`.

**has\_vertex\_integrals** (*L*, *ir*, *integral\_type*='vertex', *declname*='has\_vertex\_integrals', *impl*=<function *ufc\_form.has\_foo\_integrals*>)

Return implementation of `ufc::form::has_vertex_integrals()`.

**max\_cell\_subdomain\_id** (*L*, *ir*, *integral\_type*='cell', *declname*='max\_cell\_subdomain\_id', *impl*=<function *ufc\_form.max\_foo\_subdomain\_id*>)

Return implementation of `ufc::form::max_cell_subdomain_id()`.

**max\_custom\_subdomain\_id** (*L*, *ir*, *integral\_type*='custom', *declname*='max\_custom\_subdomain\_id', *impl*=<function *ufc\_form.max\_foo\_subdomain\_id*>)

Return implementation of `ufc::form::max_custom_subdomain_id()`.

**max\_cutcell\_subdomain\_id** (*L*, *ir*, *integral\_type*='cutcell', *declname*='max\_cutcell\_subdomain\_id', *impl*=<function *ufc\_form.max\_foo\_subdomain\_id*>)

Return implementation of `ufc::form::max_cutcell_subdomain_id()`.

**max\_exterior\_facet\_subdomain\_id** (*L*, *ir*, *integral\_type*='exterior\_facet', *declname*='max\_exterior\_facet\_subdomain\_id', *impl*=<function *ufc\_form.max\_foo\_subdomain\_id*>)

Return implementation of `ufc::form::max_exterior_facet_subdomain_id()`.

**max\_interface\_subdomain\_id** (*L*, *ir*, *integral\_type*='interface', *declname*='max\_interface\_subdomain\_id', *impl*=<function *ufc\_form.max\_foo\_subdomain\_id*>)

Return implementation of `ufc::form::max_interface_subdomain_id()`.

**max\_interior\_facet\_subdomain\_id** (*L*, *ir*, *integral\_type*='interior\_facet', *declname*='max\_interior\_facet\_subdomain\_id', *impl*=<function *ufc\_form.max\_foo\_subdomain\_id*>)

Return implementation of `ufc::form::max_interior_facet_subdomain_id()`.

**max\_overlap\_subdomain\_id** (*L*, *ir*, *integral\_type*='overlap', *declname*='max\_overlap\_subdomain\_id', *impl*=<function *ufc\_form.max\_foo\_subdomain\_id*>)

Return implementation of `ufc::form::max_overlap_subdomain_id()`.

**max\_vertex\_subdomain\_id** (*L*, *ir*, *integral\_type*='vertex', *declname*='max\_vertex\_subdomain\_id', *impl*=<function *ufc\_form.max\_foo\_subdomain\_id*>)

Return implementation of `ufc::form::max_vertex_subdomain_id()`.

**num\_coefficients** (*L*, *ir*)

**original\_coefficient\_position** (*L*, *ir*)

**rank** (*L*, *ir*)

**topological\_dimension** (*L*, *ir*)

Default implementation of returning topological dimension fetched from *ir*.

### **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 template keywords and inserts the results from calls to `self.<keyword>(language, ir)`, or the value of `ir[keyword]` if there is no `self.<keyword>`.

**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*, *snippets=None*)

Return composition of templates with generated snippets.

**generate\_snippets** (*L*, *ir*)

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

    enabled_coefficients(L, ir)

    tabulate_tensor(L, ir)

    tabulate_tensor_comment(L, ir)

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.templates module

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

## ffc.uflacs.backends.ufc.utils module

```

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, fac-
                                                         tory=False)

```

## 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, 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)
    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)  
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).

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)

    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)

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)

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

    cs_format (precision=None)
```

```
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)

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)

class ffc.uflacs.language.cnodes.ElseIf(condition, body)
    Bases: ffc.uflacs.language.cnodes.CStatement
    body
    condition
    cs_format (precision=None)

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
    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)
    init
```

**pragma**

**update**

**class** `ffc.uflacs.language.cnodes.ForRange` (*index, begin, end, body, 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*)

**end**

**index**

**index\_type**

**pragma**

**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*)

**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.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*)

**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

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)

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)
    Bases: ffc.uflacs.language.cnodes.CStatement

    cs_format(precision=None)

    value

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

    body

    cs_format(precision=None)

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

```
    op = 'sizeof'
    precedence = 3

class ffc.uflacs.language.cnodes.Statement(expr)
    Bases: ffc.uflacs.language.cnodes.CStatement
    Make an expression into a statement.
    cs_format (precision=None)
    expr

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)
    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, auto-
                                     scope=True)
    Bases: ffc.uflacs.language.cnodes.CStatement
    arg
    autobreak
    autoscope
    cases
    cs_format (precision=None)
    default

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)
```

**exception**

**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*)

**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*)

**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*)

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

**body**

**condition**

**cs\_format** (*precision=None*)

`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_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.commentated_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 string.

- 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, re-  
                                                    strictions, 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, fac-  
                                                           tor_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,      fac-
                                                             tor_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)

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][\_FIV][\_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 table_origin
```

```
ffc.uflacs.elementtables.unique_table_reference_t
alias of 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 occurring 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 representation of integral.

## Module contents

This is UFLACS, the UFL Analyser and Compiler System.

## Submodules

### ffc.analysis module

`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

## 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.

## ffc.codesnippets module

Code snippets for code generation.

## 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.

## ffc.cpp module

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

`ffc.cpp.count_ops(code)`

Count the number of operations in code (multiply-add pairs).

`ffc.cpp.indent(block, num_spaces)`

Indent each row of the given string block with n spaces.

`ffc.cpp.make_classname(prefix, basename, signature)`

`ffc.cpp.make_integral_classname(prefix, integral_type, form_id, subdomain_id)`

```
ffc.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.cpp.set_exception_handling (convert_exceptions_to_warnings)
```

Set handling of exceptions.

```
ffc.cpp.set_float_formatting (precision)
```

Set floating point formatting based on precision.

## ffc.enrichedelement module

```
class ffc.enrichedelement.EnrichedElement (elements)
```

Bases: `object`

Create the space spanned by a list of ffc elements.

`degree ()`

`dual_basis ()`

`elements ()`

`entity_dofs ()`

`mapping ()`

`space_dimension ()`

`tabulate (order, points)`

`value_shape ()`

```
class ffc.enrichedelement.SpaceOfReals (element)
```

Bases: `object`

`degree ()`

`dmats ()`

`dual_basis ()`

`entity_dofs ()`

`get_coeffs ()`

`get_num_members (arg)`

`mapping ()`

`space_dimension ()`

`tabulate (order, points)`

`value_shape ()`

## ffc.evaluatebasis module

Code generation for evaluation of finite element basis values. This module generates code which is more or less a C++ representation of the code found in FIAT.

## ffc.evaluatebasisderivatives module

Code generation for evaluation of derivatives of finite element basis values. This module generates code which is more or less a C++ representation of the code found in FIAT\_NEW.

## ffc.evaluatedof module

Code generation for evaluate\_dof.

This module generates the functions evaluate\_dof and evaluate\_dofs. These evaluate the degree of freedom (dof) number  $i$  and all degrees of freedom for an element respectively.

Each dof  $L$  is assumed to act on a field  $f$  in the following manner:

$$L(f) = w_{\{j, k\}} f_k(x_j)$$

where  $w$  is a set of weights,  $j$  is an index set corresponding to the number of points involved in the evaluation of the functional, and  $k$  is a multi-index set with rank corresponding to the value rank of the function  $f$ .

For common degrees of freedom such as point evaluations and directional component evaluations, there is just one point. However, for various integral moments, the integrals are evaluated using quadrature. The number of points therefore correspond to the quadrature points.

The points  $x_j$ , weights  $w_{\{j, k\}}$  and components  $k$  are extracted from FIAT (functional.pt\_dict) in the intermediate representation stage.

```
ffc.evaluatedof.evaluate_dof_and_dofs(ir)
    Generate code for evaluate_dof and evaluate_dof.
```

```
ffc.evaluatedof.affine_weights(dim)
    Compute coefficients for mapping from reference to physical element
```

## ffc.extras module

This modules provides additional functionality for users of FFC.

```
ffc.extras.compute_tensor_representation(form)
    Compute tensor representation for given form. This function may be useful for those (Hi Matt!) that want to
    access the FFC tensor representation from outside FFC.
```

## ffc.fiatinterface module

```
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)
    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(dim)
```

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

## 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)
```

## ffc.interpolatevertexvalues module

Code generation for `interpolate_vertex_values`.

```
ffc.interpolatevertexvalues.interpolate_vertex_values (ir)  
    Generate code for interpolate_vertex_values.
```

## 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.
```

## 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)
```

## 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).
```

## ffc.mixedelement module

```
class ffc.mixedelement.MixedElement(elements)
    Bases: object

    Create a FFC mixed element from a list of FFC/FIAT elements.

    dual_basis()

    elements()

    entity_dofs()

    mapping()

    num_components()

    space_dimension()

    tabulate(order, points)
        Tabulate values on mixed element by appropriately reordering the tabulated values for the nested elements.
        The table of values is organized as follows:
            
$$D^a v_i[j](x_k) = \text{table}[a][i][j][k]$$

        where a is a multi-index (tuple) representing the derivative. For example, a = (1, 1) represents d/dx d/dy.

    value_shape()
```

## 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.
```

## 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.

## ffc.plot module

This module provides functionality for plotting finite elements.

`ffc.plot.plot(element, rotate=True)`  
Plot finite element.

## ffc.quadratureelement module

`class ffc.quadratureelement.QuadratureElement(ufl_element)`  
Bases: `object`

Write description of QuadratureElement

`dual_basis()`  
Return list of PointEvaluations

`entity_dofs()`  
Entity dofs are like that of DG, all internal to the cell

`mapping()`  
The mapping is not really affine, but it is easier to handle the code generation this way.

`space_dimension()`  
The element space dimension is simply the number of quadrature points

`tabulate(order, points)`  
Return the identity matrix of size (num\_quad\_points, num\_quad\_points), in a format that monomialintegration and monomialtabulation understands.

`value_shape()`  
The QuadratureElement is scalar valued

## 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.
```

## 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’.
```

## ffc.restrictedelement module

```
class ffc.restrictedelement.RestrictedElement (element, indices, restriction_domain)
    Bases: object

    Create a restriction of a given FIAT element.

    degree ()
```



```
dmats ()  
dual_basis ()  
entity_dofs ()  
get_coeffs ()  
get_num_members (arg)  
mapping ()  
restriction_domain ()  
space_dimension ()  
tabulate (order, points)  
value_shape ()
```

## 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.
```

## ffc.wrappers module

```
ffc.wrappers.generate_wrapper_code (analysis, prefix, object_names, parameters)  
    Generate code for additional wrappers.
```

## 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`)

## Release notes

### 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).

---

### 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

### 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  $\geq 4.8$
- 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`.

### 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  $\geq 4.8$
- Add function `ufc_signature()` to the form compiler interface
- Add function `git_commit_hash()`

### 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](#)



### f

`ffc`, 77  
`ffc.analysis`, 67  
`ffc.backends`, 7  
`ffc.backends.dolfin`, 6  
`ffc.backends.dolfin.capsules`, 5  
`ffc.backends.dolfin.form`, 5  
`ffc.backends.dolfin.functionspace`, 5  
`ffc.backends.dolfin.goalfunctional`, 6  
`ffc.backends.dolfin.includes`, 6  
`ffc.backends.dolfin.wrappers`, 6  
`ffc.backends.ufc`, 7  
`ffc.backends.ufc.coordinate_mapping`, 6  
`ffc.backends.ufc.dofmap`, 6  
`ffc.backends.ufc.finite_element`, 6  
`ffc.backends.ufc.form`, 6  
`ffc.backends.ufc.function`, 6  
`ffc.backends.ufc.integrals`, 7  
`ffc.codegeneration`, 68  
`ffc.codesnippets`, 68  
`ffc.compiler`, 68  
`ffc.cpp`, 69  
`ffc.enrichedelement`, 70  
`ffc.errorcontrol`, 9  
`ffc.errorcontrol.errorcontrol`, 8  
`ffc.errorcontrol.errorcontrolgenerators`, 8  
`ffc.evaluatebasis`, 70  
`ffc.evaluatebasisderivatives`, 71  
`ffc.evaluatedof`, 71  
`ffc.extras`, 71  
`ffc.fiatinterface`, 71  
`ffc.formatting`, 72  
`ffc.interpolatevertexvalues`, 72  
`ffc.jitcompiler`, 72  
`ffc.log`, 72  
`ffc.main`, 73  
`ffc.mixedelement`, 74  
`ffc.optimization`, 74  
`ffc.parameters`, 74  
`ffc.plot`, 75  
`ffc.quadrature`, 20  
`ffc.quadrature.expr`, 9  
`ffc.quadrature.floatvalue`, 10  
`ffc.quadrature.fraction`, 10  
`ffc.quadrature.optimisedquadraturetransformer`, 10  
`ffc.quadrature.parameters`, 12  
`ffc.quadrature.product`, 12  
`ffc.quadrature.quadraturegenerator`, 12  
`ffc.quadrature.quadratureoptimization`, 12  
`ffc.quadrature.quadraturerepresentation`, 13  
`ffc.quadrature.quadraturetransformer`, 13  
`ffc.quadrature.quadraturetransformerbase`, 14  
`ffc.quadrature.quadratureutils`, 17  
`ffc.quadrature.reduce_operations`, 17  
`ffc.quadrature.sumobj`, 19  
`ffc.quadrature.symbol`, 19  
`ffc.quadrature.symbolics`, 20  
`ffc.quadrature.tabulate_basis`, 20  
`ffc.quadratureelement`, 75  
`ffc.representation`, 75  
`ffc.representationutils`, 76  
`ffc.restrictedelement`, 76  
`ffc.tensor`, 24  
`ffc.tensor.costestimation`, 20  
`ffc.tensor.geometrytensor`, 20  
`ffc.tensor.monomialextraction`, 21  
`ffc.tensor.monomialintegration`, 22  
`ffc.tensor.monomialtransformation`, 22  
`ffc.tensor.multiindex`, 23  
`ffc.tensor.referencetensor`, 24  
`ffc.tensor.tensorgenerator`, 24  
`ffc.tensor.tensoroptimization`, 24  
`ffc.tensor.tensorreordering`, 24

- `ffc.tensor.tensorrepresentation`, [24](#)
- `ffc.uflacs`, [67](#)
- `ffc.uflacs.analysis`, [33](#)
- `ffc.uflacs.analysis.balancing`, [25](#)
- `ffc.uflacs.analysis.crsarray`, [25](#)
- `ffc.uflacs.analysis.dependencies`, [26](#)
- `ffc.uflacs.analysis.expr_shapes`, [26](#)
- `ffc.uflacs.analysis.factorization`, [26](#)
- `ffc.uflacs.analysis.graph`, [27](#)
- `ffc.uflacs.analysis.graph_rebuild`, [28](#)
- `ffc.uflacs.analysis.graph_ssa`, [29](#)
- `ffc.uflacs.analysis.graph_symbols`, [30](#)
- `ffc.uflacs.analysis.graph_vertices`, [30](#)
- `ffc.uflacs.analysis.indexing`, [30](#)
- `ffc.uflacs.analysis.modified_terminals`,  
[31](#)
- `ffc.uflacs.analysis.valuenumbering`, [31](#)
- `ffc.uflacs.backends`, [44](#)
- `ffc.uflacs.backends.ffc`, [37](#)
- `ffc.uflacs.backends.ffc.access`, [33](#)
- `ffc.uflacs.backends.ffc.backend`, [34](#)
- `ffc.uflacs.backends.ffc.common`, [34](#)
- `ffc.uflacs.backends.ffc.definitions`, [34](#)
- `ffc.uflacs.backends.ffc.symbols`, [36](#)
- `ffc.uflacs.backends.ufc`, [44](#)
- `ffc.uflacs.backends.ufc.coordinate_mapping`,  
[37](#)
- `ffc.uflacs.backends.ufc.dofmap`, [38](#)
- `ffc.uflacs.backends.ufc.evaluatebasis`,  
[39](#)
- `ffc.uflacs.backends.ufc.finite_element`,  
[39](#)
- `ffc.uflacs.backends.ufc.form`, [40](#)
- `ffc.uflacs.backends.ufc.generator`, [43](#)
- `ffc.uflacs.backends.ufc.generators`, [43](#)
- `ffc.uflacs.backends.ufc.integrals`, [43](#)
- `ffc.uflacs.backends.ufc.templates`, [44](#)
- `ffc.uflacs.backends.ufc.utils`, [44](#)
- `ffc.uflacs.build_uflacs_ir`, [60](#)
- `ffc.uflacs.elementtables`, [64](#)
- `ffc.uflacs.integralgenerator`, [65](#)
- `ffc.uflacs.language`, [60](#)
- `ffc.uflacs.language.cnodes`, [45](#)
- `ffc.uflacs.language.format_lines`, [57](#)
- `ffc.uflacs.language.format_value`, [57](#)
- `ffc.uflacs.language.precedence`, [57](#)
- `ffc.uflacs.language.ufl_to_cnodes`, [58](#)
- `ffc.uflacs.params`, [67](#)
- `ffc.uflacs.tools`, [67](#)
- `ffc.uflacs.uflacsgenerator`, [67](#)
- `ffc.uflacs.uflacsoptimization`, [67](#)
- `ffc.uflacs.uflacsrepresentation`, [67](#)
- `ffc.utils`, [77](#)
- `ffc.wrappers`, [77](#)

## A

- abs() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt method), 10
- abs() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 13
- abs() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 14
- abs() (ffc.uflacs.analysis.graph\_rebuild.ReconstructScalarSubexpressions method), 28
- abs() (ffc.uflacs.language.ufl\_to\_cnodes.RulesForC method), 58
- abs() (ffc.uflacs.language.ufl\_to\_cnodes.RulesForCpp method), 59
- accumulate\_integrals() (in module ffc.uflacs.tools), 67
- accumulate\_loop() (in module ffc.uflacs.language.cnodes), 55
- acos() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 14
- acos() (ffc.uflacs.language.ufl\_to\_cnodes.UFL2CNodesMixin method), 59
- Add (class in ffc.uflacs.language.cnodes), 45
- ADD (ffc.uflacs.language.precedence.PRECEDENCE attribute), 57
- add() (ffc.uflacs.analysis.factorization.Factors method), 26
- add\_indent() (in module ffc.log), 72
- add\_logfile() (in module ffc.log), 73
- add\_to\_fv() (in module ffc.uflacs.analysis.factorization), 27
- add\_ufc\_form\_integral\_methods() (in module ffc.uflacs.backends.ufc.form), 40
- AddressOf (class in ffc.uflacs.language.cnodes), 45
- ADDRESSOF (ffc.uflacs.language.precedence.PRECEDENCE attribute), 57
- adj\_expr\_2x2() (in module ffc.uflacs.backends.ufc.coordinate\_mapping), 37
- adj\_expr\_3x3() (in module ffc.uflacs.backends.ufc.coordinate\_mapping), 37
- affine\_weights() (in module ffc.evaluatedof), 71
- affine\_weights() (in module ffc.uflacs.backends.ufc.finite\_element), 39
- alignas (ffc.uflacs.language.cnodes.ArrayDecl attribute), 45
- all\_elements() (in module ffc.representation), 75
- all\_equal() (in module ffc.utils), 77
- all\_ufc\_classnames() (in module ffc.backends.ufc), 7
- allocate\_registers() (in module ffc.uflacs.analysis.graph\_ssa), 29
- analyse\_dependencies() (in module ffc.uflacs.build\_uflacs\_ir), 60
- analyse\_modified\_terminal() (in module ffc.uflacs.analysis.modified\_terminals), 31
- analyse\_table\_type() (in module ffc.uflacs.elementtables), 64
- analyse\_table\_types() (in module ffc.uflacs.elementtables), 64
- analyze\_coordinate\_mappings() (in module ffc.analysis), 67
- analyze\_elements() (in module ffc.analysis), 67
- analyze\_forms() (in module ffc.analysis), 67
- analyze\_ufl\_objects() (in module ffc.analysis), 68
- And (class in ffc.uflacs.language.cnodes), 45
- AND (ffc.uflacs.language.precedence.PRECEDENCE attribute), 57
- and\_condition() (ffc.uflacs.language.ufl\_to\_cnodes.UFL2CNodesMixin method), 59
- append() (ffc.tensor.monomialextraction.MonomialForm method), 21
- apply\_derivative() (ffc.tensor.monomialextraction.Monomial method), 21
- apply\_derivative() (ffc.tensor.monomialextraction.MonomialFactor method), 21
- apply\_derivative() (ffc.tensor.monomialextraction.MonomialSum method), 21
- apply\_function\_space\_template() (in module ffc.backends.dolfin.functionspace), 5
- apply\_indices() (ffc.tensor.monomialextraction.Monomial

`method`), 21  
`apply_indices()` (`ffc.tensor.monomialextraction.MonomialSum` method), 21  
`apply_multimesh_function_space_template()` (in module `ffc.backends.dofin.functionspace`), 5  
`apply_restriction()` (`ffc.tensor.monomialextraction.MonomialSum` method), 21  
`apply_restriction()` (`ffc.tensor.monomialextraction.MonomialSum` method), 21  
`apply_restriction()` (`ffc.tensor.monomialextraction.MonomialSum` method), 21  
`apply_tensor()` (`ffc.tensor.monomialextraction.MonomialSum` method), 21  
`apply_tensor()` (`ffc.tensor.monomialextraction.MonomialSum` method), 21  
`arg` (`ffc.uflacs.language.cnodes.Switch` attribute), 54  
`arg` (`ffc.uflacs.language.cnodes.UnaryOp` attribute), 55  
`args` (`ffc.uflacs.language.cnodes.NaryOp` attribute), 52  
`argument()` (`ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase` method), 14  
`argument()` (`ffc.tensor.monomialextraction.MonomialTransformation` method), 21  
`argument()` (`ffc.uflacs.backends.fcc.definitions.FFCBackendDefinition` method), 34  
`argument_loop_index()` (`ffc.uflacs.backends.fcc.symbols.FFCBackendSymbol` method), 36  
`argument_ordering_key()` (`ffc.uflacs.analysis.modified_terminals.ModifiedTerminal` method), 31  
`arguments` (`ffc.uflacs.language.cnodes.Call` attribute), 48  
`array` (`ffc.uflacs.language.cnodes.ArrayAccess` attribute), 45  
`array` (`ffc.uflacs.language.cnodes.FlattenedArray` attribute), 49  
`ArrayAccess` (class in `ffc.uflacs.language.cnodes`), 45  
`ArrayDecl` (class in `ffc.uflacs.language.cnodes`), 45  
`as_cexpr()` (in module `ffc.uflacs.language.cnodes`), 55  
`as_cstatement()` (in module `ffc.uflacs.language.cnodes`), 56  
`as_pragma()` (in module `ffc.uflacs.language.cnodes`), 56  
`as_symbol()` (in module `ffc.uflacs.language.cnodes`), 56  
`as_tuple()` (`ffc.uflacs.analysis.modified_terminals.ModifiedTerminal` method), 31  
`asin()` (`ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase` method), 14  
`asin()` (`ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin` method), 59  
`Assign` (class in `ffc.uflacs.language.cnodes`), 45  
`ASSIGN` (`ffc.uflacs.language.precedence.PRECEDENCE` attribute), 57  
`assign_loop()` (in module `ffc.uflacs.language.cnodes`), 56  
`AssignAdd` (class in `ffc.uflacs.language.cnodes`), 45  
`AssignAnd` (class in `ffc.uflacs.language.cnodes`), 45  
`AssignBitAnd` (class in `ffc.uflacs.language.cnodes`), 46  
`AssignBitOr` (class in `ffc.uflacs.language.cnodes`), 46  
`AssignDiv` (class in `ffc.uflacs.language.cnodes`), 46  
`AssignLShift` (class in `ffc.uflacs.language.cnodes`), 46  
`AssignMod` (class in `ffc.uflacs.language.cnodes`), 46  
`AssignMul` (class in `ffc.uflacs.language.cnodes`), 46  
`AssignOp` (class in `ffc.uflacs.language.cnodes`), 46  
`AssignOr` (class in `ffc.uflacs.language.cnodes`), 46  
`AssignRShift` (class in `ffc.uflacs.language.cnodes`), 46  
`AssignSub` (class in `ffc.uflacs.language.cnodes`), 46  
`atan()` (`ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase` method), 14  
`atan()` (`ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin` method), 59  
`atan_2()` (`ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase` method), 14  
`atan_2()` (`ffc.uflacs.analysis.graph_rebuild.ReconstructScalarSubexpression` method), 28  
`atan_2()` (`ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin` method), 59  
`atan_2_function()` (`ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase` method), 14  
`AttributeError` (`ffc.uflacs.language.cnodes.Switch` attribute), 54  
`BackendSymbol` (`ffc.uflacs.language.cnodes.Switch` attribute), 54  
**B**  
`balance_modified_terminal()` (in module `ffc.uflacs.analysis.balancing`), 25  
`balance_modifiers()` (in module `ffc.uflacs.analysis.balancing`), 25  
`BalanceModifiers` (class in `ffc.uflacs.analysis.balancing`), 25  
`base_expr` (`ffc.quadrature.symbol.Symbol` attribute), 19  
`base_op` (`ffc.quadrature.symbol.Symbol` attribute), 19  
`begin` (`ffc.uflacs.language.cnodes.ForRange` attribute), 50  
`begin()` (in module `ffc.log`), 73  
`bessel_function()` (`ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase` method), 15  
`bessel_function()` (`ffc.uflacs.analysis.graph_rebuild.ReconstructScalarSubexpression` method), 28  
`bessel_i()` (`ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase` method), 15  
`bessel_i()` (`ffc.uflacs.language.ufl_to_cnodes.RulesForCpp` method), 59  
`bessel_j()` (`ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase` method), 15  
`bessel_j()` (`ffc.uflacs.language.ufl_to_cnodes.RulesForCpp` method), 59  
`bessel_k()` (`ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase` method), 15  
`bessel_k()` (`ffc.uflacs.language.ufl_to_cnodes.RulesForCpp` method), 59



bessel\_y() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 15

bessel\_y() (ffc.uflacs.language.ufl\_to\_cnodes.RulesForCpp method), 59

binary\_condition() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformer method), 10

binary\_condition() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 13

BinOp (class in ffc.uflacs.language.cnodes), 46

BIT\_AND (ffc.uflacs.language.precedence.PRECEDENCE attribute), 57

BIT\_NOT (ffc.uflacs.language.precedence.PRECEDENCE attribute), 58

BIT\_OR (ffc.uflacs.language.precedence.PRECEDENCE attribute), 58

BIT\_XOR (ffc.uflacs.language.precedence.PRECEDENCE attribute), 58

BitAnd (class in ffc.uflacs.language.cnodes), 47

BitNot (class in ffc.uflacs.language.cnodes), 47

BitOr (class in ffc.uflacs.language.cnodes), 47

BitXor (class in ffc.uflacs.language.cnodes), 47

block\_mode (ffc.uflacs.build\_uflacs\_ir.common\_block\_data\_t attribute), 60

block\_mode (ffc.uflacs.build\_uflacs\_ir.full\_block\_data\_t attribute), 61

block\_mode (ffc.uflacs.build\_uflacs\_ir.partial\_block\_data\_t attribute), 62

block\_mode (ffc.uflacs.build\_uflacs\_ir.preintegrated\_block\_data\_t attribute), 62

block\_mode (ffc.uflacs.build\_uflacs\_ir.premultiplied\_block\_data\_t attribute), 63

body (ffc.uflacs.language.cnodes.Do attribute), 49

body (ffc.uflacs.language.cnodes.Else attribute), 49

body (ffc.uflacs.language.cnodes.ElseIf attribute), 49

body (ffc.uflacs.language.cnodes.For attribute), 49

body (ffc.uflacs.language.cnodes.ForRange attribute), 50

body (ffc.uflacs.language.cnodes.If attribute), 50

body (ffc.uflacs.language.cnodes.Namespace attribute), 51

body (ffc.uflacs.language.cnodes.Scope attribute), 53

body (ffc.uflacs.language.cnodes.While attribute), 55

body (ffc.uflacs.language.format\_lines.Indented attribute), 57

Break (class in ffc.uflacs.language.cnodes), 47

build\_1d\_initializer\_list() (in module ffc.uflacs.language.cnodes), 56

build\_argument\_dependencies() (in module ffc.uflacs.analysis.factorization), 27

build\_argument\_indices() (in module ffc.uflacs.analysis.factorization), 27

build\_array\_from\_counts() (in module ffc.uflacs.analysis.graph\_vertices), 30

build\_element\_tables() (in module ffc.uflacs.elementtables), 64

build\_element\_tables() (in module ffc.uflacs.analysis.graph), 28

build\_graph\_symbols() (in module ffc.uflacs.analysis.graph\_symbols), 30

build\_graph\_vertices() (in module ffc.uflacs.analysis.graph\_vertices), 30

build\_indices() (in module ffc.tensor.multiindex), 23

build\_node\_counts() (in module ffc.uflacs.language.cnodes), 56

build\_node\_shapes() (in module ffc.uflacs.analysis.graph\_vertices), 30

build\_node\_sizes() (in module ffc.uflacs.analysis.graph\_symbols), 30

build\_node\_symbols() (in module ffc.uflacs.analysis.graph\_symbols), 30

build\_optimized\_tables() (in module ffc.uflacs.elementtables), 64

build\_scalar\_graph() (in module ffc.uflacs.build\_uflacs\_ir), 60

build\_scalar\_graph\_vertices() (in module ffc.uflacs.analysis.graph\_vertices), 30

build\_scalar\_node\_counts() (in module ffc.uflacs.analysis.graph\_vertices), 30

build\_uflacs\_ir() (in module ffc.uflacs.build\_uflacs\_ir), 60

build\_unique\_tables() (in module ffc.uflacs.elementtables), 64

Call (class in ffc.uflacs.language.cnodes), 48

CALL (ffc.uflacs.language.precedence.PRECEDENCE attribute), 58

Case (class in ffc.uflacs.language.cnodes), 48

cases (ffc.uflacs.language.cnodes.Switch attribute), 54

ce\_format() (ffc.uflacs.language.cnodes.ArrayAccess method), 45

ce\_format() (ffc.uflacs.language.cnodes.BinOp method), 46

ce\_format() (ffc.uflacs.language.cnodes.Call method), 48

ce\_format() (ffc.uflacs.language.cnodes.CExpr method), 47

ce\_format() (ffc.uflacs.language.cnodes.Conditional method), 48

ce\_format() (ffc.uflacs.language.cnodes.LiteralBool method), 50

ce\_format() (ffc.uflacs.language.cnodes.LiteralFloat method), 51

ce\_format() (ffc.uflacs.language.cnodes.LiteralInt method), 51

ce\_format() (ffc.uflacs.language.cnodes.LiteralString method), 51

ce\_format() (ffc.uflacs.language.cnodes.NaryOp method), 52



method), 15

CExpr (class in ffc.uflacs.language.cnodes), 47

CExprLiteral (class in ffc.uflacs.language.cnodes), 47

CExprOperator (class in ffc.uflacs.language.cnodes), 47

CExprTerminal (class in ffc.uflacs.language.cnodes), 47

check (ffc.uflacs.language.cnodes.For attribute), 49

circumradius() (ffc.quadrature.optimisedquadraturetransformerbase.QuadratureTransformerBase), 11

circumradius() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase), 13

circumradius() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase), 15

clamp\_table\_small\_numbers() (in module ffc.uflacs.elementtables), 64

classname() (ffc.uflacs.backends.ufc.generator.ufc\_generator method), 43

CNode (class in ffc.uflacs.language.cnodes), 47

codestring (ffc.uflacs.language.cnodes.VerbatimExpr attribute), 55

codestring (ffc.uflacs.language.cnodes.VerbatimStatement attribute), 55

codet\_nn() (in module ffc.uflacs.backends.ufc.coordinate\_mapping), 37

coefficient() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase), 15

coefficient() (ffc.tensor.monomialextraction.MonomialTransformer method), 21

coefficient() (ffc.uflacs.backends.ffc.access.FFCBackendAccess method), 33

coefficient() (ffc.uflacs.backends.ffc.definitions.FFCBackendDefinition), 35

coefficient\_dof\_access() (ffc.uflacs.backends.ffc.symbols.FFCBackendSymbols), 36

coefficient\_dof\_sum\_index() (ffc.uflacs.backends.ffc.symbols.FFCBackendSymbols method), 36

coefficient\_value() (ffc.uflacs.backends.ffc.symbols.FFCBackendSymbols method), 36

collect\_quadrature\_rules() (in module ffc.uflacs.tools), 67

Comment (class in ffc.uflacs.language.cnodes), 48

comment (ffc.uflacs.language.cnodes.Comment attribute), 48

comment (ffc.uflacs.language.cnodes.Pragma attribute), 53

commented\_code\_list() (in module ffc.uflacs.language.cnodes), 56

common\_block\_data\_t (class in ffc.uflacs.build\_uflacs\_ir), 60

compilation\_relevant\_parameters() (in module ffc.parameters), 74

compile\_element() (in module ffc.compiler), 69

compile\_form() (in module ffc.compiler), 69

compile\_ufl\_data() (in module ffc.main), 73

compile\_with\_error\_control() (in module ffc.errorcontrol.errorcontrol), 8

component() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase), 15

component\_tensor() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase), 15

compute\_monomial\_extraction() (ffc.tensor.monomialextraction.MonomialTransformer method), 21

compute\_reconstruct\_scalar\_sum() (ffc.uflacs.analysis.graph\_rebuild.ReconstructScalarSum method), 28

compute\_value\_numbering() (ffc.uflacs.analysis.valuenumbering.ValueNumberer method), 32

compound\_tensor\_operator() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase), 15

compute\_all\_shapes() (in module ffc.uflacs.analysis.expr\_shapes), 26

compute\_argument\_factorization() (in module ffc.uflacs.analysis.factorization), 27

compute\_cache\_scores() (in module ffc.uflacs.analysis.graph\_ssa), 29

compute\_dependencies() (in module ffc.uflacs.analysis.dependencies), 26

compute\_dependency\_count() (in module ffc.uflacs.analysis.graph\_ssa), 29

compute\_geometry() (ffc.uflacs.backends.ufc.coordinate\_mapping.ufc\_coordinate\_mapping), 38

compute\_index\_shape() (in module ffc.uflacs.analysis.expr\_shapes), 26

compute\_integral\_ir() (in module ffc.quadrature.quadraturerepresentation), 13

compute\_integral\_ir() (in module ffc.tensor.tensorrepresentation), 24

compute\_integral\_ir() (in module ffc.uflacs.uflacsrepresentation), 67

compute\_ir() (in module ffc.representation), 75

compute\_kronecker\_determinants() (ffc.uflacs.backends.ufc.coordinate\_mapping.ufc\_coordinate\_mapping), 38

compute\_jacobian\_inverses() (ffc.uflacs.backends.ufc.coordinate\_mapping.ufc\_coordinate\_mapping), 38

compute\_jacobians() (ffc.uflacs.backends.ufc.coordinate\_mapping.ufc\_coordinate\_mapping), 38

compute\_jit\_parameters\_signature() (in module ffc.parameters), 74

compute\_jit\_prefix() (in module ffc.jitcompiler), 72

compute\_permutations() (in module ffc.utils), 77

compute\_physical\_coordinates() (ffc.uflacs.backends.ufc.coordinate\_mapping.ufc\_coordinate\_mapping), 38

compute\_quadrature\_rules() (in module ffc.uflacs.tools), 67

`compute_reference_coordinates()` (method), 13  
`(ffc.uflacs.backends.ufc.coordinate_mapping.ufc_coordinate_mapping)` (ffc.uflacs.backends.ufc.form.ufc\_form method), 38  
`compute_tensor_representation()` (in module `ffc.extras`), 71  
`cond` (ffc.quadrature.symbol.Symbol attribute), 19  
`condition` (ffc.uflacs.language.cnodes.Conditional attribute), 48  
`condition` (ffc.uflacs.language.cnodes.Do attribute), 49  
`condition` (ffc.uflacs.language.cnodes.ElseIf attribute), 49  
`condition` (ffc.uflacs.language.cnodes.If attribute), 50  
`condition` (ffc.uflacs.language.cnodes.While attribute), 55  
`condition()` (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 15  
`condition()` (ffc.uflacs.analysis.graph\_rebuild.ReconstructScalarSubexpressions method), 28  
`Conditional` (class in ffc.uflacs.language.cnodes), 48  
`CONDITIONAL` (ffc.uflacs.language.precedence.PRECEDENCE attribute), 58  
`conditional()` (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerBase method), 11  
`conditional()` (ffc.quadrature.quadraturetransformer.QuadratureTransformerBase method), 13  
`conditional()` (ffc.uflacs.analysis.graph\_rebuild.ReconstructScalarSubexpressions method), 28  
`conditional()` (ffc.uflacs.language.ufl\_to\_cnodes.UFL2CNodesMixin method), 59  
`constant_value()` (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 15  
`constant_value()` (ffc.uflacs.backends.ffc.definitions.FFCBackendDefinitions method), 35  
`constructor()` (ffc.uflacs.backends.ufc.generator.ufc\_generator method), 43  
`constructor_arguments()` (ffc.uflacs.backends.ufc.generator.ufc\_generator method), 43  
`contains_zeros()` (in module `ffc.quadrature.quadratureutils`), 17  
`Continue` (class in ffc.uflacs.language.cnodes), 48  
`cos()` (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 15  
`cos()` (ffc.uflacs.language.ufl\_to\_cnodes.UFL2CNodesMixin method), 59  
`cosh()` (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 15  
`cosh()` (ffc.uflacs.language.ufl\_to\_cnodes.UFL2CNodesMixin method), 59  
`count_nodes_with_unique_post_traversal()` (in module `ffc.uflacs.analysis.graph_vertices`), 30  
`count_ops()` (in module `ffc.cpp`), 69  
`create()` (ffc.uflacs.backends.ufc.generator.ufc\_generator method), 43  
`create_argument()` (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerBase method), 11  
`create_argument()` (ffc.quadrature.quadraturetransformer.QuadratureTransformerBase method), 13  
`create_coordinate_dofmap()` (ffc.uflacs.backends.ufc.coordinate\_mapping.ufc\_coordinate\_mapping method), 40  
`create_coordinate_dofmap()` (ffc.uflacs.backends.ufc.form.ufc\_form method), 40  
`create_coordinate_finite_element()` (ffc.uflacs.backends.ufc.coordinate\_mapping.ufc\_coordinate\_mapping method), 38  
`create_coordinate_finite_element()` (ffc.uflacs.backends.ufc.form.ufc\_form method), 40  
`create_custom_integral()` (ffc.uflacs.backends.ufc.form.ufc\_form method), 40  
`create_default_custom_integral()` (ffc.uflacs.backends.ufc.form.ufc\_form method), 40  
`create_default_exterior_facet_integral()` (ffc.uflacs.backends.ufc.form.ufc\_form method), 40  
`create_default_interface_integral()` (ffc.uflacs.backends.ufc.form.ufc\_form method), 40  
`create_default_overlap_integral()` (ffc.uflacs.backends.ufc.form.ufc\_form method), 41  
`create_default_vertex_integral()` (ffc.uflacs.backends.ufc.form.ufc\_form method), 41  
`create_dofmap()` (ffc.uflacs.backends.ufc.form.ufc\_form method), 41  
`create_element()` (in module `ffc.fiatinterface`), 71  
`create_exterior_facet_integral()` (ffc.uflacs.backends.ufc.form.ufc\_form method), 41  
`create_finite_element()` (ffc.uflacs.backends.ufc.form.ufc\_form method), 41

method), 41  
 create\_float() (in module ffc.quadrature.symbolics), 20  
 create\_fraction() (in module ffc.quadrature.symbolics), 20  
 create\_function() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt method), 11  
 create\_function() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 13  
 create\_interface\_integral() (ffc.uflacs.backends.ufc.form.ufc\_form method), 41  
 create\_interior\_facet\_integral() (ffc.uflacs.backends.ufc.form.ufc\_form method), 41  
 create\_multiindex() (in module ffc.tensor.multiindex), 23  
 create\_overlap\_integral() (ffc.uflacs.backends.ufc.form.ufc\_form method), 41  
 create\_permutations() (in module ffc.quadrature.quadratureutils), 17  
 create\_product() (in module ffc.quadrature.symbolics), 20  
 create\_psi\_tables() (in module ffc.quadrature.quadratureutils), 17  
 create\_quadrature() (in module ffc.fiatinterface), 71  
 create\_quadrature\_points\_and\_weights() (in module ffc.representationutils), 76  
 create\_sub\_dofmap() (ffc.uflacs.backends.ufc.dofmap.ufc\_dofmap method), 38  
 create\_sub\_element() (ffc.uflacs.backends.ufc.finite\_element.ufc\_finite\_element method), 39  
 create\_sum() (in module ffc.quadrature.symbolics), 20  
 create\_symbol() (in module ffc.quadrature.symbolics), 20  
 create\_vertex\_integral() (ffc.uflacs.backends.ufc.form.ufc\_form method), 41  
 cross\_expr() (in module ffc.uflacs.backends.ufc.coordinate\_mapping), 37  
 CRSArray (class in ffc.uflacs.analysis.crsarray), 25  
 cs\_format() (ffc.uflacs.language.cnodes.ArrayDecl method), 45  
 cs\_format() (ffc.uflacs.language.cnodes.Break method), 47  
 cs\_format() (ffc.uflacs.language.cnodes.Case method), 48  
 cs\_format() (ffc.uflacs.language.cnodes.Comment method), 48  
 cs\_format() (ffc.uflacs.language.cnodes.Continue method), 48  
 cs\_format() (ffc.uflacs.language.cnodes.CStatement method), 48  
 cs\_format() (ffc.uflacs.language.cnodes.Default method), 48  
 cs\_format() (ffc.uflacs.language.cnodes.Do method), 49  
 cs\_format() (ffc.uflacs.language.cnodes.Else method), 49  
 cs\_format() (ffc.uflacs.language.cnodes.ElseIf method), 49  
 cs\_format() (ffc.uflacs.language.cnodes.For method), 49  
 cs\_format() (ffc.uflacs.language.cnodes.ForRange method), 50  
 cs\_format() (ffc.uflacs.language.cnodes.If method), 50  
 cs\_format() (ffc.uflacs.language.cnodes.Namespace method), 51  
 cs\_format() (ffc.uflacs.language.cnodes.Pragma method), 53  
 cs\_format() (ffc.uflacs.language.cnodes.Return method), 53  
 cs\_format() (ffc.uflacs.language.cnodes.Scope method), 53  
 cs\_format() (ffc.uflacs.language.cnodes.Statement method), 54  
 cs\_format() (ffc.uflacs.language.cnodes.StatementList method), 54  
 cs\_format() (ffc.uflacs.language.cnodes.Switch method), 54  
 cs\_format() (ffc.uflacs.language.cnodes.Throw method), 54  
 cs\_format() (ffc.uflacs.language.cnodes.Using method), 55  
 cs\_format() (ffc.uflacs.language.cnodes.VariableDecl method), 55  
 cs\_format() (ffc.uflacs.language.cnodes.VerbatimStatement method), 55  
 cs\_format() (ffc.uflacs.language.cnodes.While method), 55  
 CStatement (class in ffc.uflacs.language.cnodes), 48  
 custom\_points\_table() (ffc.uflacs.backends.ffc.symbols.FFCBackendSymbols method), 36  
 custom\_quadrature\_points() (ffc.uflacs.backends.ffc.symbols.FFCBackendSymbols method), 36  
 custom\_quadrature\_weights() (ffc.uflacs.backends.ffc.symbols.FFCBackendSymbols method), 36  
 custom\_weights\_table() (ffc.uflacs.backends.ffc.symbols.FFCBackendSymbols method), 36

## D

debug (ffc.uflacs.language.cnodes.CNode attribute), 48  
 debug() (in module ffc.log), 73  
 debug\_code() (in module ffc.log), 73  
 debug\_dict() (in module ffc.log), 73  
 debug\_ir() (in module ffc.log), 73  
 Default (class in ffc.uflacs.language.cnodes), 48  
 default (ffc.uflacs.language.cnodes.Switch attribute), 54  
 default\_cache\_score\_policy() (in module ffc.uflacs.analysis.graph\_ssa), 29  
 default\_jit\_parameters() (in module ffc.parameters), 74



[default\\_optimize\\_parameters\(\)](#) (in module [ffc.quadrature.parameters](#)), 12  
[default\\_parameters\(\)](#) (in module [ffc.parameters](#)), 74  
[default\\_parameters\(\)](#) (in module [ffc.uflacs.params](#)), 67  
[default\\_partition\\_seed\(\)](#) (in module [ffc.uflacs.analysis.graph\\_ssa](#)), 29  
[degree\(\)](#) ([ffc.enrichedelement.EnrichedElement](#) method), 70  
[degree\(\)](#) ([ffc.enrichedelement.SpaceOfReals](#) method), 70  
[degree\(\)](#) ([ffc.restrictedelement.RestrictedElement](#) method), 76  
[degree\(\)](#) ([ffc.uflacs.backends.ufc.finite\\_element.ufc\\_finite\\_element](#) method), 39  
[denom](#) ([ffc.quadrature.fraction.Fraction](#) attribute), 10  
[deprecate\(\)](#) (in module [ffc.log](#)), 73  
[Dereference](#) (class in [ffc.uflacs.language.cnodes](#)), 48  
[DEREFERENCE](#) ([ffc.uflacs.language.precedence.PRECEDENCE](#) attribute), 58  
[derivative\(\)](#) ([ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase](#) method), 15  
[derivatives\(\)](#) ([ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase](#) method), 15  
[destructor\(\)](#) ([ffc.uflacs.backends.ufc.generator.ufc\\_generator](#) method), 43  
[det\\_22\(\)](#) (in module [ffc.uflacs.backends.ufc.coordinate\\_mapping](#)), 37  
[det\\_nn\(\)](#) (in module [ffc.uflacs.backends.ufc.coordinate\\_mapping](#)), 37  
[disp\(\)](#) ([ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase](#) method), 15  
[Div](#) (class in [ffc.uflacs.language.cnodes](#)), 49  
[DIV](#) ([ffc.uflacs.language.precedence.PRECEDENCE](#) attribute), 58  
[division\(\)](#) ([ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt](#) method), 11  
[division\(\)](#) ([ffc.quadrature.quadraturetransformer.QuadratureTransformer](#) method), 13  
[division\(\)](#) ([ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase](#) method), 15  
[division\(\)](#) ([ffc.tensor.monomialextraction.MonomialTransformer](#) method), 21  
[division\(\)](#) ([ffc.uflacs.analysis.graph\\_rebuild.ReconstructScalarSubexpressions](#) method), 28  
[division\(\)](#) ([ffc.uflacs.language.ufl\\_to\\_cnodes.UFL2CNodesMixin](#) method), 59  
[dmats\(\)](#) ([ffc.enrichedelement.SpaceOfReals](#) method), 70  
[dmats\(\)](#) ([ffc.restrictedelement.RestrictedElement](#) method), 76  
[Do](#) (class in [ffc.uflacs.language.cnodes](#)), 49  
[domain\\_dof\\_access\(\)](#) ([ffc.uflacs.backends.ffc.symbols.FFCBackendSymbols](#) method), 36  
[domain\\_dofs\\_access\(\)](#) ([ffc.uflacs.backends.ffc.symbols.FFCBackendSymbols](#) method), 36  
[dual\\_basis\(\)](#) ([ffc.enrichedelement.EnrichedElement](#) method), 70  
[dual\\_basis\(\)](#) ([ffc.enrichedelement.SpaceOfReals](#) method), 70  
[dual\\_basis\(\)](#) ([ffc.mixedelement.MixedElement](#) method), 74  
[dual\\_basis\(\)](#) ([ffc.quadratureelement.QuadratureElement](#) method), 75  
[dual\\_basis\(\)](#) ([ffc.restrictedelement.RestrictedElement](#) method), 77  
[dual\\_forms\(\)](#) ([ffc.errorcontrol.errorcontrolgenerators.ErrorControlGenerators](#) method), 8  

## E

[element\(\)](#) ([ffc.tensor.monomialextraction.MonomialFactor](#) method), 21  
[element\\_table\(\)](#) ([ffc.uflacs.backends.ffc.symbols.FFCBackendSymbols](#) method), 36  
[element\\_tensor\(\)](#) ([ffc.uflacs.backends.ffc.symbols.FFCBackendSymbols](#) method), 36  
[elements\(\)](#) ([ffc.enrichedelement.EnrichedElement](#) method), 74  
[elements\(\)](#) ([ffc.mixedelement.MixedElement](#) method), 74  
[Else](#) (class in [ffc.uflacs.language.cnodes](#)), 49  
[ElseIf](#) (class in [ffc.uflacs.language.cnodes](#)), 49  
[empty\\_expr\\_ir\(\)](#) (in module [ffc.uflacs.build\\_uflacs\\_ir](#)), 61  
[enabled\\_coefficients\(\)](#) ([ffc.uflacs.backends.ufc.integrals.ufc\\_integral](#) method), 44  
[end](#) ([ffc.uflacs.language.cnodes.ForRange](#) attribute), 50  
[end\(\)](#) (in module [ffc.log](#)), 73  
[EnrichedElement](#) (class in [ffc.enrichedelement](#)), 70  
[entity\(\)](#) ([ffc.uflacs.backends.ffc.symbols.FFCBackendSymbols](#) method), 37  
[entity\\_dofs\(\)](#) ([ffc.enrichedelement.EnrichedElement](#) method), 70  
[entity\\_dofs\(\)](#) ([ffc.enrichedelement.SpaceOfReals](#) method), 70  
[entity\\_dofs\(\)](#) ([ffc.mixedelement.MixedElement](#) method), 74  
[entity\\_dofs\(\)](#) ([ffc.quadratureelement.QuadratureElement](#) method), 75  
[entity\\_dofs\(\)](#) ([ffc.restrictedelement.RestrictedElement](#) method), 77  
[entity\\_type\\_from\\_integral\\_type\(\)](#) (in module [ffc.representationutils](#)), 76  
[EQ](#) (class in [ffc.uflacs.language.cnodes](#)), 49  
[EQ](#) ([ffc.uflacs.language.precedence.PRECEDENCE](#) attribute), 58  
[eq\(\)](#) ([ffc.uflacs.language.ufl\\_to\\_cnodes.UFL2CNodesMixin](#) method), 59  
[eq\(\)](#) (in module [ffc.uflacs.elementtables](#)), 64  
[erf\(\)](#) ([ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase](#) method), 15  
[erf\(\)](#) ([ffc.uflacs.language.ufl\\_to\\_cnodes.UFL2CNodesMixin](#) method), 59

`error()` (in module `ffc.log`), 73  
`error_estimate()` (`ffc.errorcontrol.errorcontrolgenerators.ErrorControlGenerator` method), 8  
`error_indicators()` (`ffc.errorcontrol.errorcontrolgenerators.ErrorControlGenerator` method), 8  
`ErrorControlGenerator` (class in `ffc.errorcontrol.errorcontrolgenerators`), 8  
`estimate_cost()` (in module `ffc.tensor.costestimation`), 20  
`evaluate_basis()` (`ffc.uflacs.backends.ufc.finite_element.ufc_finite_element` method), 39  
`evaluate_basis_all()` (`ffc.uflacs.backends.ufc.finite_element.ufc_finite_element` method), 39  
`evaluate_basis_derivatives()` (`ffc.uflacs.backends.ufc.finite_element.ufc_finite_element` method), 39  
`evaluate_basis_derivatives_all()` (`ffc.uflacs.backends.ufc.finite_element.ufc_finite_element` method), 39  
`evaluate_dof()` (`ffc.uflacs.backends.ufc.finite_element.ufc_finite_element` method), 39  
`evaluate_dof_and_dofs()` (in module `ffc.evaluatedof`), 71  
`evaluate_dofs()` (`ffc.uflacs.backends.ufc.finite_element.ufc_finite_element` method), 39  
`evaluate_reference_basis()` (`ffc.uflacs.backends.ufc.finite_element.ufc_finite_element` method), 39  
`evaluate_reference_basis_derivatives()` (`ffc.uflacs.backends.ufc.finite_element.ufc_finite_element` method), 39  
exception (`ffc.uflacs.language.cnodes.Throw` attribute), 54  
`exp` (`ffc.quadrature.symbol.Symbol` attribute), 19  
`exp()` (`ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase` method), 15  
`exp()` (`ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin` method), 59  
`expand()` (`ffc.quadrature.expr.Expr` method), 9  
`expand()` (`ffc.quadrature.fraction.Fraction` method), 10  
`expand()` (`ffc.quadrature.product.Product` method), 12  
`expand()` (`ffc.quadrature.sumobj.Sum` method), 19  
`expand_operations()` (in module `ffc.quadrature.reduce_operations`), 17  
`Expr` (class in `ffc.quadrature.expr`), 9  
`expr` (`ffc.uflacs.language.cnodes.Statement` attribute), 54  
`expr()` (`ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase` method), 15  
`expr()` (`ffc.tensor.monomialextraction.MonomialTransformer` method), 22  
`expr()` (`ffc.uflacs.analysis.balancing.BalanceModifiers` method), 25  
`expr()` (`ffc.uflacs.analysis.graph_rebuild.ReconstructScalarSubexpressions` method), 28  
`expr()` (`ffc.uflacs.analysis.valuenumbering.ValueNumberer` method), 32  
`expr()` (`ffc.uflacs.backends.ffc.access.FFCBackendAccess` method), 33  
`expr()` (`ffc.uflacs.backends.ffc.definitions.FFCBackendDefinitions` method), 35  
`expr()` (`ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin` method), 59  
`expr_list()` (`ffc.uflacs.analysis.graph_rebuild.ReconstructScalarSubexpressions` method), 28  
`extract_coefficient_spaces()` (in module `ffc.backends.dolfin.functionspace`), 5  
`extract_keywords()` (in module `ffc.uflacs.backends.ufc.templates`), 44  
`extract_monomial_form()` (in module `ffc.tensor.monomialextraction`), 22  
`extract_monomial_integrand()` (in module `ffc.tensor.monomialextraction`), 22  
`extract_unique_indices()` (`ffc.tensor.monomialtransformation.TransformedMonomialIndex` method), 23

## F

`facet_area()` (`ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformer` method), 11  
`facet_area()` (`ffc.quadrature.quadraturetransformer.QuadratureTransformer` method), 13  
`facet_area()` (`ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase` method), 15  
`facet_avg()` (`ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase` method), 15  
`facet_avg()` (`ffc.uflacs.analysis.balancing.BalanceModifiers` method), 25  
`facet_avg()` (`ffc.uflacs.analysis.valuenumbering.ValueNumberer` method), 32  
`facet_coordinate()` (`ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformer` method), 11  
`facet_coordinate()` (`ffc.quadrature.quadraturetransformer.QuadratureTransformer` method), 13  
`facet_coordinate()` (`ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase` method), 15  
`facet_coordinate()` (`ffc.uflacs.backends.ffc.access.FFCBackendAccess` method), 33  
`facet_edge_vectors()` (`ffc.uflacs.backends.ffc.access.FFCBackendAccess` method), 33  
`facet_edge_vectors()` (`ffc.uflacs.backends.ffc.definitions.FFCBackendDefinitions` method), 35  
`facet_jacobian()` (`ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformer` method), 11  
`facet_jacobian()` (`ffc.quadrature.quadraturetransformer.QuadratureTransformer` method), 13  
`facet_jacobian()` (`ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase` method), 15

[facet\\_jacobian\(\) \(ffc.uflacs.backends.ffc.access.FFCBackendAccess attribute\), 61](#)  
[method\), 33](#)  
[facet\\_jacobian\(\) \(ffc.uflacs.backends.ffc.definitions.FFCBackendDefinitions attribute\), 62](#)  
[method\), 35](#)  
[facet\\_jacobian\\_determinant\(\) \(ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt attribute\), 62](#)  
[method\), 11](#)  
[facet\\_jacobian\\_determinant\(\) \(ffc.quadrature.quadraturetransformer.QuadratureTransformer attribute\), 61](#)  
[method\), 14](#)  
[facet\\_jacobian\\_determinant\(\) \(ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase attribute\), 61](#)  
[method\), 16](#)  
[facet\\_jacobian\\_determinant\(\) \(ffc.uflacs.backends.ffc.access.FFCBackendAccess attribute\), 63](#)  
[method\), 33](#)  
[facet\\_jacobian\\_determinant\(\) \(ffc.uflacs.backends.ffc.definitions.FFCBackendDefinitions attribute\), 63](#)  
[method\), 35](#)  
[facet\\_jacobian\\_inverse\(\) \(ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt attribute\), 63](#)  
[method\), 11](#)  
[facet\\_jacobian\\_inverse\(\) \(ffc.quadrature.quadraturetransformer.QuadratureTransformer attribute\), 63](#)  
[method\), 14](#)  
[facet\\_jacobian\\_inverse\(\) \(ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase attribute\), 63](#)  
[method\), 16](#)  
[facet\\_jacobian\\_inverse\(\) \(ffc.uflacs.backends.ffc.access.FFCBackendAccess attribute\), 63](#)  
[method\), 33](#)  
[facet\\_jacobian\\_inverse\(\) \(ffc.uflacs.backends.ffc.definitions.FFCBackendDefinitions attribute\), 63](#)  
[method\), 35](#)  
[facet\\_normal\(\) \(ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt attribute\), 63](#)  
[method\), 11](#)  
[facet\\_normal\(\) \(ffc.quadrature.quadraturetransformer.QuadratureTransformer attribute\), 63](#)  
[method\), 14](#)  
[facet\\_normal\(\) \(ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase attribute\), 63](#)  
[method\), 16](#)  
[facet\\_normal\(\) \(ffc.uflacs.backends.ffc.access.FFCBackendAccess attribute\), 63](#)  
[method\), 33](#)  
[facet\\_normal\(\) \(ffc.uflacs.backends.ffc.definitions.FFCBackendDefinitions attribute\), 63](#)  
[method\), 35](#)  
[facet\\_orientation\(\) \(ffc.uflacs.backends.ffc.access.FFCBackendAccess attribute\), 63](#)  
[method\), 33](#)  
[facet\\_orientation\(\) \(ffc.uflacs.backends.ffc.definitions.FFCBackendDefinitions attribute\), 63](#)  
[method\), 35](#)  
[facet\\_origin\(\) \(ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt attribute\), 63](#)  
[method\), 11](#)  
[facet\\_origin\(\) \(ffc.quadrature.quadraturetransformer.QuadratureTransformer attribute\), 63](#)  
[method\), 14](#)  
[facet\\_origin\(\) \(ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase attribute\), 63](#)  
[method\), 16](#)  
[facet\\_residual\(\) \(ffc.errorcontrol.errorcontrolgenerators.ErrorControlGenerators attribute\), 71](#)  
[method\), 8](#)  
[factor\\_index \(ffc.uflacs.build\\_uflacs\\_ir.common\\_block\\_data\\_t attribute\), 61](#)  
[factor\\_index \(ffc.uflacs.build\\_uflacs\\_ir.full\\_block\\_data\\_t attribute\), 61](#)  
[factor\\_index \(ffc.uflacs.build\\_uflacs\\_ir.preintegrated\\_block\\_data\\_t attribute\), 61](#)  
[factor\\_index \(ffc.uflacs.build\\_uflacs\\_ir.premultiplied\\_block\\_data\\_t attribute\), 61](#)  
[factor\\_is\\_piecewise \(ffc.uflacs.build\\_uflacs\\_ir.common\\_block\\_data\\_t attribute\), 61](#)  
[factor\\_is\\_piecewise \(ffc.uflacs.build\\_uflacs\\_ir.full\\_block\\_data\\_t attribute\), 61](#)  
[factor\\_is\\_piecewise \(ffc.uflacs.build\\_uflacs\\_ir.partial\\_block\\_data\\_t attribute\), 61](#)  
[factor\\_is\\_piecewise \(ffc.uflacs.build\\_uflacs\\_ir.premultiplied\\_block\\_data\\_t attribute\), 61](#)  
[factorization \(class in ffc.uflacs.analysis.factorization\), 26](#)  
[false \(ffc.uflacs.language.cnodes.Conditional attribute\), 63](#)  
[family\(\) \(ffc.uflacs.backends.ufc.finite\\_element.ufc\\_finite\\_element attribute\), 63](#)  
[ffc \(module\), 77](#)  
[ffc.backends \(module\), 7](#)  
[ffc.backends.dofin \(module\), 6](#)  
[ffc.backends.dofin.capsules \(module\), 5](#)  
[ffc.backends.dofin.functionspace \(module\), 5](#)  
[ffc.backends.dofin.includes \(module\), 6](#)  
[ffc.backends.dofin.wrappers \(module\), 6](#)  
[ffc.backends.ufc \(module\), 7](#)  
[ffc.backends.ufc.mapping \(module\), 6](#)  
[ffc.backends.ufc.dofmap \(module\), 6](#)  
[ffc.backends.ufc.finite\\_element \(module\), 6](#)  
[ffc.backends.ufc.form \(module\), 6](#)  
[ffc.backends.ufc.function \(module\), 6](#)  
[ffc.backends.ufc.integrals \(module\), 7](#)  
[ffc.codesnippets \(module\), 68](#)  
[ffc.compilers \(module\), 68](#)  
[ffc.cpp \(module\), 69](#)  
[ffc.errorcontrol \(module\), 9](#)  
[ffc.errorcontrol.errorcontrolgenerators \(module\), 8](#)  
[ffc.evaluatebasisderivatives \(module\), 71](#)  
[ffc.evaluatebasisderivatives \(module\), 71](#)  
[ffc.extras \(module\), 71](#)  
[ffc.fiatinterface \(module\), 71](#)  
[ffc.formatting \(module\), 72](#)  
[ffc.interpolatevertexvalues \(module\), 72](#)



- ffc.jitcompiler (module), 72
- ffc.log (module), 72
- ffc.main (module), 73
- ffc.mixedelement (module), 74
- ffc.optimization (module), 74
- ffc.parameters (module), 74
- ffc.plot (module), 75
- ffc.quadrature (module), 20
- ffc.quadrature.expr (module), 9
- ffc.quadrature.floatvalue (module), 10
- ffc.quadrature.fraction (module), 10
- ffc.quadrature.optimisedquadraturetransformer (module), 10
- ffc.quadrature.parameters (module), 12
- ffc.quadrature.product (module), 12
- ffc.quadrature.quadraturegenerator (module), 12
- ffc.quadrature.quadratureoptimization (module), 12
- ffc.quadrature.quadraturerepresentation (module), 13
- ffc.quadrature.quadraturetransformer (module), 13
- ffc.quadrature.quadraturetransformerbase (module), 14
- ffc.quadrature.quadratureutils (module), 17
- ffc.quadrature.reduce\_operations (module), 17
- ffc.quadrature.sumobj (module), 19
- ffc.quadrature.symbol (module), 19
- ffc.quadrature.symbolics (module), 20
- ffc.quadrature.tabulate\_basis (module), 20
- ffc.quadratureelement (module), 75
- ffc.representation (module), 75
- ffc.representationutils (module), 76
- ffc.restrictedelement (module), 76
- ffc.tensor (module), 24
- ffc.tensor.costestimation (module), 20
- ffc.tensor.geometrytensor (module), 20
- ffc.tensor.monomialextraction (module), 21
- ffc.tensor.monomialintegration (module), 22
- ffc.tensor.monomialtransformation (module), 22
- ffc.tensor.multiindex (module), 23
- ffc.tensor.referencetensor (module), 24
- ffc.tensor.tensorgenerator (module), 24
- ffc.tensor.tensoroptimization (module), 24
- ffc.tensor.tensorreordering (module), 24
- ffc.tensor.tensorrepresentation (module), 24
- ffc.uflacs (module), 67
- ffc.uflacs.analysis (module), 33
- ffc.uflacs.analysis.balancing (module), 25
- ffc.uflacs.analysis.crsarray (module), 25
- ffc.uflacs.analysis.dependencies (module), 26
- ffc.uflacs.analysis.expr\_shapes (module), 26
- ffc.uflacs.analysis.factorization (module), 26
- ffc.uflacs.analysis.graph (module), 27
- ffc.uflacs.analysis.graph\_rebuild (module), 28
- ffc.uflacs.analysis.graph\_ssa (module), 29
- ffc.uflacs.analysis.graph\_symbols (module), 30
- ffc.uflacs.analysis.graph\_vertices (module), 30
- ffc.uflacs.analysis.indexing (module), 30
- ffc.uflacs.analysis.modified\_terminals (module), 31
- ffc.uflacs.analysis.valuenumbering (module), 31
- ffc.uflacs.backends (module), 44
- ffc.uflacs.backends.ffc (module), 37
- ffc.uflacs.backends.ffc.access (module), 33
- ffc.uflacs.backends.ffc.backend (module), 34
- ffc.uflacs.backends.ffc.common (module), 34
- ffc.uflacs.backends.ffc.definitions (module), 34
- ffc.uflacs.backends.ffc.symbols (module), 36
- ffc.uflacs.backends.ufc (module), 44
- ffc.uflacs.backends.ufc.coordinate\_mapping (module), 37
- ffc.uflacs.backends.ufc.dofmap (module), 38
- ffc.uflacs.backends.ufc.evaluatebasis (module), 39
- ffc.uflacs.backends.ufc.finite\_element (module), 39
- ffc.uflacs.backends.ufc.form (module), 40
- ffc.uflacs.backends.ufc.generator (module), 43
- ffc.uflacs.backends.ufc.generators (module), 43
- ffc.uflacs.backends.ufc.integrals (module), 43
- ffc.uflacs.backends.ufc.templates (module), 44
- ffc.uflacs.backends.ufc.utils (module), 44
- ffc.uflacs.build\_uflacs\_ir (module), 60
- ffc.uflacs.elementtables (module), 64
- ffc.uflacs.integralgenerator (module), 65
- ffc.uflacs.language (module), 60
- ffc.uflacs.language.cnodes (module), 45
- ffc.uflacs.language.format\_lines (module), 57
- ffc.uflacs.language.format\_value (module), 57
- ffc.uflacs.language.precedence (module), 57
- ffc.uflacs.language.ufl\_to\_cnodes (module), 58
- ffc.uflacs.params (module), 67
- ffc.uflacs.tools (module), 67
- ffc.uflacs.uflacsgenerator (module), 67
- ffc.uflacs.uflacsoptimization (module), 67
- ffc.uflacs.uflacsrepresentation (module), 67
- ffc.utils (module), 77
- ffc.wrappers (module), 77
- ffc\_assert() (in module ffc.log), 73
- FFCBackend (class in ffc.uflacs.backends.ffc.backend), 34
- FFCBackendAccess (class in ffc.uflacs.backends.ffc.access), 33
- FFCBackendDefinitions (class in ffc.uflacs.backends.ffc.definitions), 34
- FFCBackendSymbols (class in ffc.uflacs.backends.ffc.symbols), 36
- FFCError, 72
- FFCJitError, 72
- firstkey() (in module ffc.quadrature.optimisedquadraturetransformer), 11
- firstkey() (in module ffc.quadrature.quadraturetransformer), 14
- FIXED (ffc.tensor.monomialtransformation.MonomialIndex attribute), 23

[flatten\\_psi\\_tables\(\)](#) (in module [ffc.quadrature.quadratureutils](#)), 17  
[flattened\\_indices\(\)](#) (in module [ffc.uflacs.language.cnodes](#)), 56  
[FlattenedArray](#) (class in [ffc.uflacs.language.cnodes](#)), 49  
[float\\_product\(\)](#) (in module [ffc.uflacs.language.cnodes](#)), 56  
[float\\_value\(\)](#) ([ffc.uflacs.backends.ffc.access.FFCBackendAccess](#) method), 33  
[float\\_value\(\)](#) ([ffc.uflacs.language.ufl\\_to\\_cnodes.UFL2CNodesMixin](#) method), 59  
[FloatValue](#) (class in [ffc.quadrature.floatvalue](#)), 10  
[For](#) (class in [ffc.uflacs.language.cnodes](#)), 49  
[form\\_argument\(\)](#) ([ffc.uflacs.analysis.valuenumbering.ValueNumberer](#) method), 32  
[format\\_code\(\)](#) (in module [ffc.formatting](#)), 72  
[format\\_float\(\)](#) (in module [ffc.uflacs.language.format\\_value](#)), 57  
[format\\_indented\\_lines\(\)](#) (in module [ffc.uflacs.language.format\\_lines](#)), 57  
[format\\_int\(\)](#) (in module [ffc.uflacs.language.format\\_value](#)), 57  
[format\\_mt\\_name\(\)](#) (in module [ffc.uflacs.backends.ffc.symbols](#)), 37  
[format\\_value\(\)](#) (in module [ffc.uflacs.language.format\\_value](#)), 57  
[ForRange](#) (class in [ffc.uflacs.language.cnodes](#)), 50  
[Fraction](#) (class in [ffc.quadrature.fraction](#)), 10  
[from\\_rows\(\)](#) ([ffc.uflacs.analysis.crsarray.CRSArray](#) class method), 25  
[full\\_block\\_data\\_t](#) (class in [ffc.uflacs.build\\_uflacs\\_ir](#)), 61  
[function](#) ([ffc.uflacs.language.cnodes.Call](#) attribute), 48

## G

[GE](#) (class in [ffc.uflacs.language.cnodes](#)), 50  
[GE](#) ([ffc.uflacs.language.precedence.PRECEDENCE](#) attribute), 58  
[ge\(\)](#) ([ffc.uflacs.language.ufl\\_to\\_cnodes.UFL2CNodesMixin](#) method), 59  
[generate\(\)](#) ([ffc.uflacs.backends.ufc.generator.ufc\\_generator](#) method), 43  
[generate\(\)](#) ([ffc.uflacs.integralgenerator.IntegralGenerator](#) method), 65  
[generate\\_accumulation\\_loop\(\)](#) (in module [ffc.uflacs.backends.ufc.coordinate\\_mapping](#)), 37  
[generate\\_all\\_error\\_control\\_forms\(\)](#) ([ffc.errorcontrol.errorcontrolgenerators.ErrorControlGenerators](#) method), 8  
[generate\\_assign\\_inverse\(\)](#) (in module [ffc.uflacs.backends.ufc.coordinate\\_mapping](#)), 37  
[generate\\_aux\\_constants\(\)](#) (in module [ffc.quadrature.symbolics](#)), 20  
[generate\\_block\\_parts\(\)](#) ([ffc.uflacs.integralgenerator.IntegralGenerator](#) method), 65  
[generate\\_code\(\)](#) (in module [ffc.codegeneration](#)), 68  
[generate\\_compute\\_ATA\(\)](#) (in module [ffc.uflacs.backends.ufc.coordinate\\_mapping](#)), 37  
[generate\\_copyout\\_statements\(\)](#) ([ffc.uflacs.integralgenerator.IntegralGenerator](#) method), 65  
[generate\\_cross\\_decl\(\)](#) (in module [ffc.uflacs.backends.ufc.coordinate\\_mapping](#)), 38  
[generate\\_dofblock\\_partition\(\)](#) ([ffc.uflacs.integralgenerator.IntegralGenerator](#) method), 65  
[generate\\_dolfin\\_code\(\)](#) (in module [ffc.backends.dolfin.wrappers](#)), 6  
[generate\\_element\\_tables\(\)](#) ([ffc.uflacs.integralgenerator.IntegralGenerator](#) method), 65  
[generate\\_enabled\\_coefficients\(\)](#) (in module [ffc.representationutils](#)), 76  
[generate\\_evaluate\\_reference\\_basis\(\)](#) (in module [ffc.uflacs.backends.ufc.evaluatebasis](#)), 39  
[generate\\_expr\\_copyout\\_statements\(\)](#) ([ffc.uflacs.integralgenerator.IntegralGenerator](#) method), 65  
[generate\\_factory\\_functions\(\)](#) (in module [ffc.formatting](#)), 72  
[generate\\_form\(\)](#) (in module [ffc.backends.dolfin.form](#)), 5  
[generate\\_integral\\_code\(\)](#) (in module [ffc.quadrature.quadraturegenerator](#)), 12  
[generate\\_integral\\_code\(\)](#) (in module [ffc.tensor.tensorgenerator](#)), 24  
[generate\\_integral\\_code\(\)](#) (in module [ffc.uflacs.uflacsgenerator](#)), 67  
[generate\\_jit\\_factory\\_functions\(\)](#) (in module [ffc.formatting](#)), 72  
[generate\\_partition\(\)](#) ([ffc.uflacs.integralgenerator.IntegralGenerator](#) method), 66  
[generate\\_preintegrated\\_dofblock\\_partition\(\)](#) ([ffc.uflacs.integralgenerator.IntegralGenerator](#) method), 66  
[generate\\_psi\\_table\\_name\(\)](#) (in module [ffc.uflacs.elementtables](#)), 64  
[generate\\_quadrature\\_loop\(\)](#) ([ffc.uflacs.integralgenerator.IntegralGenerator](#) method), 66  
[generate\\_quadrature\\_tables\(\)](#) ([ffc.uflacs.integralgenerator.IntegralGenerator](#) method), 66  
[generate\\_return\\_new\(\)](#) (in module [ffc.uflacs.backends.ufc.utils](#)), 44  
[generate\\_return\\_new\\_switch\(\)](#) (in module

ffc.uflacs.backends.ufc.utils), 44

generate\_runtime\_quadrature\_loop() (ffc.uflacs.integralgenerator.IntegralGenerator method), 66

generate\_snippets() (ffc.uflacs.backends.ufc.generator.ufc\_generator method), 43

generate\_tensor\_copyout\_statements() (ffc.uflacs.integralgenerator.IntegralGenerator method), 66

generate\_tensor\_reset() (ffc.uflacs.integralgenerator.IntegralGenerator method), 66

generate\_tensor\_value\_initialization() (ffc.uflacs.integralgenerator.IntegralGenerator method), 66

generate\_terms() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 16

generate\_typedefs() (in module ffc.backends.dofin.functionspace), 6

generate\_unstructured\_pieewise\_partition() (ffc.uflacs.integralgenerator.IntegralGenerator method), 66

generate\_unstructured\_varying\_partition() (ffc.uflacs.integralgenerator.IntegralGenerator method), 66

generate\_update\_ec() (in module ffc.backends.dofin.goalfunctional), 6

generate\_wrapper\_code() (in module ffc.wrappers), 77

geometric\_dimension() (ffc.uflacs.backends.ufc.coordinate\_geometry method), 38

geometric\_dimension() (ffc.uflacs.backends.ufc.finite\_element\_finite\_element method), 39

geometric\_dimension() (ffc.uflacs.backends.ufc.form.ufc\_form method), 41

geometric\_quantity() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 16

GeometryTensor (class in ffc.tensor.geometrytensor), 20

get\_arg\_factors() (ffc.uflacs.integralgenerator.IntegralGenerator method), 66

get\_coeffs() (ffc.enrichedelement.SpaceOfReals method), 70

get\_coeffs() (ffc.restrictedelement.RestrictedElement method), 77

get\_common\_block\_data() (in module ffc.uflacs.build\_uflacs\_ir), 61

get\_constants() (in module ffc.quadrature.reduce\_operations), 18

get\_entities() (ffc.uflacs.integralgenerator.IntegralGenerator method), 66

get\_ffc\_table\_values() (in module ffc.uflacs.elementtables), 64

get\_geo\_terms() (in module ffc.quadrature.reduce\_operations), 18

get\_handler() (in module ffc.log), 73

get\_include\_path() (in module ffc.backends.ufc), 7

get\_includes() (ffc.uflacs.integralgenerator.IntegralGenerator method), 66

get\_indices() (in module ffc.quadrature.reduce\_operations), 18

get\_logger() (in module ffc.log), 73

get\_modified\_terminal\_element() (in module ffc.uflacs.elementtables), 64

get\_node\_symbols() (ffc.uflacs.analysis.valuenumbering.ValueNumberer method), 32

get\_num\_members() (ffc.enrichedelement.SpaceOfReals method), 70

get\_num\_members() (ffc.restrictedelement.RestrictedElement method), 77

get\_ones() (in module ffc.quadrature.quadratureutils), 17

get\_quadrature\_transformer\_base() (in module ffc.quadrature.reduce\_operations), 18

get\_temp\_symbol() (ffc.uflacs.integralgenerator.IntegralGenerator method), 66

get\_ufc\_cxx\_flags() (in module ffc.backends.ufc), 7

get\_ufc\_signature() (in module ffc.backends.ufc), 7

get\_ufc\_templates\_signature() (in module ffc.backends.ufc), 7

get\_unique\_vars() (ffc.quadrature.expr.Expr method), 9

get\_unique\_vars() (ffc.quadrature.fraction.Fraction method), 10

get\_unique\_vars() (ffc.quadrature.product.Product method), 12

get\_uflacs\_cxx\_flags() (in module ffc.backends.ufc), 7

get\_uflacs\_templates\_signature() (in module ffc.backends.ufc), 7

get\_uflacs\_unique\_vars() (ffc.quadrature.sumobj.Sum method), 19

get\_uflacs\_unique\_vars() (ffc.quadrature.symbol.Symbol method), 19

get\_var() (ffc.uflacs.integralgenerator.IntegralGenerator method), 66

get\_var\_occurrences() (ffc.quadrature.fraction.Fraction method), 10

get\_var\_occurrences() (ffc.quadrature.product.Product method), 12

get\_var\_occurrences() (ffc.quadrature.sumobj.Sum method), 19

get\_var\_occurrences() (ffc.quadrature.symbol.Symbol method), 19

get\_variables() (in module ffc.quadrature.reduce\_operations), 18

get\_vrs() (ffc.quadrature.product.Product method), 12

global\_dimension() (ffc.uflacs.backends.ufc.dofmap.ufc\_dofmap method), 38

grad() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 16

grad() (ffc.tensor.monomialextraction.MonomialTransformer method), 22

grad() (ffc.uflacs.analysis.balancing.BalanceModifiers method), 25

grad() (ffc.uflacs.analysis.valuenumbering.ValueNumberer method), 32

Graph2 (class in ffc.uflacs.analysis.graph), 27

group\_vars() (in module ffc.quadrature.reduce\_operations), 18

GT (class in ffc.uflacs.language.cnodes), 50

GT (ffc.uflacs.language.precedence.PRECEDENCE attribute), 58

gt() (ffc.uflacs.language.ufl\_to\_cnodes.UFL2CNodesMixin method), 59

## H

handle\_conditional() (in module ffc.uflacs.analysis.factorization), 27

handle\_division() (in module ffc.uflacs.analysis.factorization), 27

handle\_operator() (in module ffc.uflacs.analysis.factorization), 27

handle\_product() (in module ffc.uflacs.analysis.factorization), 27

handle\_sum() (in module ffc.uflacs.analysis.factorization), 27

has\_cell\_integrals() (ffc.uflacs.backends.ufc.form.ufc\_form method), 41

has\_custom\_integrals() (ffc.uflacs.backends.ufc.form.ufc\_form method), 41

has\_cutcell\_integrals() (ffc.uflacs.backends.ufc.form.ufc\_form method), 41

has\_exterior\_facet\_integrals() (ffc.uflacs.backends.ufc.form.ufc\_form method), 41

has\_interface\_integrals() (ffc.uflacs.backends.ufc.form.ufc\_form method), 42

has\_interior\_facet\_integrals() (ffc.uflacs.backends.ufc.form.ufc\_form method), 42

has\_overlap\_integrals() (ffc.uflacs.backends.ufc.form.ufc\_form method), 42

has\_var() (ffc.uflacs.integralgenerator.IntegralGenerator method), 66

has\_vertex\_integrals() (ffc.uflacs.backends.ufc.form.ufc\_form method), 42

HIGHEST (ffc.uflacs.language.precedence.PRECEDENCE attribute), 58

## I

identity() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 16

If (class in ffc.uflacs.language.cnodes), 50

indent() (in module ffc.cpp), 69

Indented (class in ffc.uflacs.language.format\_lines), 57

index (ffc.uflacs.language.cnodes.ForRange attribute), 50

index() (ffc.tensor.monomialextraction.MonomialFactor method), 21

index\_sum() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformer method), 16

index\_sum() (ffc.tensor.monomialextraction.MonomialTransformer method), 22

index\_sum() (ffc.uflacs.analysis.graph\_rebuild.ReconstructScalarSubexpression method), 28

index\_type (ffc.uflacs.language.cnodes.ForRange attribute), 50

indexed() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformer method), 16

indexed() (ffc.tensor.monomialextraction.MonomialTransformer method), 22

indexed() (ffc.uflacs.analysis.valuenumbering.ValueNumberer method), 32

indices (ffc.uflacs.language.cnodes.ArrayAccess attribute), 45

info() (in module ffc.log), 73

info\_blue() (in module ffc.log), 73

info\_green() (in module ffc.log), 73

info\_red() (in module ffc.log), 73

info\_usage() (in module ffc.main), 73

info\_version() (in module ffc.main), 74

init (ffc.uflacs.language.cnodes.For attribute), 49

init\_scopes() (ffc.uflacs.integralgenerator.IntegralGenerator method), 66

initialize\_data() (ffc.errorcontrol.errorcontrolgenerators.ErrorControlGenerators method), 8

initialize\_data() (ffc.errorcontrol.errorcontrolgenerators.UFL\_ErrorControlGenerators method), 9

initialize\_integral\_code() (in module ffc.representationutils), 76

initialize\_integral\_ir() (in module ffc.representationutils), 76

initializer\_list() (ffc.uflacs.backends.ufc.generator.ufc\_generator method), 43

insert\_nested\_dict() (in module ffc.utils), 77

int\_value() (ffc.uflacs.backends.ffc.access.FFCBackendAccess method), 34

int\_value() (ffc.uflacs.language.ufl\_to\_cnodes.UFL2CNodesMixin method), 59

integral\_type\_to\_entity\_dim() (in module ffc.representationutils), 76

IntegralGenerator (class in ffc.uflacs.integralgenerator), 65

integrate() (in module ffc.tensor.monomialintegration), 22

integrate\_block() (in module ffc.uflacs.build\_uflacs\_ir), 61

integrate\_block\_interior\_facets() (in module ffc.uflacs.build\_uflacs\_ir), 61

INTERNAL (ffc.tensor.monomialtransformation.MonomialIndex attribute), 23

interpolate\_vertex\_values() (ffc.uflacs.backends.ufc.finite\_element.ufc\_finite\_element method), 39

`interpolate_vertex_values()` (in module `ffc.interpolatevertexvalues`), 72  
`invert_dependencies()` (in module `ffc.uflacs.analysis.graph_ssa`), 29  
`is_constant()` (in module `ffc.quadrature.reduce_operations`), 18  
`is_modified_terminal()` (in module `ffc.uflacs.analysis.modified_terminals`), 31  
`is_negative_one_cexpr()` (in module `ffc.uflacs.language.cnodes`), 56  
`is_one_cexpr()` (in module `ffc.uflacs.language.cnodes`), 56  
`is_ones_table()` (in module `ffc.uflacs.elementtables`), 64  
`is_piecewise_table()` (in module `ffc.uflacs.elementtables`), 64  
`is_quadrature_table()` (in module `ffc.uflacs.elementtables`), 64  
`is_simple_inner_loop()` (in module `ffc.uflacs.language.cnodes`), 56  
`is_uniform` (`ffc.uflacs.build_uflacs_ir.preintegrated_block_data` attribute), 63  
`is_uniform` (`ffc.uflacs.build_uflacs_ir.premultiplied_block_data` attribute), 63  
`is_uniform_table()` (in module `ffc.uflacs.elementtables`), 65  
`is_zero_cexpr()` (in module `ffc.uflacs.language.cnodes`), 56  
`is_zeros_table()` (in module `ffc.uflacs.elementtables`), 65  
`iter_indented_lines()` (in module `ffc.uflacs.language.format_lines`), 57

## J

`J` (`ffc.tensor.monomialtransformation.MonomialTransformation` attribute), 23  
`J_component()` (`ffc.uflacs.backends.ffc.symbols.FFCBackendSymbols` method), 36  
`jacobian()` (`ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt` method), 11  
`jacobian()` (`ffc.quadrature.quadraturetransformer.QuadratureTransformer` method), 14  
`jacobian()` (`ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase` method), 16  
`jacobian()` (`ffc.uflacs.backends.ffc.access.FFCBackendAccess` method), 34  
`jacobian()` (`ffc.uflacs.backends.ffc.definitions.FFCBackendDefinitions` method), 16  
`jacobian_determinant()` (`ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt` method), 11  
`jacobian_determinant()` (`ffc.quadrature.quadraturetransformer.QuadratureTransformer` method), 14  
`jacobian_determinant()` (`ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase` method), 16  
`jacobian_determinant()` (`ffc.uflacs.backends.ffc.access.FFCBackendAccess` method), 34

`jacobian_determinant()` (`ffc.uflacs.backends.ffc.definitions.FFCBackendDefinitions` method), 35  
`jacobian_inverse()` (`ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt` method), 11  
`jacobian_inverse()` (`ffc.quadrature.quadraturetransformer.QuadratureTransformer` method), 14  
`jacobian_inverse()` (`ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase` method), 16  
`jacobian_inverse()` (`ffc.uflacs.backends.ffc.access.FFCBackendAccess` method), 34  
`jacobian_inverse()` (`ffc.uflacs.backends.ffc.definitions.FFCBackendDefinitions` method), 35  
`JINV` (`ffc.tensor.monomialtransformation.MonomialTransformation` attribute), 23  
`jit()` (in module `ffc.jitcompiler`), 72  
`jit_build()` (in module `ffc.jitcompiler`), 72  
`jit_generate()` (in module `ffc.jitcompiler`), 72

## L

`label()` (`ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase` method), 16  
`label()` (`ffc.uflacs.analysis.graph_rebuild.ReconstructScalarSubexpressions` method), 28  
`LE` (class in `ffc.uflacs.language.cnodes`), 50  
`LE` (`ffc.uflacs.language.precedence.PRECEDENCE` attribute), 58  
`le()` (`ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin` method), 59  
`leftover()` (in module `ffc.uflacs.language.cnodes`), 56  
`lhs` (`ffc.uflacs.language.cnodes.BinOp` attribute), 47  
`list_tensor()` (`ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase` method), 16  
`list_tensor()` (`ffc.uflacs.analysis.graph_rebuild.ReconstructScalarSubexpressions` method), 28  
`list_tensor()` (`ffc.uflacs.analysis.valuenumbering.ValueNumberer` method), 32  
`LiteralBool` (`ffc.uflacs.language.precedence.PRECEDENCE` attribute), 58  
`LiteralBool` (class in `ffc.uflacs.language.cnodes`), 50  
`LiteralInt` (class in `ffc.uflacs.language.cnodes`), 51  
`LiteralInt` (class in `ffc.uflacs.language.cnodes`), 51  
`LiteralString` (class in `ffc.uflacs.language.cnodes`), 51  
`ln()` (`ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase` method), 16  
`ln()` (`ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin` method), 59  
`log()` (in module `ffc.log`), 73  
`LOWEST` (`ffc.uflacs.language.precedence.PRECEDENCE` attribute), 58  
`LT` (class in `ffc.uflacs.language.cnodes`), 50  
`LT` (`ffc.uflacs.language.precedence.PRECEDENCE` attribute), 58



lt() (ffc.uflacs.language.ufl\_to\_cnodes.UFL2CNodesMixin method), 59

## M

ma\_data (ffc.uflacs.build\_uflacs\_ir.full\_block\_data\_t attribute), 61

ma\_data (ffc.uflacs.build\_uflacs\_ir.partial\_block\_data\_t attribute), 62

ma\_data\_t (class in ffc.uflacs.build\_uflacs\_ir), 61

ma\_index (ffc.uflacs.build\_uflacs\_ir.ma\_data\_t attribute), 61

main() (in module ffc.main), 74

make\_all\_element\_classnames() (in module ffc.representation), 76

make\_classname() (in module ffc.cpp), 69

make\_coordinate\_mapping\_jit\_classname() (in module ffc.representation), 76

make\_dofmap\_jit\_classname() (in module ffc.representation), 76

make\_finite\_element\_jit\_classname() (in module ffc.representation), 76

make\_integral\_classname() (in module ffc.cpp), 69

map\_component\_tensor\_arg\_components() (in module ffc.uflacs.analysis.indexing), 30

map\_facet\_points() (in module ffc.fiatinterface), 71

map\_indexed\_arg\_components() (in module ffc.uflacs.analysis.indexing), 30

map\_integral\_points() (in module ffc.representationutils), 76

mapping() (ffc.enrichedelement.EnrichedElement method), 70

mapping() (ffc.enrichedelement.SpaceOfReals method), 70

mapping() (ffc.mixedelement.MixedElement method), 74

mapping() (ffc.quadratureelement.QuadratureElement method), 75

mapping() (ffc.restrictedelement.RestrictedElement method), 77

mark\_active() (in module ffc.uflacs.analysis.dependencies), 26

mark\_image() (in module ffc.uflacs.analysis.dependencies), 26

mark\_partitions() (in module ffc.uflacs.analysis.graph\_ssa), 29

math\_function() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 16

math\_function() (ffc.uflacs.analysis.graph\_rebuild.ReconstructScalarSubexpressions method), 28

math\_function() (ffc.uflacs.language.ufl\_to\_cnodes.UFL2CNodesMixin method), 59

max\_cell\_subdomain\_id() (ffc.uflacs.backends.ufc.form.ufc\_form method), 42

max\_custom\_subdomain\_id() (ffc.uflacs.backends.ufc.form.ufc\_form method), 42

max\_cutcell\_subdomain\_id() (ffc.uflacs.backends.ufc.form.ufc\_form method), 42

max\_exterior\_facet\_subdomain\_id() (ffc.uflacs.backends.ufc.form.ufc\_form method), 42

max\_facet\_edge\_length() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformer method), 11

max\_facet\_edge\_length() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 14

max\_facet\_edge\_length() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 16

max\_interface\_subdomain\_id() (ffc.uflacs.backends.ufc.form.ufc\_form method), 42

max\_interior\_facet\_subdomain\_id() (ffc.uflacs.backends.ufc.form.ufc\_form method), 42

max\_overlap\_subdomain\_id() (ffc.uflacs.backends.ufc.form.ufc\_form method), 42

max\_value() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformer method), 11

max\_value() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 14

max\_value() (ffc.uflacs.analysis.graph\_rebuild.ReconstructScalarSubexpressions method), 28

max\_value() (ffc.uflacs.language.ufl\_to\_cnodes.RulesForC method), 58

max\_value() (ffc.uflacs.language.ufl\_to\_cnodes.RulesForC++ method), 59

max\_vertex\_subdomain\_id() (ffc.uflacs.backends.ufc.form.ufc\_form method), 42

members() (ffc.uflacs.backends.ufc.generator.ufc\_generator method), 43

MemZero() (in module ffc.uflacs.language.cnodes), 51

MemZeroRange() (in module ffc.uflacs.language.cnodes), 51

message (ffc.uflacs.language.cnodes.Throw attribute), 55

min\_facet\_edge\_length() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformer method), 11

min\_facet\_edge\_length() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 14

min\_facet\_edge\_length() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 16

method), 16

min\_value() (ffc.quadrature.optimisedquadraturetransformerbase.QuadratureTransformerBase method), 11

min\_value() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 14

min\_value() (ffc.uflacs.analysis.graph\_rebuild.ReconstructScalarSubexpressions method), 28

min\_value() (ffc.uflacs.language.ufl\_to\_cnodes.RulesForC namespace attribute), 59

min\_value() (ffc.uflacs.language.ufl\_to\_cnodes.RulesForC namespace attribute), 59

MixedElement (class in ffc.mixedelement), 74

Mod (class in ffc.uflacs.language.cnodes), 51

MOD (ffc.uflacs.language.precedence.PRECEDENCE attribute), 58

ModifiedTerminal (class in ffc.uflacs.analysis.modified\_terminals), 31

Monomial (class in ffc.tensor.monomialextraction), 21

MonomialArgument (class in ffc.tensor.monomialtransformation), 22

MonomialCoefficient (class in ffc.tensor.monomialtransformation), 22

MonomialDeterminant (class in ffc.tensor.monomialtransformation), 22

MonomialException, 21

MonomialFactor (class in ffc.tensor.monomialextraction), 21

MonomialForm (class in ffc.tensor.monomialextraction), 21

MonomialIndex (class in ffc.tensor.monomialtransformation), 23

MonomialSum (class in ffc.tensor.monomialextraction), 21

MonomialTransform (class in ffc.tensor.monomialtransformation), 23

MonomialTransformer (class in ffc.tensor.monomialextraction), 21

Mul (class in ffc.uflacs.language.cnodes), 51

MUL (ffc.uflacs.language.precedence.PRECEDENCE attribute), 58

multi\_index() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 16

multi\_index() (ffc.tensor.monomialextraction.MonomialTransformer method), 14

multi\_index() (ffc.uflacs.analysis.graph\_rebuild.ReconstructScalarSubexpressions method), 28

MultiIndex (class in ffc.tensor.multiindex), 23

multiply\_block() (in module ffc.uflacs.build\_uflacs\_ir), 62

multiply\_block\_interior\_facets() (in module ffc.uflacs.build\_uflacs\_ir), 62

name (ffc.uflacs.build\_uflacs\_ir.preintegrated\_block\_data\_t attribute), 63

name (ffc.uflacs.build\_uflacs\_ir.preintegrated\_block\_data\_t attribute), 63

name (ffc.uflacs.language.cnodes.Namespace attribute), 51

name (ffc.uflacs.language.cnodes.Symbol attribute), 54

name (ffc.uflacs.language.cnodes.Using attribute), 55

Namespace (class in ffc.uflacs.language.cnodes), 51

NaryOp (class in ffc.uflacs.language.cnodes), 51

NE (class in ffc.uflacs.language.cnodes), 51

NE (ffc.uflacs.language.precedence.PRECEDENCE attribute), 58

ne() (ffc.uflacs.language.ufl\_to\_cnodes.UFL2CNodesMixin method), 60

needs\_mesh\_entities() (ffc.uflacs.backends.ufc.dofmap.ufc\_dofmap method), 38

needs\_oriented\_jacobian() (in module ffc.representation), 76

needs\_oriented\_jacobian() (in module ffc.representationutils), 76

Neg (class in ffc.uflacs.language.cnodes), 52

NEG (ffc.uflacs.language.precedence.PRECEDENCE attribute), 58

negative\_restricted() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 16

negative\_restricted() (ffc.tensor.monomialextraction.MonomialTransformer method), 22

negative\_restricted() (ffc.uflacs.analysis.balancing.BalanceModifiers method), 25

New (class in ffc.uflacs.language.cnodes), 52

new\_symbol() (ffc.uflacs.analysis.valuenumbering.ValueNumberer method), 32

new\_symbols() (ffc.uflacs.analysis.valuenumbering.ValueNumberer method), 32

new\_temp\_symbol() (ffc.uflacs.integralgenerator.IntegralGenerator method), 66

Not (class in ffc.uflacs.language.cnodes), 52

NOT (ffc.uflacs.language.precedence.PRECEDENCE attribute), 58

not\_condition() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerBase method), 16

not\_condition() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 14

not\_condition() (ffc.uflacs.language.ufl\_to\_cnodes.UFL2CNodesMixin method), 60

Null (class in ffc.uflacs.language.cnodes), 52

num (ffc.quadrature.fraction.Fraction attribute), 10

num\_cells() (ffc.uflacs.backends.ufc.integrals.ufc\_custom\_integral method), 43

num\_coefficients() (ffc.uflacs.backends.ufc.form.ufc\_form method), 42

num\_components() (ffc.mixedelement.MixedElement method), 74

num\_coordinate\_component\_dofs() (in module

ffc.uflacs.backends.ffc.common), 34  
 num\_custom\_quadrature\_points()  
   (ffc.uflacs.backends.ffc.symbols.FFCBackendSymbols  
   method), 37  
 num\_element\_dofs() (ffc.uflacs.backends.ufc.dofmap.ufc\_dofmap  
   method), 38  
 num\_elements (ffc.uflacs.analysis.crsarray.CRSArray at-  
   tribute), 26  
 num\_entity\_closure\_dofs()  
   (ffc.uflacs.backends.ufc.dofmap.ufc\_dofmap  
   method), 38  
 num\_entity\_dofs() (ffc.uflacs.backends.ufc.dofmap.ufc\_dofmap  
   method), 38  
 num\_facet\_dofs() (ffc.uflacs.backends.ufc.dofmap.ufc\_dofmap  
   method), 38  
 num\_sub\_dofmaps() (ffc.uflacs.backends.ufc.dofmap.ufc\_dofmap  
   method), 38  
 num\_sub\_elements() (ffc.uflacs.backends.ufc.finite\_element  
   method), 39

## O

offset (ffc.uflacs.language.cnodes.FlattenedArray at-  
   tribute), 49  
 op (ffc.uflacs.language.cnodes.Add attribute), 45  
 op (ffc.uflacs.language.cnodes.AddressOf attribute), 45  
 op (ffc.uflacs.language.cnodes.And attribute), 45  
 op (ffc.uflacs.language.cnodes.Assign attribute), 45  
 op (ffc.uflacs.language.cnodes.AssignAdd attribute), 45  
 op (ffc.uflacs.language.cnodes.AssignAnd attribute), 46  
 op (ffc.uflacs.language.cnodes.AssignBitAnd attribute),  
   46  
 op (ffc.uflacs.language.cnodes.AssignBitOr attribute), 46  
 op (ffc.uflacs.language.cnodes.AssignBitXor attribute),  
   46  
 op (ffc.uflacs.language.cnodes.AssignDiv attribute), 46  
 op (ffc.uflacs.language.cnodes.AssignLShift attribute), 46  
 op (ffc.uflacs.language.cnodes.AssignMod attribute), 46  
 op (ffc.uflacs.language.cnodes.AssignMul attribute), 46  
 op (ffc.uflacs.language.cnodes.AssignOr attribute), 46  
 op (ffc.uflacs.language.cnodes.AssignRShift attribute),  
   46  
 op (ffc.uflacs.language.cnodes.AssignSub attribute), 46  
 op (ffc.uflacs.language.cnodes.BitAnd attribute), 47  
 op (ffc.uflacs.language.cnodes.BitNot attribute), 47  
 op (ffc.uflacs.language.cnodes.BitOr attribute), 47  
 op (ffc.uflacs.language.cnodes.BitXor attribute), 47  
 op (ffc.uflacs.language.cnodes.Dereference attribute), 49  
 op (ffc.uflacs.language.cnodes.Div attribute), 49  
 op (ffc.uflacs.language.cnodes.EQ attribute), 49  
 op (ffc.uflacs.language.cnodes.GE attribute), 50  
 op (ffc.uflacs.language.cnodes.GT attribute), 50  
 op (ffc.uflacs.language.cnodes.LE attribute), 50  
 op (ffc.uflacs.language.cnodes.LT attribute), 50  
 op (ffc.uflacs.language.cnodes.Mod attribute), 51  
 op (ffc.uflacs.language.cnodes.Mul attribute), 51  
 op (ffc.uflacs.language.cnodes.NE attribute), 51  
 op (ffc.uflacs.language.cnodes.Neg attribute), 52  
 op (ffc.uflacs.language.cnodes.Not attribute), 52  
 op (ffc.uflacs.language.cnodes.Or attribute), 52  
 op (ffc.uflacs.language.cnodes.Pos attribute), 52  
 op (ffc.uflacs.language.cnodes.PostDecrement attribute),  
   52  
 op (ffc.uflacs.language.cnodes.PostIncrement attribute),  
   52  
 op (ffc.uflacs.language.cnodes.PreDecrement attribute),  
   53  
 op (ffc.uflacs.language.cnodes.PreIncrement attribute),  
   53  
 op (ffc.uflacs.language.cnodes.Product attribute), 53  
 op (ffc.uflacs.language.cnodes.SizeOf attribute), 53  
 op (ffc.uflacs.language.cnodes.Sub attribute), 54  
 op (ffc.uflacs.language.cnodes.Sum attribute), 54  
 operation\_count() (in module  
   ffc.quadrature.reduce\_operations), 18  
 ops() (ffc.quadrature.expr.Expr method), 9  
 ops() (ffc.quadrature.fraction.Fraction method), 10  
 ops() (ffc.quadrature.product.Product method), 12  
 ops() (ffc.quadrature.sumobj.Sum method), 19  
 ops() (ffc.quadrature.symbol.Symbol method), 19  
 optimise\_code() (in module ffc.quadrature.symbolics), 20  
 optimize\_element\_tables() (in module  
   ffc.uflacs.elementtables), 65  
 optimize\_integral\_ir() (in module  
   ffc.quadrature.quadratureoptimization), 12  
 optimize\_integral\_ir() (in module  
   ffc.tensor.tensoroptimization), 24  
 optimize\_integral\_ir() (in module  
   ffc.uflacs.uflacsoptimization), 67  
 optimize\_ir() (in module ffc.optimization), 74  
 Or (class in ffc.uflacs.language.cnodes), 52  
 OR (ffc.uflacs.language.precedence.PRECEDENCE at-  
   tribute), 58  
 or\_condition() (ffc.uflacs.language.ufl\_to\_cnodes.UFL2CNodesMixin  
   method), 60  
 original\_coefficient\_position()  
   (ffc.uflacs.backends.ufc.form.ufc\_form  
   method), 42  
 outer\_join() (in module ffc.tensor.multiindex), 23

## P

pad\_dim() (in module ffc.uflacs.language.cnodes), 56  
 pad\_innermost\_dim() (in module  
   ffc.uflacs.language.cnodes), 56  
 padlen (ffc.uflacs.language.cnodes.ArrayDecl attribute),  
   45  
 parse\_optimise\_parameters() (in module  
   ffc.quadrature.parameters), 12



- parse\_uflacs\_optimization\_parameters() (in module  
     ffc.uflacs.build\_uflacs\_ir), 62
- partial\_block\_data\_t (class in ffc.uflacs.build\_uflacs\_ir),  
     62
- pdet\_m1() (in module  
     ffc.uflacs.backends.ufc.coordinate\_mapping),  
     38
- pick\_first() (in module ffc.utils), 77
- pick\_representation() (in module ffc.representation), 76
- piecewise\_ma\_index (ffc.uflacs.build\_uflacs\_ir.partial\_block\_data\_t  
     attribute), 62
- plot() (in module ffc.plot), 75
- points\_table() (ffc.uflacs.backends.ffc.symbols.FFCBackendSymbols  
     method), 37
- pop\_level() (in module ffc.log), 73
- Pos (class in ffc.uflacs.language.cnodes), 52
- POS (ffc.uflacs.language.precedence.PRECEDENCE at-  
     tribute), 58
- positive\_restricted() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase.  
     method), 16
- positive\_restricted() (ffc.tensor.monomialextraction.MonomialTransformation.  
     method), 22
- positive\_restricted() (ffc.uflacs.analysis.balancing.BalanceModifiers.  
     method), 25
- POST\_DEC (ffc.uflacs.language.precedence.PRECEDENCE at-  
     tribute), 58
- POST\_INC (ffc.uflacs.language.precedence.PRECEDENCE at-  
     tribute), 58
- PostDecrement (class in ffc.uflacs.language.cnodes), 52
- PostfixUnaryOp (class in ffc.uflacs.language.cnodes), 53
- PostIncrement (class in ffc.uflacs.language.cnodes), 52
- power() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformer.  
     method), 11
- power() (ffc.quadrature.quadraturetransformer.QuadratureTransformer.  
     method), 14
- power() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase.  
     method), 16
- power() (ffc.tensor.monomialextraction.MonomialTransformation.  
     method), 22
- power() (ffc.uflacs.analysis.graph\_rebuild.ReconstructScalarSubexpressions  
     method), 28
- power() (ffc.uflacs.language.ufl\_to\_cnodes.RulesForC  
     method), 59
- power() (ffc.uflacs.language.ufl\_to\_cnodes.RulesForCpp  
     method), 59
- Pragma (class in ffc.uflacs.language.cnodes), 53
- pragma (ffc.uflacs.language.cnodes.For attribute), 49
- pragma (ffc.uflacs.language.cnodes.ForRange attribute),  
     50
- PRE\_DEC (ffc.uflacs.language.precedence.PRECEDENCE at-  
     tribute), 58
- PRE\_INC (ffc.uflacs.language.precedence.PRECEDENCE at-  
     tribute), 58
- PRECEDENCE (class in ffc.uflacs.language.precedence),
- precedence (ffc.uflacs.language.cnodes.Add attribute), 45
- precedence (ffc.uflacs.language.cnodes.AddressOf  
     attribute), 45
- precedence (ffc.uflacs.language.cnodes.And attribute), 45
- precedence (ffc.uflacs.language.cnodes.ArrayAccess at-  
     tribute), 45
- precedence (ffc.uflacs.language.cnodes.AssignOp at-  
     tribute), 46
- precedence (ffc.uflacs.language.cnodes.BitAnd attribute),  
     47
- precedence (ffc.uflacs.language.cnodes.BitNot attribute),  
     47
- precedence (ffc.uflacs.language.cnodes.BitOr attribute),  
     47
- precedence (ffc.uflacs.language.cnodes.BitXor attribute),  
     47
- precedence (ffc.uflacs.language.cnodes.Call attribute), 48
- precedence (ffc.uflacs.language.cnodes.CExprLiteral at-  
     tribute), 47
- precedence (ffc.uflacs.language.cnodes.Conditional at-  
     tribute), 48
- precedence (ffc.uflacs.language.cnodes.Dereference at-  
     tribute), 49
- precedence (ffc.uflacs.language.cnodes.Div attribute), 49
- precedence (ffc.uflacs.language.cnodes.EQ attribute), 49
- precedence (ffc.uflacs.language.cnodes.GE attribute), 50
- precedence (ffc.uflacs.language.cnodes.GT attribute), 50
- precedence (ffc.uflacs.language.cnodes.LE attribute), 50
- precedence (ffc.uflacs.language.cnodes.LiteralBool at-  
     tribute), 50
- precedence (ffc.uflacs.language.cnodes.LiteralFloat at-  
     tribute), 51
- precedence (ffc.uflacs.language.cnodes.LiteralInt at-  
     tribute), 51
- precedence (ffc.uflacs.language.cnodes.LiteralString at-  
     tribute), 51
- precedence (ffc.uflacs.language.cnodes.LT attribute), 50
- precedence (ffc.uflacs.language.cnodes.Mod attribute),  
     51
- precedence (ffc.uflacs.language.cnodes.Mul attribute), 51
- precedence (ffc.uflacs.language.cnodes.NE attribute), 51
- precedence (ffc.uflacs.language.cnodes.Neg attribute), 52
- precedence (ffc.uflacs.language.cnodes.Not attribute), 52
- precedence (ffc.uflacs.language.cnodes.Null attribute), 52
- precedence (ffc.uflacs.language.cnodes.Or attribute), 52
- precedence (ffc.uflacs.language.cnodes.Pos attribute), 52
- precedence (ffc.uflacs.language.cnodes.PostDecrement  
     attribute), 52
- precedence (ffc.uflacs.language.cnodes.PostIncrement at-  
     tribute), 52
- precedence (ffc.uflacs.language.cnodes.PreDecrement at-  
     tribute), 53
- precedence (ffc.uflacs.language.cnodes.PreIncrement at-

tribute), 53  
 precedence (ffc.uflacs.language.cnodes.Product attribute), 53  
 precedence (ffc.uflacs.language.cnodes.SizeOf attribute), 54  
 precedence (ffc.uflacs.language.cnodes.Sub attribute), 54  
 precedence (ffc.uflacs.language.cnodes.Sum attribute), 54  
 precedence (ffc.uflacs.language.cnodes.Symbol attribute), 54  
 precedence (ffc.uflacs.language.cnodes.VerbatimExpr attribute), 55  
 PreDecrement (class in ffc.uflacs.language.cnodes), 53  
 PrefixUnaryOp (class in ffc.uflacs.language.cnodes), 53  
 PreIncrement (class in ffc.uflacs.language.cnodes), 53  
 preintegrated\_block\_data\_t (class in ffc.uflacs.build\_uflacs\_ir), 62  
 premultiplied\_block\_data\_t (class in ffc.uflacs.build\_uflacs\_ir), 63  
 primal\_forms() (ffc.errorcontrol.errorcontrolgenerators.ErrorControlGenerators method), 8  
 PRIMARY (ffc.tensor.monomialtransformation.MonomialTransformation attribute), 23  
 print\_error() (in module ffc.main), 74  
 Product (class in ffc.quadrature.product), 12  
 Product (class in ffc.uflacs.language.cnodes), 53  
 product() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt method), 11  
 product() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 14  
 product() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 16  
 product() (ffc.tensor.monomialextraction.MonomialTransformer method), 22  
 product() (ffc.uflacs.analysis.graph\_rebuild.ReconstructScalarSubexpressions method), 28  
 product() (ffc.uflacs.language.ufl\_to\_cnodes.UFL2CNodesMixin method), 60  
 push\_level() (in module ffc.log), 73  
 push\_row() (ffc.uflacs.analysis.crsarray.CRSArray method), 26

## Q

quadrature\_loop\_index() (ffc.uflacs.backends.ffc.symbols.FFCBackendSymbols method), 37  
 quadrature\_weight() (ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt method), 11  
 quadrature\_weight() (ffc.quadrature.quadraturetransformer.QuadratureTransformer method), 14  
 quadrature\_weight() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 16  
 QuadratureElement (class in ffc.quadratureelement), 75  
 QuadratureTransformer (class in ffc.quadrature.quadraturetransformer), 13  
 QuadratureTransformerBase (class in ffc.quadrature.quadraturetransformerbase), 14  
 QuadratureTransformerOpt (class in ffc.quadrature.optimisedquadraturetransformer), 10  
 rank() (ffc.uflacs.backends.ffc.form.ffc\_form method), 42  
 rebuild\_expression\_from\_graph() (in module ffc.uflacs.analysis.graph\_rebuild), 28  
 rebuild\_scalar\_graph\_from\_factorization() (in module ffc.uflacs.analysis.factorization), 27  
 rebuild\_with\_scalar\_subexpressions() (in module ffc.uflacs.analysis.graph\_rebuild), 28  
 ReconstructScalarSubexpressions (class in ffc.uflacs.analysis.graph\_rebuild), 28  
 reduce\_ops() (ffc.quadrature.expr.Expr method), 9  
 reduce\_ops() (ffc.quadrature.fraction.Fraction method), 10  
 reduce\_ops() (ffc.quadrature.product.Product method), 12  
 reduce\_ops() (ffc.quadrature.sumobj.Sum method), 19  
 reduce\_var() (ffc.quadrature.expr.Expr method), 9  
 reduce\_var() (ffc.quadrature.fraction.Fraction method), 10  
 reduce\_var() (ffc.quadrature.expr.Expr method), 9  
 reduce\_vartype() (ffc.quadrature.fraction.Fraction method), 10  
 reduce\_vartype() (ffc.quadrature.product.Product method), 12  
 reduce\_vartype() (ffc.quadrature.sumobj.Sum method), 19  
 reduction\_possible() (in module ffc.quadrature.reduce\_operations), 18  
 reference\_cell() (in module ffc.fiatinterface), 71  
 reference\_cell\_vertices() (in module ffc.fiatinterface), 71  
 reference\_cell\_volume() (ffc.uflacs.backends.ffc.access.FFCBackendAccess method), 34  
 reference\_facet\_volume() (ffc.uflacs.backends.ffc.access.FFCBackendAccess method), 35  
 reference\_facet\_volume() (ffc.uflacs.backends.ffc.access.FFCBackendAccess method), 34  
 reference\_facet\_volume() (ffc.uflacs.backends.ffc.access.FFCBackendAccess method), 35  
 reference\_grad() (ffc.uflacs.analysis.balancing.BalanceModifiers method), 25  
 reference\_grad() (ffc.uflacs.analysis.valuenumbering.ValueNumberer method), 32

reference\_normal() (ffc.uflacs.backends.ffc.access.FFCBackendSecondary (ffc.tensor.monomialtransformation.MonomialIndex attribute), 34

reference\_normal() (ffc.uflacs.backends.ffc.definitions.FFCBackendDefinitionHandling() (in module ffc.cpp), 70 method), 35

reference\_value() (ffc.uflacs.analysis.balancing.BalanceModelSideHandler() (in module ffc.log), 73 method), 25

reference\_value() (ffc.uflacs.analysis.valuenumbering.ValueNumbering() (in module ffc.log), 73 method), 32

reference\_value\_dimension() (ffc.uflacs.backends.ufc.finite\_element.ufc\_finite\_element method), 66

reference\_value\_rank() (ffc.uflacs.backends.ufc.finite\_element.ufc\_finite\_element method), 39

reference\_value\_size() (ffc.uflacs.backends.ufc.finite\_element.ufc\_finite\_element method), 39

ReferenceTensor (class in ffc.tensor.referencetensor), 24

remove\_unused() (in module ffc.cpp), 69

reorder\_entries() (in module ffc.tensor.tensorreordering), 24

replace\_indices() (ffc.tensor.monomialextraction.MonomialExtraction method), 21

restricted() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 16

restricted() (ffc.uflacs.analysis.valuenumbering.ValueNumbering method), 32

RestrictedElement (class in ffc.restrictedelement), 76

restriction\_domain() (ffc.restrictedelement.RestrictedElement method), 77

restrictions (ffc.uflacs.build\_uflacs\_ir.common\_block\_data\_t attribute), 61

restrictions (ffc.uflacs.build\_uflacs\_ir.full\_block\_data\_t attribute), 61

restrictions (ffc.uflacs.build\_uflacs\_ir.partial\_block\_data\_t attribute), 62

restrictions (ffc.uflacs.build\_uflacs\_ir.preintegrated\_block\_data\_t attribute), 63

restrictions (ffc.uflacs.build\_uflacs\_ir.premultiplied\_block\_data\_t attribute), 63

Return (class in ffc.uflacs.language.cnodes), 53

rhs (ffc.uflacs.language.cnodes.BinOp attribute), 47

RulesForC (class in ffc.uflacs.language.ufl\_to\_cnodes), 58

RulesForCpp (class in ffc.uflacs.language.ufl\_to\_cnodes), 59

scalar\_nary() (ffc.uflacs.analysis.graph\_rebuild.ReconstructSpaceSubexpressions method), 28

scalar\_value() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 16

scalar\_value() (ffc.tensor.monomialextraction.MonomialTransformation method), 22

scale\_loop() (in module ffc.uflacs.language.cnodes), 56

Scope (class in ffc.uflacs.language.cnodes), 53

SECONDARY (ffc.tensor.monomialtransformation.MonomialIndex attribute), 23

set\_float\_formatting() (in module ffc.cpp), 70

set\_handler() (in module ffc.log), 73

set\_indent() (in module ffc.log), 73

set\_number() (in module ffc.log), 73

set\_prefix() (in module ffc.log), 73

set\_var() (ffc.uflacs.integralgenerator.IntegralGenerator method), 66

sideeffect (ffc.uflacs.language.cnodes.AssignOp attribute), 46

sideeffect (ffc.uflacs.language.cnodes.Call attribute), 48

sideeffect (ffc.uflacs.language.cnodes.CExprOperator attribute), 47

sideeffect (ffc.uflacs.language.cnodes.CExprTerminal attribute), 47

sideeffect (ffc.uflacs.language.cnodes.PostDecrement attribute), 52

sideeffect (ffc.uflacs.language.cnodes.PostIncrement attribute), 52

sideeffect (ffc.uflacs.language.cnodes.PreDecrement attribute), 53

sideeffect (ffc.uflacs.language.cnodes.PreIncrement attribute), 53

signature() (ffc.uflacs.backends.ufc.generator.ufc\_generator method), 43

sin() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 16

sin() (ffc.uflacs.language.ufl\_to\_cnodes.UFL2CNodesMixin method), 60

sinh() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 16

sinh() (ffc.uflacs.language.ufl\_to\_cnodes.UFL2CNodesMixin method), 60

SizeOf (class in ffc.uflacs.language.cnodes), 53

SIZEOF (ffc.uflacs.language.precedence.PRECEDENCE attribute), 58

sizes (ffc.uflacs.language.cnodes.ArrayDecl attribute), 45

sort\_integrals() (in module ffc.quadrature.quadraturerepresentation), 13

space\_dimension() (ffc.enrichedelement.EnrichedElement method), 70

space\_dimension() (ffc.enrichedelement.SpaceOfReals method), 70

SpaceSubexpressions (ffc.mixedelement.MixedElement method), 74

space\_dimension() (ffc.quadratureelement.QuadratureElement method), 75

space\_dimension() (ffc.restrictedelement.RestrictedElement method), 77

space\_dimension() (ffc.uflacs.backends.ufc.finite\_element.ufc\_finite\_element method), 39

SpaceOfReals (class in `ffc.enrichedelement`), 70

`spatial_coordinate()` (`ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase` method), 16

`spatial_coordinate()` (`ffc.uflacs.backends.ffc.access.FFCBackendAccess` method), 34

`spatial_coordinate()` (`ffc.uflacs.backends.ffc.definitions.FFCBackendDefinitions` method), 36

`split_expression()` (in module `ffc.quadrature.reduce_operations`), 18

`split_parameters()` (in module `ffc.parameters`), 75

`sqrt()` (`ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase` method), 16

`sqrt()` (`ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin` method), 60

`Statement` (class in `ffc.uflacs.language.cnodes`), 54

`StatementList` (class in `ffc.uflacs.language.cnodes`), 54

`statements` (`ffc.uflacs.language.cnodes.StatementList` attribute), 54

`strides` (`ffc.uflacs.language.cnodes.FlattenedArray` attribute), 49

`strip_modified_terminal()` (in module `ffc.uflacs.analysis.modified_terminals`), 31

`strip_table_zeros()` (in module `ffc.uflacs.elementtables`), 65

`Sub` (class in `ffc.uflacs.language.cnodes`), 54

`SUB` (`ffc.uflacs.language.precedence.PRECEDENCE` attribute), 58

`SUBSCRIPT` (`ffc.uflacs.language.precedence.PRECEDENCE` attribute), 58

`sufficient_int()` (in module `ffc.uflacs.analysis.crsarray`), 26

`Sum` (class in `ffc.quadrature.sumobj`), 19

`Sum` (class in `ffc.uflacs.language.cnodes`), 54

`sum()` (`ffc.quadrature.optimisedquadraturetransformer.QuadratureTransformerOpt` method), 11

`sum()` (`ffc.quadrature.quadraturetransformer.QuadratureTransformer` method), 14

`sum()` (`ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase` method), 16

`sum()` (`ffc.tensor.monomialextraction.MonomialTransformer` method), 22

`sum()` (`ffc.uflacs.analysis.graph_rebuild.ReconstructScalarSubexpressions` method), 28

`sum()` (`ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin` method), 60

`Switch` (class in `ffc.uflacs.language.cnodes`), 54

`Symbol` (class in `ffc.quadrature.symbol`), 19

`Symbol` (class in `ffc.uflacs.language.cnodes`), 54

`symbol` (`ffc.uflacs.language.cnodes.ArrayDecl` attribute), 45

`symbol` (`ffc.uflacs.language.cnodes.VariableDecl` attribute), 55

`SYMBOL` (`ffc.uflacs.language.precedence.PRECEDENCE` attribute), 58

## T

`t` (`ffc.quadrature.expr.Expr` attribute), 9

`table_origin_t` (in module `ffc.uflacs.elementtables`), 65

`tabulatedata` (`ffc.uflacs.build_uflacs_ir.ma_data_t` attribute), 62

`tabulate()` (`ffc.enrichedelement.EnrichedElement` method), 70

`tabulate()` (`ffc.enrichedelement.SpaceOfReals` method), 70

`tabulate()` (`ffc.mixedelement.MixedElement` method), 74

`tabulate()` (`ffc.quadratureelement.QuadratureElement` method), 75

`tabulate()` (`ffc.restrictedelement.RestrictedElement` method), 77

`tabulate_basis()` (in module `ffc.quadrature.tabulate_basis`), 20

`tabulate_dof_coordinates()` (`ffc.uflacs.backends.ufc.finite_element.ufc_finite_element` method), 39

`tabulate_dofs()` (`ffc.uflacs.backends.ufc.dofmap.ufc_dofmap` method), 38

`tabulate_entity_closure_dofs()` (`ffc.uflacs.backends.ufc.dofmap.ufc_dofmap` method), 38

`tabulate_entity_dofs()` (`ffc.uflacs.backends.ufc.dofmap.ufc_dofmap` method), 38

`tabulate_facet_dofs()` (`ffc.uflacs.backends.ufc.dofmap.ufc_dofmap` method), 38

`tabulate_tensor()` (`ffc.uflacs.backends.ufc.integrals.ufc_integral` method), 44

`tabulate_tensor_comment()` (`ffc.uflacs.backends.ufc.integrals.ufc_integral` method), 44

`tan()` (`ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase` method), 16

`tan()` (`ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin` method), 60

`tanh()` (`ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase` method), 16

`tanh()` (`ffc.uflacs.language.ufl_to_cnodes.UFL2CNodesMixin` method), 60

`terminal()` (`ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase` method), 16

`terminal()` (`ffc.tensor.monomialextraction.MonomialTransformer` method), 22

`terminal()` (`ffc.uflacs.analysis.balancing.BalanceModifiers` method), 25

`terminal()` (`ffc.uflacs.analysis.graph_rebuild.ReconstructScalarSubexpressions` method), 28

`Throw` (class in `ffc.uflacs.language.cnodes`), 54

`topological_dimension()` (`ffc.uflacs.backends.ufc.coordinate_mapping.ufc_c` method), 38

`topological_dimension()` (`ffc.uflacs.backends.ufc.dofmap.ufc_dofmap` method), 38

[topological\\_dimension\(\)](#) (ffc.uflacs.backends.ufc.finite\_element method), 39  
[topological\\_dimension\(\)](#) (ffc.uflacs.backends.ufc.form.ufc\_form method), 43  
[total\\_shape\(\)](#) (in module ffc.uflacs.analysis.expr\_shapes), 26  
[transform\\_component\(\)](#) (in module ffc.representationutils), 76  
[transform\\_monomial\\_form\(\)](#) (in module ffc.tensor.monomialtransformation), 23  
[TransformedMonomial](#) (class in ffc.tensor.monomialtransformation), 23  
[transposed](#) (ffc.uflacs.build\_uflacs\_ir.common\_block\_data\_t attribute), 61  
[transposed](#) (ffc.uflacs.build\_uflacs\_ir.full\_block\_data\_t attribute), 61  
[transposed](#) (ffc.uflacs.build\_uflacs\_ir.partial\_block\_data\_t attribute), 62  
[transposed](#) (ffc.uflacs.build\_uflacs\_ir.preintegrated\_block\_data\_t attribute), 63  
[transposed](#) (ffc.uflacs.build\_uflacs\_ir.premultiplied\_block\_data\_t attribute), 63  
[transposed\(\)](#) (ffc.uflacs.analysis.graph\_rebuild.ReconstructScalarSubexpression method), 28  
[true](#) (ffc.uflacs.language.cnodes.Conditional attribute), 48  
[ttypes](#) (ffc.uflacs.build\_uflacs\_ir.common\_block\_data\_t attribute), 61  
[ttypes](#) (ffc.uflacs.build\_uflacs\_ir.full\_block\_data\_t attribute), 61  
[ttypes](#) (ffc.uflacs.build\_uflacs\_ir.partial\_block\_data\_t attribute), 62  
[ttypes](#) (ffc.uflacs.build\_uflacs\_ir.preintegrated\_block\_data\_t attribute), 63  
[ttypes](#) (ffc.uflacs.build\_uflacs\_ir.premultiplied\_block\_data\_t attribute), 63  
[typename](#) (ffc.uflacs.language.cnodes.ArrayDecl attribute), 45  
[typename](#) (ffc.uflacs.language.cnodes.New attribute), 52  
[typename](#) (ffc.uflacs.language.cnodes.VariableDecl attribute), 55

## U

[ufc\\_cell\\_integral](#) (class in ffc.uflacs.backends.ufc.integrals), 43  
[ufc\\_coordinate\\_mapping](#) (class in ffc.uflacs.backends.ufc.coordinate\_mapping), 38  
[ufc\\_custom\\_integral](#) (class in ffc.uflacs.backends.ufc.integrals), 43  
[ufc\\_cutcell\\_integral](#) (class in ffc.uflacs.backends.ufc.integrals), 43  
[ufc\\_dofmap](#) (class in ffc.uflacs.backends.ufc.dofmap), 38  
[ufc\\_exterior\\_facet\\_integral](#) (class in ffc.uflacs.backends.ufc.integrals), 44  
[ufc\\_finite\\_element](#) (class in ffc.uflacs.backends.ufc.finite\_element), 39  
[ufc\\_form](#) (class in ffc.uflacs.backends.ufc.form), 40  
[ufc\\_generator](#) (class in ffc.uflacs.backends.ufc.generator), 43  
[ufc\\_integral](#) (class in ffc.uflacs.backends.ufc.integrals), 44  
[ufc\\_interface\\_integral](#) (class in ffc.uflacs.backends.ufc.integrals), 44  
[ufc\\_interior\\_facet\\_integral](#) (class in ffc.uflacs.backends.ufc.integrals), 44  
[ufc\\_overlap\\_integral](#) (class in ffc.uflacs.backends.ufc.integrals), 44  
[ufc\\_restriction\\_offset\(\)](#) (in module ffc.uflacs.backends.fcc.common), 34  
[ufc\\_restriction\\_postfix\(\)](#) (in module ffc.uflacs.backends.fcc.symbols), 37  
[ufc\\_signature\(\)](#) (in module ffc.backends.ufc), 7  
[ufc\\_tvertex\\_integral](#) (class in ffc.uflacs.backends.ufc.integrals), 44  
[UFCElementNames](#) (class in ffc.backends.dolfin.capsules), 5  
[UFCSubexpression](#) (class in ffc.backends.dolfin.capsules), 5  
[UFL2CNodesMixin](#) (class in ffc.uflacs.language.ufl\_to\_cnodes), 59  
[UFL2CNodesTranslatorC](#) (class in ffc.uflacs.language.ufl\_to\_cnodes), 60  
[UFL2CNodesTranslatorCpp](#) (class in ffc.uflacs.language.ufl\_to\_cnodes), 60  
[uflacs\\_default\\_parameters\(\)](#) (in module ffc.uflacs.build\_uflacs\_ir), 63  
[UFL2ErrorControlGenerator](#) (class in ffc.errorcontrol.errorcontrolgenerators), 9  
[unames](#) (ffc.uflacs.build\_uflacs\_ir.common\_block\_data\_t attribute), 61  
[unames](#) (ffc.uflacs.build\_uflacs\_ir.full\_block\_data\_t attribute), 61  
[unames](#) (ffc.uflacs.build\_uflacs\_ir.partial\_block\_data\_t attribute), 62  
[unames](#) (ffc.uflacs.build\_uflacs\_ir.preintegrated\_block\_data\_t attribute), 63  
[unames](#) (ffc.uflacs.build\_uflacs\_ir.premultiplied\_block\_data\_t attribute), 63  
[UnaryOp](#) (class in ffc.uflacs.language.cnodes), 55  
[unexpected\(\)](#) (ffc.uflacs.analysis.graph\_rebuild.ReconstructScalarSubexpression method), 28  
[unique\\_psi\\_tables\(\)](#) (in module ffc.quadrature.quadratureutils), 17  
[unique\\_table\\_reference\\_t](#) (in module ffc.uflacs.elementtables), 65  
[unique\\_tables\(\)](#) (in module ffc.quadrature.quadratureutils), 17  
[update](#) (ffc.uflacs.language.cnodes.For attribute), 50



update\_cell() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 17  
 update\_facets() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 17  
 update\_points() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 17  
 update\_vertex() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 17  
 uses\_integral\_moments() (in module ffc.representation), 76  
 Using (class in ffc.uflacs.language.cnodes), 55  
 utility\_type() (ffc.uflacs.analysis.graph\_rebuild.ReconstructScalarSubexpressions method), 28

## V

v (ffc.quadrature.symbol.Symbol attribute), 19  
 val (ffc.quadrature.expr.Expr attribute), 9  
 validate\_jit\_parameters() (in module ffc.parameters), 75  
 validate\_parameters() (in module ffc.parameters), 75  
 value (ffc.uflacs.language.cnodes.Case attribute), 48  
 value (ffc.uflacs.language.cnodes.LiteralBool attribute), 50  
 value (ffc.uflacs.language.cnodes.LiteralFloat attribute), 51  
 value (ffc.uflacs.language.cnodes.LiteralInt attribute), 51  
 value (ffc.uflacs.language.cnodes.LiteralString attribute), 51  
 value (ffc.uflacs.language.cnodes.Return attribute), 53  
 value (ffc.uflacs.language.cnodes.VariableDecl attribute), 55  
 value\_dimension() (ffc.uflacs.backends.ufc.finite\_element.ufc\_finite\_element method), 40  
 value\_rank() (ffc.uflacs.backends.ufc.finite\_element.ufc\_finite\_element method), 40  
 value\_shape() (ffc.enrichedelement.EnrichedElement method), 70  
 value\_shape() (ffc.enrichedelement.SpaceOfReals method), 70  
 value\_shape() (ffc.mixedelement.MixedElement method), 74  
 value\_shape() (ffc.quadratureelement.QuadratureElement method), 75  
 value\_shape() (ffc.restrictedelement.RestrictedElement method), 77  
 value\_size() (ffc.uflacs.backends.ufc.finite\_element.ufc\_finite\_element method), 40  
 ValueNumberer (class in ffc.uflacs.analysis.valuenumbering), 31  
 values (ffc.uflacs.language.cnodes.ArrayDecl attribute), 45  
 variable() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 17  
 variable() (ffc.tensor.monomialextraction.MonomialTransformer method), 22

## W

warning() (in module ffc.log), 73  
 warning\_blue() (in module ffc.log), 73  
 warning\_green() (in module ffc.log), 73  
 warning\_red() (in module ffc.log), 73  
 weights\_table() (ffc.uflacs.backends.ffc.symbols.FFCBackendSymbols method), 37  
 While (class in ffc.uflacs.language.cnodes), 55  
 write\_code() (in module ffc.formatting), 72

## X

X\_component() (ffc.uflacs.backends.ffc.symbols.FFCBackendSymbols method), 36  
 x\_component() (ffc.uflacs.backends.ffc.symbols.FFCBackendSymbols method), 37

## Z

zero() (ffc.quadrature.quadraturetransformerbase.QuadratureTransformerBase method), 17  
 zero() (ffc.uflacs.backends.ffc.access.FFCBackendAccess method), 34  
 zero() (ffc.uflacs.language.ufl\_to\_cnodes.UFL2CNodesMixin method), 60