Boundary Condition Development in OpenFOAM

Introduction

In OpenFOAM, developing custom boundary conditions (BCs) allows users to define specific behaviors at boundaries that are not available in the standard set of BCs. This document provides a systematic guide for creating a custom BC template in OpenFOAM by leveraging an existing boundary condition as a starting point.

We use the **swirlInletVelocity** boundary condition, available in OpenFOAM's source folder, as a template to develop a new custom boundary condition. This guide focuses on the steps needed to rename, adapt, and compile the custom BC for integration into OpenFOAM.

Objective

To create a template for custom BC development by:

1. Copying an existing boundary condition folder.
2. Renaming files and classes.
3. Integrating and compiling the custom BC with OpenFOAM.

This guide is structured to make BC development approachable for new users.

Step-by-Step Guide

Step 1: Locate the Template

The OpenFOAM source directory contains predefined BCs. For this guide, navigate to the following folder:

```
$FOAM_SRC/finiteVolume/fields/fvPatchFields/derived/swirlInletVelocity/
```

This folder contains:

- `swirlInletVelocityFvPatchVectorField.C`
- `swirlInletVelocityFvPatchVectorField.H`

These files define the logic and behavior of the `swirlInletVelocity` boundary condition.

Step 2: Copy the Folder to Your Workspace

1. Create a new working folder in your user area (e.g., `$FOAM_USER_DIR`):

```
mkdir -p $FOAM_USER_DIR/customBC
```

2. Copy the **swirlInletVelocity** folder to your working directory and rename it:

```
bash
$FOAM_SRC/finiteVolume/fields/fvPatchFields/derived/swirlInletVelocity
$FOAM_USER_DIR/customBC/parabolicVelocity
```

Step 3: Rename the Files

1. Navigate to the copied folder:

```
cd $FOAM_USER_DIR/customBC/parabolicVelocity
```

2. Rename the source files:

```
mv swirlInletVelocityFvPatchVectorField.C parabolicVelocityFvPatchVectorField.C
mv swirlInletVelocityFvPatchVectorField.H parabolicVelocityFvPatchVectorField.H
```

Step 4: Modify the File Content

Open both `.C` and `.H` files, and replace all occurrences of `swirlInletVelocity` with the name of the new boundary condition, `parabolicVelocity`.

Step 5: Add the New BC to the Make Folder

To integrate the new BC into OpenFOAM, you need to register it in the Make system.

1. Navigate to the `Make` folder in the `finiteVolume` directory:

```
cd $FOAM_SRC/finiteVolume/Make/
```

2. Open the `files` file and add the path to your new BC files:

```
$(FOAM_USER_DIR)/customBC/parabolicVelocity/
parabolicVelocityFvPatchVectorField.C
```

3. Ensure the `options` file includes the necessary libraries (no changes are required if using the default setup).

Step 6: Compile the New Boundary Condition

1. Clean any previous builds and compile the new boundary condition:

```
wclean libso
wmake libso
```

2. Verify that the compiled library is available in the `$FOAM_USER_LIBBIN` directory.

Detailed Explanation of `swirlInletVelocityFvPatchVectorField` Files

These files are templates for developing a new boundary condition.

Contents of `.H` File

The `.H` file is the header file where the class is declared, and it defines the properties and methods of the boundary condition.

1. Class Declaration

- **Namespace**: The class is defined inside the `Foam` namespace, ensuring it integrates seamlessly into OpenFOAM.
- **Class Name**: `swirlInletVelocityFvPatchVectorField`.
  - It inherits from `fixedValueFvPatchVectorField`, which means it uses a fixed value boundary condition as its base.

2. Private Data Members

- `origin_`: A constant vector specifying the origin of rotation.
- `axis_`: A constant vector defining the axis of rotation.
- `axialVelocity_`: An `autoPtr<Function1<scalar>>` object, defining the axial velocity as a function.
- `radialVelocity_`: Similar to `axialVelocity_` but for radial velocity.
- `tangentialVelocity_`: Similar to `axialVelocity_` but for tangential velocity.

3. Constructors

- **From Patch and Internal Field**: Initializes the boundary condition based on the given patch and internal field.
- **From Dictionary**: Reads user-specified values (like `axis`, `origin`, and velocity functions) from the dictionary file.
- **Copy Constructor**: Handles copying boundary conditions while mapping them onto new patches or fields.

4. Key Member Functions

- `updateCoeffs()`: Calculates the boundary values for the velocity field based on the rotation axis, radius, and velocity profiles.
- `write()`: Writes the boundary condition's settings into output files (useful for debugging or restarting simulations).

5. Runtime Type Information

- `TypeName("swirlInletVelocity")`: Specifies the type name for this boundary condition to ensure OpenFOAM can identify it.

Contents of `.C` File

The `.C` file implements the functions declared in the header file.

1. Includes

- `swirlInletVelocityFvPatchVectorField.H`: Includes its own header file.
- `addToRunTimeSelectionTable.H`: Allows OpenFOAM to select this boundary condition at runtime.

2. Constructors

- Implements the constructors declared in the header file:
  - Initializes default values.
  - Reads and processes input from dictionaries.

3. `updateCoeffs()` Function

- **Purpose**: This is the core function that updates the boundary values for the velocity field.
  - Computes the unit vector along the rotation axis (`axisHat`).
  - Calculates the radius vector (`r`) in the plane of rotation.
  - Derives velocity components (axial, radial, tangential) from the provided functions (`axialVelocity`, `radialVelocity`, `tangentialVelocity`).
  - Combines these components to compute the final velocity at the boundary patch.

4. `write()` Function

- Outputs the boundary condition details (e.g., origin, axis, velocities) to the simulation files.

5. Runtime Selection

- The `makePatchTypeField` macro registers this boundary condition in OpenFOAM's runtime selection mechanism, allowing it to be specified in dictionaries under the `type` key.

Step-by-Step Template Structure

1. Key Sections in `.H` File

- **Namespace and Class Declaration**: Define the class under the `Foam` namespace and inherit from a base class, like `fixedValueFvPatchVectorField`.
- **Private Data**: Add data members to store boundary condition parameters (e.g., axis, origin, velocities).
- **Constructors**: Provide constructors to initialize the boundary condition from dictionaries or copy instances.
- **Member Functions**: Include `updateCoeffs()` to calculate the boundary values and `write()` to output them.

2. Key Sections in `.C` File

- **Includes**: Include the header file and any required OpenFOAM utilities.
- **Constructor Implementation**: Implement the constructors to initialize values from the dictionary or default settings.
- **Update Coefficients**: Define the mathematical logic to calculate the boundary values dynamically.
- **Runtime Registration**: Use macros like `makePatchTypeField` to register the boundary condition with OpenFOAM.

Understanding the Constructor Section

The constructors in this boundary condition file define how the `swirlInletVelocityFvPatchVectorField` class initializes its data members under various scenarios. Each constructor serves a specific purpose, and their implementation is crucial for integrating the custom boundary condition into OpenFOAM. Let's analyze each constructor in detail:

---

1. Default Constructor

```
Foam::swirlInletVelocityFvPatchVectorField::
swirlInletVelocityFvPatchVectorField
(
    const fvPatch& p,
    const DimensionedField<vector, volMesh>& iF
)
:
    fixedValueFvPatchField<vector>(p, iF),
    origin_(Zero),
    axis_(Zero),
    axialVelocity_(),
    radialVelocity_(),
    tangentialVelocity_()
{}
```

Purpose:
This is the **default constructor**, which initializes the boundary condition with default values.

Key Points:

1. **Inheritance Initialization**:
    - The line `fixedValueFvPatchField<vector>(p, iF)` calls the parent class constructor to handle standard boundary condition initialization.
    - It takes two arguments:
        - `p`: The associated boundary patch.
        - `iF`: The internal field of the vector type.
2. **Member Initialization**:
    - `origin_(Zero)`: Sets the origin of rotation to `(0, 0, 0)` by default.
    - `axis_(Zero)`: Sets the rotation axis to `(0, 0, 0)` by default.
    - `axialVelocity_`, `radialVelocity_`, `tangentialVelocity_`: These are initialized as empty function pointers, meaning no velocity profiles are defined yet.

Usage:
This constructor is used when no additional input is provided (e.g., if the user doesn't specify any boundary condition parameters in the dictionary).

---

2. Dictionary-Based Constructor

```
Foam::swirlInletVelocityFvPatchVectorField::
swirlInletVelocityFvPatchVectorField
(
    const fvPatch& p,
    const DimensionedField<vector, volMesh>& iF,
    const dictionary& dict
)
:
    fixedValueFvPatchField<vector>(p, iF, dict),
    origin_(dict.lookup("origin")),
    axis_(dict.lookup("axis")),
    axialVelocity_(Function1<scalar>::New("axialVelocity", dict)),
    radialVelocity_(Function1<scalar>::New("radialVelocity", dict)),
    tangentialVelocity_(Function1<scalar>::New("tangentialVelocity", dict))
{}
```

Purpose:
This constructor initializes the boundary condition using user-defined values from the dictionary file.

Key Points:

1. **Dictionary Input**:
    - The `dict` argument contains all user-specified parameters for this boundary condition (defined in the `boundaryField` section of the `0/U` file).
    - The values for `origin`, `axis`, and velocity profiles (`axialVelocity`, `radialVelocity`, `tangentialVelocity`) are extracted using the `dict.lookup()` function.
2. **Function1 Initialization**:
    - `Function1<scalar>::New("axialVelocity", dict)` dynamically initializes the `axialVelocity_` object from the dictionary.
    - This allows users to specify velocity profiles as analytical expressions (e.g., constant, linear, sinusoidal) in the dictionary.

Usage:
This is the **primary constructor**, used when the boundary condition is applied with user-specified parameters.

Example Dictionary Input:

```
type swirlInletVelocity;
origin (0 0 0);
axis (0 0 1);
axialVelocity constant 1.0;
radialVelocity constant 0.0;
tangentialVelocity uniform 0.5;
```

---

3. Copy Constructor (With Mapper)

```
Foam::swirlInletVelocityFvPatchVectorField::
swirlInletVelocityFvPatchVectorField
(
    const swirlInletVelocityFvPatchVectorField& ptf,
    const fvPatch& p,
```

```
    const DimensionedField<vector, volMesh>& iF,
    const fvPatchFieldMapper& mapper
)
:
    fixedValueFvPatchField<vector>(ptf, p, iF, mapper),
    origin_(ptf.origin_),
    axis_(ptf.axis_),
    axialVelocity_(ptf.axialVelocity_, false),
    radialVelocity_(ptf.radialVelocity_, false),
    tangentialVelocity_(ptf.tangentialVelocity_, false)
{}
```

Purpose:
This constructor is used when the boundary condition needs to be **mapped** from one patch to another. This might happen when the mesh topology changes, and the boundary condition must adapt to the new configuration.

Key Points:

1. **Mapping Argument**:
    - The `mapper` argument ensures that the boundary condition values are transferred correctly to the new patch.
    - This is crucial for dynamic mesh cases (e.g., sliding meshes or deforming geometries).
2. **Shallow Copy of Function1 Objects**:
    - `axialVelocity_(ptf.axialVelocity_, false)`: This ensures that the velocity profiles are copied without duplicating the underlying data.

Usage:
Used in cases where boundary condition instances are copied between patches.

---

4. Copy Constructor (Without Mapper)

```
Foam::swirlInletVelocityFvPatchVectorField::
swirlInletVelocityFvPatchVectorField
(
    const swirlInletVelocityFvPatchVectorField& ptf,
    const DimensionedField<vector, volMesh>& iF
)
:
    fixedValueFvPatchField<vector>(ptf, iF),
    origin_(ptf.origin_),
    axis_(ptf.axis_),
    axialVelocity_(ptf.axialVelocity_, false),
    radialVelocity_(ptf.radialVelocity_, false),
    tangentialVelocity_(ptf.tangentialVelocity_, false)
{}
```

Purpose:
This is a **simple copy constructor**, used when a boundary condition instance needs to be duplicated without remapping (e.g., for internal operations).

Key Points:

- Similar to the previous constructor, but without the `mapper` argument.
- Copies data from an existing `swirlInletVelocityFvPatchVectorField` object to create a new instance.

Summary Table

Constructor Type	Purpose	Key Feature
Default Constructor	Initializes with default values.	Origin, axis, and velocity profiles are set to zero or uninitialized.
Dictionary-Based Constructor	Reads user-defined values from a dictionary file.	Allows customization via `boundaryField` dictionary.
Copy Constructor (With Mapper)	Maps boundary condition from one patch to another.	Used for dynamic or changing mesh topologies.
Copy Constructor (Without Mapper)	Creates a duplicate of an existing boundary condition without remapping.	Useful for internal operations or static configurations.

Detailed Explanation of the Code in `.H` File

This section of the `.H` file defines the **constructors** and **member functions** for the `swirlInletVelocityFvPatchVectorField` boundary condition class. Each function serves a specific purpose for creating and managing instances of the boundary condition. Let's go through each part in detail.

---

1. Constructor Definitions

a) Default Constructor (Patch and Internal Field)

```
swirlInletVelocityFvPatchVectorField
(
    const fvPatch&,
    const DimensionedField<vector, volMesh>&
);
```

Purpose:

- This constructor creates a boundary condition object using the associated patch (`fvPatch`) and the internal field (`DimensionedField<vector, volMesh>`).
- It does not rely on external inputs like dictionaries or mapping.

Use Case:

- Typically used when no user-defined parameters are required, and default values suffice.

---

b) Dictionary-Based Constructor

```
swirlInletVelocityFvPatchVectorField
(
    const fvPatch&,
    const DimensionedField<vector, volMesh>&,
    const dictionary&
);
```

Purpose:

- This constructor reads parameters for the boundary condition from a dictionary (e.g., `0/U` file).
- It initializes additional data like velocity profiles, origin, and axis.

Use Case:

- Allows users to specify custom properties for the boundary condition via input files.

---

c) Mapping Constructor

```
swirlInletVelocityFvPatchVectorField
```

```
(
    const swirlInletVelocityFvPatchVectorField&,
    const fvPatch&,
    const DimensionedField<vector, volMesh>&,
    const fvPatchFieldMapper&
);
```

Purpose:

- This constructor maps an existing `swirlInletVelocityFvPatchVectorField` object to a new patch.
- It ensures that the boundary condition is consistent across meshes with different patch structures.

Use Case:

- Useful for simulations involving mesh deformation or dynamic patches.

Special Argument (`fvPatchFieldMapper`):

- Handles the mapping of values between the old patch and the new patch.

---

d) Deleted Copy Constructor

```
swirlInletVelocityFvPatchVectorField
(
    const swirlInletVelocityFvPatchVectorField&
) = delete;
```

Purpose:

- Disallows the use of a default copy constructor.
- Prevents shallow copies of the boundary condition without explicitly defining how the internal field (`DimensionedField`) should be handled.

Reason for Deletion:

- Copying the boundary condition directly could lead to unintended behavior if the internal field references are not updated correctly.

---

e) Copy Constructor with Internal Field Reference

```
swirlInletVelocityFvPatchVectorField
(
    const swirlInletVelocityFvPatchVectorField&,
    const DimensionedField<vector, volMesh>&
);
```

Purpose:

- This constructor allows for explicit copying of the boundary condition with the correct reference to the internal field.

Use Case:

- Used in cases where the boundary condition needs to be duplicated, ensuring proper linking to the correct internal field.

---

f) Clone Function

```
virtual tmp<fvPatchVectorField> clone
(
    const DimensionedField<vector, volMesh>& iF
) const
{
    return tmp<fvPatchVectorField>
    (
        new swirlInletVelocityFvPatchVectorField(*this, iF)
    );
}
```

Purpose:

- Provides a mechanism to create a copy of the boundary condition object dynamically at runtime.
- Ensures that the new object references the correct internal field.

Key Features:

- Returns a `tmp<fvPatchVectorField>` object, which is a managed smart pointer in OpenFOAM.
- Uses the copy constructor internally to create the new object.

Use Case:

- Supports dynamic polymorphism, allowing the boundary condition to be cloned when required (e.g., during mesh changes or initialization).

---

2. Member Function Definitions

a) `updateCoeffs` Function

```
virtual void updateCoeffs();
```

Purpose:

- Updates the coefficients used in the boundary condition.
- Typically invoked during solver iterations to account for changes in time or other parameters.

Use Case:

- Required for boundary conditions where the values depend on time, space, or flow variables (e.g., sinusoidal or dynamic profiles).

---

b) `write` Function

```
virtual void write(Ostream&) const;
```

Purpose:

- Writes the current state of the boundary condition to the output stream.
- Used for logging, debugging, or writing data to files.

Key Points:

- Ensures that all parameters (e.g., origin, axis, velocity profiles) are saved for future reference.
- Overridden from the base class to include custom behavior specific to this boundary condition.

Use Case:

- Enables OpenFOAM to output boundary condition details in case files.

Summary Table

Section	Purpose	Key Features
Default Constructor	Initializes boundary condition with default values.	No external input required.
Dictionary-Based Constructor	Reads parameters from a dictionary.	Allows user customization.
Mapping Constructor	Maps boundary condition to a new patch.	Handles dynamic meshes and topological changes.
Deleted Copy Constructor	Prevents shallow copying of boundary condition.	Ensures proper reference management.
Copy Constructor with Field Ref	Allows explicit copying with correct internal field reference.	Ensures proper linking to internal data.
Clone Function	Creates a new copy of the boundary condition dynamically at runtime.	Supports polymorphism and runtime flexibility.
`updateCoeffs`	Updates coefficients dynamically during simulation.	Essential for time-varying or spatially varying boundary conditions.
`write`	Outputs the current state of the boundary condition.	Useful for logging and debugging.

Task 10: Boundary Condition Development for Advanced Simulations

Objective:

Develop and implement custom boundary conditions (BCs) for velocity and temperature fields in OpenFOAM. These BCs are designed for specific scenarios such as time-varying, spatially varying, and oscillatory profiles. Students will use the **pimpleHeatFoam** solver (developed in earlier tasks) to simulate forced convection for cavity and pipe flow cases.

Subtasks

Subtask 1: Fully Developed Parabolic Velocity Profile for Pipe Flow

- **Scenario**: Simulate steady, fully developed parabolic velocity profile at the inlet of a circular pipe.
- **BC Description**:

$$u(r) = u_{max}\,(1 - \frac{r^2}{R^2})$$

  Where:

  - u_{max}: Maximum velocity at the centerline.
  - R: Pipe radius.
  - r: Radial distance from the centerline.
- **Test Case**:
  - **Geometry**:
    - **2D Pipe Flow**: $L = 5\,\text{m}, R = 0.1\,\text{m}.$
    - **3D Pipe Flow**: $L = 5\,\text{m}, R = 0.1\,\text{m}.$
    - Meshes for both cases are provided in the test case folder.
  - **Boundary Conditions**:

Boundary	Velocity (U)	Temperature (T)
Inlet	Parabolic velocity profile	$T = 300\,\text{K}$
Outlet	Zero-gradient	Zero-gradient
Wall	No-slip	Fixed value $T = 400\,\text{K}$

- **Expected Results**: A parabolic velocity profile at the inlet and fully developed flow within the pipe.

Subtask 2: Time-Varying Fully Developed Velocity Profile for Pipe Flow

- **Scenario**: Extend Subtask 1 by introducing time dependence to the velocity profile.
- **BC Description**:

$$u(r,t) = u_{max}(t)\,(1 - \frac{r^2}{R^2}), \quad u_{max}(t) = u_0 + A\sin(\omega t)$$

  Where:

  - u_0: Base velocity.
  - A: Amplitude of oscillation.
  - ω: Oscillation frequency.
- **Test Case**:
  - **Geometry**: Same as Subtask 1.
  - **Boundary Conditions**:

Boundary	Velocity (U)	Temperature (T)
Inlet	Time-varying parabolic profile	$T = 300\,\mathrm{K}$
Outlet	Zero-gradient	Zero-gradient
Wall	No-slip	Fixed value $T = 400\,\mathrm{K}$

- **Flow Conditions**: Transient flow with $u_0 = 1.0\,\mathrm{m/s}, A = 0.5\,\mathrm{m/s}, \omega = 2\pi/5\,\mathrm{rad/s}$.
- **Expected Results**: Oscillatory velocity profile at the inlet, propagating downstream.

Subtask 3: Sinusoidal Temperature Variation (1x1 Cavity)

- **Scenario**: Simulate sinusoidal temperature variation at the top wall of a 2D square cavity using **forced convection**.
- **BC Description**:

$$T(t) = T_0 + A_T\sin(\omega t)$$

  Where:

  - $T_0 = 300\,\mathrm{K}$: Base temperature.
  - $A_T = 50\,\mathrm{K}$: Amplitude of oscillation.
  - $\omega = 2\pi/10\,\mathrm{rad/s}$: Frequency.
- **Test Case**:
  - **Geometry**: Square cavity, $1\,\mathrm{m} \times 1\,\mathrm{m}$.
  - **Boundary Conditions**:

Boundary	Velocity (U)	Temperature (T)
Top wall	Moving wall, $Re = 1000$	Sinusoidal variation
Bottom wall	No-slip	Fixed value $T = 300\,\mathrm{K}$
Left wall	No-slip	Fixed value $T = 300\,\mathrm{K}$
Right wall	No-slip	Fixed value $T = 300\,\mathrm{K}$

- **Expected Results**: Oscillatory temperature profile propagates into the cavity, influencing the flow dynamics.

Subtask 4: Sinusoidal Spatially Varying Temperature Profile (1x1 Cavity)

- **Scenario**: Implement a spatially varying sinusoidal temperature profile at the top wall of the cavity.
- **BC Description**:

$$T(x) = T_0 + A_T \sin(kx)$$

  Where:

  - $T_0 = 300\,\mathrm{K}$: Base temperature.
  - $A_T = 50\,\mathrm{K}$: Amplitude.
  - $k = \pi/2$: Spatial frequency.
- **Test Case**:
  - **Geometry**: Same as Subtask 3.
  - **Boundary Conditions**:

Boundary	Velocity (U)	Temperature (T)
Top wall	Moving wall, $Re = 1000$	Spatial sinusoidal variation
Bottom wall	No-slip	Fixed value $T = 300\,\mathrm{K}$
Left wall	No-slip	Fixed value $T = 300\,\mathrm{K}$
Right wall	No-slip	Fixed value $T = 300\,\mathrm{K}$

- **Expected Results**: Spatial temperature variation propagates into the cavity, creating complex flow patterns.

Subtask 5: Oscillatory Velocity BC (1x1 Cavity)

- **Scenario**: Apply an oscillatory velocity boundary condition at the inlet of a cavity to simulate periodic flow.

- **BC Description**:

$$U(t) = U_0 + A\sin(\omega t)$$

  Where:

  - $U_0 = 0.5\,\text{m/s}$: Mean velocity.
  - $A = 0.2\,\text{m/s}$: Amplitude of oscillation.
  - $\omega = 2\pi/10\,\text{rad/s}$: Frequency.
- **Test Case**:
  - **Geometry**: Same as Subtask 3.
  - **Boundary Conditions**:

Boundary	Velocity (U)	Temperature (T)
Top wall	Oscillatory velocity	Fixed value $T = 300\,\text{K}$
Bottom wall	No-slip	Fixed value $T = 300\,\text{K}$
Left wall	No-slip	Fixed value $T = 300\,\text{K}$
Right wall	No-slip	Fixed value $T = 300\,\text{K}$

- **Expected Results**: Time-dependent velocity profile drives oscillatory flow within the cavity.
