Basilisk Mode for Emacs

Comprehensive User Manual

Author: Arun K Eswara
email: eswara.arun@gmail.com

Last updated: 28th February 2025

Basilisk Mode Version 1.0

Contents

Chapter 1

Introduction

1.1 What is Basilisk?

Basilisk is a free software program for solving partial differential equations on adaptive Cartesian meshes. It is particularly designed for computational fluid dynamics (CFD) and offers a variety of features for handling complex fluid simulations. Basilisk uses a C-based syntax with specialized extensions for mathematical operations, adaptive grid handling, and parallel computing through MPI.

1.2 Purpose of Basilisk Mode for Emacs

The Basilisk Mode for Emacs (`basilisk_setup.el`) enhances the standard C-mode in Emacs to better support Basilisk code development. It provides:

- Specialized syntax highlighting for Basilisk keywords and constructs
- Code templates for common Basilisk patterns
- Integrated compilation and execution tools
- Documentation access and help features
- Navigation aids and refactoring tools

This manual provides a comprehensive guide to installing, using, and customizing the Basilisk Mode for Emacs.

Chapter 2

Installation

2.1 Requirements

Before installing the Basilisk Mode for Emacs, ensure you have the following prerequisites:

- **Emacs**: Version 24.3 or later (tested on Emacs 29.1)
- **Basilisk**: Installed locally with the `qcc` compiler available in your PATH
- **MPI (Optional)**: For MPI-related features, an MPI implementation (e.g., OpenMPI) with `mpicc` and `mpirun`
- **C Compiler**: A C99-compliant compiler (e.g., `gcc`) for non-MPI compilation
- **Shell**: Bash shell (for `bash -c` execution). Ensure your `.bashrc` file is properly configured with environment settings

2.2 Installation Steps

Follow these steps to install and set up the Basilisk Mode for Emacs:

2.2.1 Step 1: Download the Mode

Save the `basilisk_setup.el` file to a directory in your Emacs load path, such as `~/.emacs.d/lisp/`.
Download link: `https://github.com/AdityasOcean/basilisk_setup.el`

2.2.2 Step 2: Update Your Emacs Configuration

Add the following lines to your `~/.emacs` or `~/.emacs.d/init.el`:

```
(add-to-list 'load-path "~/.emacs.d/lisp/")
(require 'basilisk_setup)
```

Alternatively, you can load the file manually whenever needed:

```
M-x load-file RET ~/.emacs.d/lisp/basilisk_setup.el RET
```

2.2.3 Step 3: Verify the Installation

To verify that the Basilisk Mode has been correctly installed:

1. Restart Emacs or evaluate your configuration with `M-x eval-buffer` in your `init.el`.
2. Open a `.c` or `.h` file; Basilisk mode should activate automatically.
3. A message should appear in the minibuffer: "Basilisk features enabled in C mode".
4. Check for the "Basilisk" menu in the menu bar.
5. Try a keybinding like `C-c e` to insert an event block.

Chapter 3

Syntax Highlighting

The Basilisk Mode enhances C-mode syntax highlighting with specific colors and fonts for Basilisk constructs, making code more readable and errors easier to spot.

3.1 Highlighted Elements

3.1.1 Control Keywords

Basilisk-specific control flow keywords are highlighted in `font-lock-keyword-face`:

- `event`
- `foreach`
- `foreach_face`
- `trace`
- `reduction`
- and others

3.1.2 MPI Keywords

MPI functions are highlighted in `font-lock-builtin-face`:

- `MPI_Allreduce`
- `mpi_all_reduce`
- `MPI_Bcast`
- `MPI_Init`
- `MPI_Finalize`
- and other MPI functions

3.1.3 Types and Constants

Basilisk-specific types are highlighted in `font-lock-type-face`:

- `scalar`
- `vector`
- `tensor`
- `coord`
- and other Basilisk types

Constants are highlighted in `font-lock-constant-face`:

- `PI`
- `M_PI`
- `HUGE`
- and other constants

3.1.4 Functions and Variables

Standard Basilisk functions are highlighted in `font-lock-function-name-face`:

- `solve`
- `adapt_wavelet`
- `restore`
- `boundary`
- and other functions

Common variables used in simulations are highlighted:

- `u` (velocity)
- `p` (pressure)
- `rho` (density)
- and other common simulation variables

3.1.5 Preprocessor Directives

Standard and Basilisk-specific preprocessor directives are highlighted:

- `#include`
- `#define`
- `_MPI`
- `LAYERS`
- and other directives

3.1.6 Special Patterns

Special syntax patterns receive custom highlighting:

- Event definitions (e.g., `event init (t = 0) { ... }`)
- Field access (e.g., `u[]`)
- Dimension indexing (e.g., `u[x]`, `u[y]`)

3.2 Example of Highlighted Code

```c
#include "grid/octree.h"
#include "navier-stokes/centered.h"
#include "tracer.h"

scalar T[];
face vector uf[];

#define LEVEL 8

int main() {
  init_grid(1 << LEVEL);

  // Set boundary conditions
  u.n[top] = neumann(0);
  u.t[top] = neumann(0);
  T[bottom] = dirichlet(1);
  T[top] = dirichlet(0);

  run();
}

event init (t = 0) {
  foreach() {
    T[] = y < 0.5 ? 1 : 0;
    u.x[] = 0;
    u.y[] = 0;
  }
}

event tracer_advection (i++) {
  dt = dtnext (0.1);

  foreach_face()
    uf.x[] = 0.5*(u.x[] + u.x[-1]);

  advection ({T}, uf, dt);
}

event adapt (i++) {
  adapt_wavelet ({T, u.x, u.y}, (double[]){1e-2, 1e-2, 1e-2}, LEVEL);
}
```

Chapter 4

Code Templates

The Basilisk Mode offers a variety of pre-defined code templates that can be quickly inserted using keyboard shortcuts. These templates save time and help maintain consistent code structure.

4.1 Event Templates

Events are a core Basilisk construct for scheduling code execution.

4.1.1 Basic Event Template (C-c e)

Inserts a basic event block with customizable name and timing.

```
event name (t = 0) {
  // Code here
}
```

When you use the `C-c e` shortcut, you'll be prompted for:

- Event name (default: "init")
- Event timing (default: "t = 0")

4.1.2 Common Event Types

- **Initialization**: `event init (t = 0)`
- **Time-stepping**: `event advance (i++)`
- **Output**: `event output (t += 0.1)`
- **Termination**: `event end (t = end_time)`
- **Adaptation**: `event adapt (i++)`

4.2 Loop Templates

4.2.1 foreach Loop (C-c f)

Inserts a basic `foreach` loop, which iterates over all cells in the domain.

```
foreach() {
  // Code here
}
```

4.2.2 foreach_face Loop (C-c a)

Inserts a `foreach_face` loop, which iterates over all faces in a specified direction.

```
foreach_face(direction) {
  // Code here
}
```

When using this template, you'll be prompted for the direction (x, y, or z).

4.2.3 Reduction Loop (C-c r)

Inserts a `foreach` loop with reduction operation for parallel calculations.

```
foreach (reduction(operator:variable)) {
  // Code here
}
```

You'll be prompted for:

- Reduction operator (e.g., +, max, min)
- Variable to reduce

4.3 Function Templates

4.3.1 Basic Main Function (C-c i)

Inserts a basic `main()` function template.

```
int main() {
  init_grid(N);
  run();
}
```

4.3.2 MPI Main Function (C-c o)

Inserts an MPI-enabled `main()` function.

```
int main(int argc, char * argv[]) {
  MPI_Init(&argc, &argv);
  init_grid(N);
  run();
  MPI_Finalize();
}
```

4.3.3 MPI Function Template (C-c m)

Inserts a template for an MPI function call. You'll be prompted to select from:

- `mpi_all_reduce`
- `MPI_Bcast`
- `MPI_Barrier`
- Other common MPI functions

4.4 Data Structure Templates

4.4.1 Struct Definition (C-c s)

Inserts a `typedef struct` with default fields.

```c
typedef struct {
  double field1;
  double field2;
  // Add more fields as needed
} StructName;
```

4.5 Algorithm Templates

4.5.1 Solver Template (C-c v)

Inserts a basic diffusion solver template.

```c
event diffusion (i++) {
  dt = dtnext (CFL);
  scalar d[];
  foreach() {
    d[] = dt*D[];
  }
  diffusion (T, dt, d);
}
```

Chapter 5

Compilation Methods

Compilation methods in the Basilisk Mode refer to different ways of compiling Basilisk code for different purposes and execution environments. The mode provides a comprehensive interface to compile code with various options through the `C-c c` command. Version 1.0 introduces a dual-mode execution system that allows compilation and execution either through Emacs or via a terminal.

5.1 Understanding Basilisk Compilation

5.1.1 The Basilisk Compiler: qcc

Basilisk uses a specialized compiler wrapper called `qcc` (Quick C Compiler), which processes Basilisk-specific syntax and transforms it into standard C code before passing it to the underlying C compiler. The `qcc` compiler handles:

- Basilisk's specialized syntax (foreach loops, event blocks, etc.)
- Automatic code generation for adaptive grid operations
- Stencil operations for numerical calculations
- Domain decomposition for parallel execution

5.1.2 Compilation Workflow

The general workflow for compiling a Basilisk code is:

1. **Preprocessing**: The `qcc` compiler processes Basilisk-specific syntax.
2. **Transformation**: Basilisk code is transformed into standard C code.
3. **Compilation**: The transformed code is compiled using a C compiler (gcc, clang, etc.).
4. **Linking**: The code is linked with appropriate libraries (math library, MPI libraries, etc.).

5.2 Available Compilation Methods

The Basilisk Mode provides several predefined compilation methods, each tailored for specific needs. Version 1.0 ensures proper handling of file extensions (preserving `.c` in compilation commands).

5.2.1 Basic (No MPI) Compilation

This is the simplest compilation method, suitable for sequential (non-parallel) simulations:

```
qcc -Wall code.c -o code -lm
```

- `-Wall`: Enables all warnings
- `-o code`: Specifies the output executable name
- `-lm`: Links with the math library

5.2.2 Optimized (No MPI) Compilation

Similar to the basic method but with optimization flags:

```
qcc -Wall -O2 code.c -o code -lm
```

- `-O2`: Level 2 optimization for better performance

5.2.3 Debug (No MPI) Compilation

Includes debugging information:

```
qcc -Wall -g -O0 code.c -o code -lm
```

- `-g`: Adds debugging symbols
- `-O0`: Disables optimization for better debugging

5.2.4 Makefile (No MPI) Compilation

Uses a Makefile for compilation:

```
make code.tst
```

- Requires a properly configured Makefile in the project directory

5.2.5 MPI Manual Compilation

For parallel simulations using MPI (Message Passing Interface):

```
CC99='mpicc -std=c99' qcc -Wall -O2 -D_MPI=n code.c -o code -lm
```

- `CC99='mpicc -std=c99'`: Sets the C compiler to `mpicc` with C99 standard
- `-D_MPI=n`: Defines the MPI macro with `n` processes (user-specified, 1-200)

5.2.6 MPI with Makefile Compilation

Uses a Makefile with MPI support:

```
export CC='mpicc -D_MPI=n'; make code.tst
```

- Sets the C compiler to `mpicc` with MPI support
- Requires a properly configured Makefile

5.2.7 MPI Portable Source Compilation

Creates portable MPI code:

```
qcc -source -D_MPI=n code.c && mpicc -Wall -std=c99 -O2 -D_MPI=n _code.c -o code
    -lm
```

- First generates standard C code with Basilisk extensions resolved
- Then compiles the generated code with MPI support

5.3 Dual-Mode Execution System

Version 1.0 introduces a dual-mode execution system that gives users flexibility in how they compile and run Basilisk code.

5.3.1 Emacs Mode Execution

In this mode, commands run within Emacs:

1. Press `C-c c` to invoke the compilation command generator.
2. Select a compilation method from the menu that appears.
3. If MPI is selected, enter the number of processes (1-200).
4. When prompted "Run in terminal instead of Emacs?", select "No".
5. The command executes via `bash -c "source  /.bashrc && ..."` in the `*compilation*` buffer.
6. Errors and warnings appear in the buffer with navigation support (`M-g n`, `M-g p`).

5.3.2 Terminal Mode Execution

For users who prefer working in a terminal or need specific environment settings:

1. Press `C-c c` to invoke the compilation command generator.
2. Select a compilation method and enter MPI processes if needed.
3. When prompted "Run in terminal instead of Emacs?", select "Yes".
4. The command is copied to the clipboard.
5. Paste the command into your terminal and execute it manually.

This dual-mode approach allows flexibility, especially when dealing with complex environment configurations or when the shell environment differs from Emacs.

5.4 Customizing Compilation Methods

The Basilisk Mode allows for customization of compilation methods:

5.4.1 Adding New Compilation Methods

You can add new compilation methods by customizing the `basilisk-compilation-methods` variable:

```
(setq basilisk-compilation-methods
      '(("Basic (No MPI)" . "qcc -Wall %s.c -o %s -lm")
        ("Optimized (No MPI)" . "qcc -Wall -O2 %s.c -o %s -lm")
        ;; Add your custom method here
        ("Custom Method" . "qcc -Wall -O3 -march=native %s.c -o %s -lm")))
```

5.4.2 Modifying Existing Methods

You can modify existing methods by reassigning the variable:

```
(setq basilisk-compilation-methods
      '(("Basic (No MPI)" . "qcc -Wall -Wextra %s.c -o %s -lm")
        ("Optimized (No MPI)" . "qcc -Wall -O3 %s.c -o %s -lm")
        ("MPI (4 processes)" . "CC99='mpicc -std=c99' qcc -Wall -D_MPI=4 %s.c -o
    %s -lm")))
```

5.5 Advanced Compilation Options

5.5.1 Compilation with Visualization Libraries

For compiling with visualization support (e.g., OpenGL, Glut):

```
qcc -Wall code.c -o code -lm -lGLU -lGL -lglut
```

5.5.2 Compilation with Profiling

For performance profiling:

```
qcc -Wall -pg code.c -o code -lm
```

- -pg: Adds profiling information for `gprof`

5.5.3 Cross-Compilation

For compiling for different architectures:

```
CC99='arm-linux-gnueabihf-gcc -std=c99' qcc -Wall code.c -o code -lm
```

Chapter 6

Execution Commands

The Basilisk Mode provides integrated tools for executing compiled Basilisk programs directly from Emacs.

6.1 Available Run Methods

The Basilisk Mode provides several methods for running compiled programs. As with compilation, these can be executed in either Emacs or Terminal mode.

6.1.1 Basic (No MPI) Execution

To run a compiled program without MPI:

```
./code
```

You can access this by pressing `C-c x` and selecting "Basic (No MPI)" from the menu.

6.1.2 Valgrind Execution

To run with Valgrind memory checking:

```
valgrind --leak-check=full ./code
```

Access this by selecting "Valgrind (No MPI)" from the `C-c x` menu.

6.1.3 MPI Execution

To run a program compiled with MPI support:

```
mpirun --oversubscribe -np n ./code
```

Where `n` is the number of MPI processes (1-200) entered when prompted.

6.1.4 MPI with Slurm Execution

For cluster environments with Slurm workload manager:

```
srun -n n ./code
```

Where `n` is the number of MPI processes to use.

6.1.5 Controlling Command-line Arguments

You can modify the execution command in the minibuffer before running:

```
./code arg1 arg2   # for basic execution
mpirun --oversubscribe -np 4 ./code --option=value   # for MPI
```

6.2 Combined Compilation and Execution

6.2.1 Compile and Run (C-c z)

To compile and immediately run a program:

1. Press `C-c z`
2. Select a compilation method
3. Select an execution method
4. If compiling with MPI, enter the number of processes (1-200)
5. Decide whether to run in Emacs or Terminal (respond to "Run in terminal instead of Emacs?")
6. The program will be compiled and, if successful, executed

This combines the compilation and execution steps into a single command, saving time during development iterations. The commands are combined with `&&` so execution only happens if compilation succeeds.

6.3 Output Handling

6.3.1 Viewing Program Output in Emacs Mode

When using Emacs mode execution, program output appears in the `*compilation*` buffer. You can:

- Navigate through the output using `M-g n` and `M-g p` to move between error messages
- Click on error messages to jump to the corresponding code line
- Save the output buffer to a file with `C-x C-w`
- Filter output with `M-x compilation-filter-select`
- Kill a long-running process with `C-c C-k` in the compilation buffer

6.3.2 Viewing Output in Terminal Mode

When using Terminal mode execution:

- Output appears directly in your terminal
- Use terminal features (scrollback, search) to navigate
- Pipe output to files or other programs as needed: `./code | tee output.txt`
- Control execution with terminal signals (Ctrl+C to interrupt, etc.)

6.3.3 Handling Long Outputs

For simulations with extensive output:

- Redirect output to a file: `./code > output.txt`
- Use `M-x auto-revert-tail-mode` to watch the output file as it grows
- Use Unix tools like `tail -f output.txt` in Terminal mode
- For very large outputs, consider using `logrotate` or similar tools

Chapter 7

Help and Documentation

The Basilisk Mode provides integrated access to help resources and documentation directly from Emacs.

7.1 Browsing Documentation

7.1.1 Accessing Documentation (C-c d)

Press `C-c d` to open a selection menu with documentation options:

- **Main**: Opens the main Basilisk website (`http://basilisk.fr/`)
- **Functions**: Opens the functions reference (`http://basilisk.fr/src/README`)
- **Examples**: Opens the examples page (`http://basilisk.fr/src/examples/README`)
- **Tutorial**: Opens the tutorial (`http://basilisk.fr/Tutorial`)

The selected documentation page will open in your default web browser.

7.2 Compilation Help

7.2.1 Displaying Compilation Help (C-c h)

Press `C-c h` to display a buffer with detailed compilation and running instructions. This help buffer includes:

- Explanation of different compilation methods
- MPI compilation options
- Optimization flags
- Running instructions for MPI and non-MPI programs
- Common compilation errors and solutions

7.3 Integrated Help for Keywords

7.3.1 Context-Sensitive Help

While editing code, you can get help on Basilisk keywords and functions:

- Position the cursor on a Basilisk keyword or function
- Press `C-c C-h` (or define this key if not available)
- If documentation is available, it will be displayed in a help buffer or opened in a browser

Chapter 8

Navigation and Refactoring

The Basilisk Mode enhances navigation and refactoring capabilities for Basilisk code.

8.1 Finding Event Definitions

8.1.1 Jumping to Event Definitions (C-c C-f)

To quickly navigate to an event definition:

1. Place the cursor on an event name in your code
2. Press `C-c C-f`
3. The cursor will jump to the definition of that event

This is particularly useful in larger files with many event definitions.

8.2 MPI Support

8.2.1 Toggling MPI Support (C-c C-t)

To enable or disable MPI support in your code:

1. Press `C-c C-t`
2. If MPI is not defined, a `#define _MPI 1` will be added to the buffer
3. If MPI is already defined, the definition will be commented out

8.3 Additional Navigation Features

8.3.1 Jumping Between Include Statements

Use standard Emacs commands enhanced for Basilisk:

- `C-M-f` and `C-M-b` to move by expressions
- `C-M-u` to move up to the containing parenthetical group
- `C-M-d` to move down into a nested parenthetical group

8.3.2 Function and Variable Navigation

The mode integrates with Emacs' built-in navigation:

- `M-.` with appropriate tags to jump to function definitions
- `M-*` to jump back
- `C-s` and `C-r` for incremental search

Chapter 9

Integration with Emacs

The Basilisk Mode seamlessly integrates with Emacs' ecosystem, providing a cohesive environment for Basilisk development.

9.1 Key Bindings

9.1.1 Complete Key Binding Reference

The following table lists all key bindings available in Basilisk Mode version 1.0:

Keybinding	**Function**	**Description**
`C-c e`	`basilisk-insert-event`	Insert an event block
`C-c f`	`basilisk-insert-foreach`	Insert a foreach loop
`C-c a`	`basilisk-insert-foreach-face`	Insert a foreach_face loop
`C-c r`	`basilisk-insert-foreach-reduction`	Insert a reduction loop
`C-c m`	`basilisk-insert-mpi-function`	Insert an MPI function
`C-c i`	`basilisk-insert-main`	Insert a basic main function
`C-c o`	`basilisk-insert-main-mpi`	Insert an MPI main function
`C-c s`	`basilisk-insert-struct`	Insert a struct definition
`C-c v`	`basilisk-insert-solver`	Insert a solver template
`C-c c`	`basilisk-compile-command-generator`	Compile with options
`C-c x`	`basilisk-run`	Run the compiled program
`C-c z`	`basilisk-compile-and-run`	Compile and run
`C-c d`	`basilisk-browse-documentation`	Open Basilisk documentation
`C-c h`	`basilisk-show-compilation-help`	Show compilation help
`C-c C-f`	`basilisk-find-event-definition`	Find event definition
`C-c C-t`	`basilisk-toggle-mpi`	Toggle MPI support

9.2 Menu Integration

9.2.1 Basilisk Menu

The mode adds a "Basilisk" menu to the Emacs menu bar with the following sections:

- **Templates**: Contains all code template insertion commands
- **Compilation**: Contains compilation and execution commands
- **Navigation**: Contains navigation and refactoring commands

- **Help**: Contains documentation and help commands

The detailed menu structure is:

- **Basilisk**
  - Insert Event Block
  - Insert Foreach Loop
  - Insert Foreach_face Loop
  - Insert Foreach Reduction
  - Insert MPI Function
  - Insert Main Function
  - Insert MPI Main Function
  - Insert Struct
  - Insert Solver Template
  - —
  - Compile with Options
  - Run Compiled Program
  - Compile and Run
  - Toggle MPI Support
  - —
  - Find Event Definition
  - —
  - Compilation Help
  - Browse Documentation

9.3 Indentation

9.3.1 Customized Indentation

The Basilisk Mode sets up Basilisk-friendly indentation with:

- 2-space indentation by default
- Special handling for Basilisk-specific constructs
- Proper alignment of `foreach` and `event` blocks

9.4 Compatibility with Other Emacs Packages

9.4.1 Company Mode

For autocompletion, Basilisk Mode works well with Company Mode:

```
(add-hook 'basilisk-mode-hook 'company-mode)
```

9.4.2 Flycheck

For on-the-fly syntax checking:

```
(add-hook 'basilisk-mode-hook 'flycheck-mode)
```

9.4.3 Yasnippet

For additional code templates:

```
(add-hook 'basilisk-mode-hook 'yas-minor-mode)
```

Basilisk Mode for Emacs

Chapter 10

Advanced Usage

10.1 Multi-File Projects

10.1.1 Working with Multiple Files

For larger Basilisk projects spanning multiple files:

- Use Emacs' built-in project management capabilities
- Consider `projectile` for enhanced project management
- Use `find-grep` or `rgrep` for searching across files

10.1.2 Building Complex Projects

For projects with multiple source files:

- Create a Makefile for the project
- Use `M-x compile` with `make` to build the project
- Use `make -j` for parallel builds

10.2 Performance Tuning

10.2.1 Profiling and Optimization

Use the Basilisk Mode's compilation features to:

- Compile with profiling flags (`-pg`)
- Analyze performance bottlenecks
- Recompile with higher optimization levels

10.3 Parallel Computing

10.3.1 Advanced MPI Usage

For complex parallel simulations:

- Use the MPI templates as a starting point
- Customize MPI process counts based on your hardware
- Add custom MPI communication patterns

Chapter 11

Troubleshooting

11.1 Installation Issues

11.1.1 Mode Not Activating

If the Basilisk Mode does not activate automatically:

- Ensure `(require 'basilisk_setup)` is in your init file
- Check that the file is in your `load-path`
- Try loading manually with `M-x load-file`

11.1.2 Parsing Errors

If you see "End of file during parsing" errors:

- Run `emacs -debug-init` to get a backtrace
- Check for balanced parentheses and quotes in `basilisk_setup.el`
- Verify the file ends with `(provide 'basilisk_setup)` and a newline

11.2 Compilation Problems

11.2.1 Compilation Fails

If compilation commands fail:

- Ensure `qcc` and `mpicc` (if using MPI) are in your PATH
- Check the `*compilation*` buffer for specific errors
- Try running the compilation command directly in a terminal

11.2.2 Execution Problems

If the program compiles but does not run correctly:

- Check for runtime dependencies
- Verify MPI is correctly installed (for MPI programs)
- Look for error messages in the compilation buffer

11.3 Feature Issues

11.3.1 Template Insertion Problems

If code templates don't insert correctly:

- Check the key binding to ensure it's not overridden by another mode
- Try inserting the template through the Basilisk menu
- Restart Emacs to ensure the mode is correctly loaded

11.3.2 Documentation Access Issues

If documentation links don't work:

- Check your internet connection
- Verify that your default browser is correctly set
- Try accessing the Basilisk website directly

Chapter 12

Customization

12.1 Customizing the Mode

12.1.1 Adding Keywords

To add custom keywords for syntax highlighting:

```
(defvar basilisk-additional-keywords
  '("custom_keyword1" "custom_keyword2"))

(font-lock-add-keywords 'c-mode
  `((,(regexp-opt basilisk-additional-keywords 'words)
     . font-lock-keyword-face)))
```

12.1.2 Modifying Templates

To modify or add code templates:

```
(defun basilisk-insert-custom-template ()
  "Insert a custom template."
  (interactive)
  (insert "// My custom template\n")
  (insert "custom code here;"))

(define-key c-mode-map (kbd "C-c y") 'basilisk-insert-custom-template)
```

12.2 Configuration Options

12.2.1 Common Configuration Variables

Customize these variables to change the mode's behavior:

```
;; Change indentation width
(setq-default c-basic-offset 2)

;; Customize compilation methods
(setq basilisk-compilation-methods
  '(("Custom" . "qcc -Wall -O3 %s.c -o %s -lm")))

;; Customize documentation URLs
(setq basilisk-documentation-urls
  '(("Custom Doc" . "http://example.com/doc")))
```

Chapter 13

Appendices

13.1 Appendix A: Basilisk Syntax Reference

13.1.1 Event Syntax

```
event name ([condition]) {
  // Code executed when condition is met
}
```

Common conditions:

- `t = 0`: At time zero (initialization)
- `t += dt`: Every time step
- `t += 0.1`: Every 0.1 time units
- `i++`: Every iteration

13.1.2 Foreach Syntax

```
foreach() {
  // Code executed for each cell
}
```

```
foreach_face(x) {
  // Code executed for each face in x direction
}
```

13.1.3 Reduction Syntax

```
double max_val = 0;
foreach (reduction(max:max_val)) {
  if (T[] > max_val)
    max_val = T[];
}
```

13.2 Appendix B: Common Basilisk Functions

13.2.1 Grid Functions

- `init_grid(n)`: Initialize grid with 2^n cells
- `adapt_wavelet(fields, error, level)`: Adapt grid based on wavelet error
- `boundary(fields)`: Apply boundary conditions

13.2.2 Solver Functions

- `solve(eq)`: Solve a linear system
- `advection(fields, velocity, dt)`: Advect fields
- `diffusion(field, dt, diffusivity)`: Apply diffusion

13.3 Appendix C: Key Basilisk Includes

13.3.1 Common Include Files

- `grid/octree.h`: Basic grid functionality
- `navier-stokes/centered.h`: Centered Navier-Stokes solver
- `tracer.h`: Passive tracer functionality
- `view.h`: Visualization functions

Bibliography

[1] Basilisk Official Website, http://basilisk.fr/

[2] GNU Emacs Manual, https://www.gnu.org/software/emacs/manual/

[3] MPI: A Message-Passing Interface Standard, https://www.mpi-forum.org/

[4] Bash Reference Manual, https://www.gnu.org/software/bash/manual/

[5] GNU Make Manual, https://www.gnu.org/software/make/manual/

Version History

- **Version 1.0 (February 2025)**
  - Added shell interpreter integration using `bash -c "source  /.bashrc && ..."`
  - Implemented dual-mode execution (Emacs and Terminal)
  - Fixed filename extension handling for proper `.c` reference
  - Enhanced error detection and navigation
  - Added clipboard support for Terminal mode
- **Version 0.1 (January 2025)**
  - Initial comprehensive release
  - Basic syntax highlighting for Basilisk
  - Code templates and snippets
  - Compilation and execution support
  - Documentation and navigation features