EiffelStudio – the Eiffel IDE

Christian Estler
ETH Zurich
christian.estler@inf.ethz.ch

Distributed and Outsourced Software Engineering - ETH course, Fall 2012
EiffelStudio – an Overview

• EiffelStudio (ES) is an Integrated Development Environment to write Eiffel programs
  • Developed by Eiffel Software
  • First version ca. 1990
  • Current version is 7.1
  • Available on SourceForge

• We use ES in the course

• Many things are similar to IDEs like Eclipse or VisualStudio; some things are different or tricky → that’s what we’ll talk about here
EiffelStudio – an Overview

We will talk about

• *Clean compile* (what is it, why is it needed?)
• Code Browsing
• Code Views
• EiffelStudio’s navigation using *Pick & Drop*
• Debugging in EiffelStudio
• Auto-completion and Code-Templates
• Refactoring tools
Clean Compile

- **Clean compile** means compiling the project from scratch

- Necessary e.g. after adding or removing libraries

- Deletes previously generated intermediate compile information (the EIFGENs folder)

- Rule of thumb: if your program shows strange behavior, do a clean compile

If you remember only one thing from this talk, it should be **clean compile**

Code Browsing

- ES has many features for browsing code; you’ll often use the following:
  - Group & Feature View
  - Class tool to determine
    - Ancestors
    - Descendants
    - Clients
  - Feature tool to determine
    - Flat view
    - Callers
    - Callees
    - ...
  - Navigation through *Pick & Drop* (ES way of doing Drag & Drop)
Code Browsing – the Basics

- **Group View**

- **Feature View**

Tip 1: arrange Views such that both are visible all the time. You’ll use them a lot.

Tip 2: Pick & Drop also works with these Views. Alternative to double-click.
Code Browsing – the Class Tool

• Eiffel makes have use of (multiple) inheritance
• Class tool provides all information on the class level
Feature tool provides all information of a feature

```plaintext
initialize
   -- Build the window
   -- (export status {NONE})
require -- from EV_ANY
    not_already_initialized: not is_initialized
do
   Precursor {EV_TITLED_WINDOW}
close_request_actions.extend (agent request_close_window)
create con_main
put (con_main)
create con_game
con_game.set_background_pixmap (Pix_background_sepia)
con_main.extend_with_position_and_size (con_game, 200, 0, 780, 780)
create btn_start.make_with_text ("Start Game")
btn_start.select_actions.extend (agent initialize_game_board (False))
con_main.extend_with_position_and_size (btn_start, 10, 600, 180, 50)
create btn_quit.make_with_text ("Quit Game")
btn_quit.select_actions.extend (agent request_close_window)
```

Tip: Pick & Drop also works within the Class and Feature tools (e.g. pick something inside the Feature tool and drop it right there)
Code Browsing – the Diagram Tool

- Diagram tool can be used to a high-level overview of the entire system (more than Class and Feature tool)

- Pick & Drop a cluster on the diagram target icon
You can inspect code in different views:

**Editable views**

| Basic text view | Default editor view, used for writing code |

**Non-editable views**

| Clickable view | Reformatted representation of the code; more clickable elements (e.g. comments) than in basic text view |
| Flat view | Shows the flattened version of a class (e.g. all inherited features); this is the view used by the debugger |
| Contract View | Public interface of the class, incl. contracts |
| Interface view | Same as contract view but for the flat-version of the class. |

**Code Browsing – the Addressbar**

- **Addressbar** can be used to quickly open classes or features

![Addressbar Example](image.png)

- If you only remember part of a class or feature name, use “*” in the search, e.g.
  - Search class:  *TTT*  → all classes starting with TTT
  - Search feature:  *  → all features of the current class
Tip for Pick & Drop

• Many ES tools work with *Pick & Drop*
  - Tip: try to drop elements on various kinds of icons in ES

• *Pick & Drop* feels “slow” if you go through the right-click context menu

• Make right-click the default for Pick & Drop:
  
  *Tools* -> *Preferences* -> *General*. *Pick and drop*
Tip for Pick & Drop

• Option 1:
  • Shift + right-click: Pick
  • Ctrl + right-click: Open element in a new tab in editor

• Option 2:
  • Make right-click the default for Pick & Drop:

  *Tools* -> *Preferences* -> *General.Pick and drop*
• **Compiling a system (F7)**

• *Melting*: Generates bytecode, not C code. Quick to generate but slowest execution. Use during development.

• **Freezing**: Generates C code for the whole system. Compilation takes longer but system executes faster. Can still be debugged. Use during development.

• **Finalizing**: Creates an executable production version. Finalization performs extensive time and space optimizations. Cannot be debugged.
Running a System

• Run a system by clicking “run” (F5)

  Run stepwise

• Switches to “execution mode”
  • Shows more debugging related tools
  • Shows controls for system execution (stop, pause, etc.)
Debugger

- Debugging works for *melted* and frozen *systems* (not for finalized ones)

- **Breakpoints** can only be added using a *flat view*
  - One way: switch editor view to flat view
  - Quicker: Pick & Drop feature into Feature tool
Debugger

- To start the debugger simply hit “Run” (no distinction like in Eclipse)

- Usual debugger tools are available during debugging
  - Call Stack
  - Expressions
  - Switching between threads

- Usual debugger controls are available
  - One step at a time (F10)
  - Step into a routine value (F11)
  - Step out of a routine (Shift + F11)
Debugger

- Often useful: **conditional breakpoints**
  - Execution will only be stopped under certain conditions.
Tip for debugging a client/server system

• Goal: run server and client on same machine

• Rather than using command line, you can do:
  • Run the server
  • *Detach* the server instance
  • ES returns to “edit mode”
  • Run the client
Auto-completion

- ES has auto-completion

Auto-completion knows (only) about compiled code

If it does not work (as you would expect), try the following

- Try to compile the system
- Close and reopen the file in the editor
- Do a clean compile
Auto-completion

- " * " can also be used in auto-completion
- Example:
  - Find all calls containing “set”
  - Use `my_target_name.*set`
Auto-completion – Word vs. Class

We have two types of completion

1. **Word-completion (Ctrl + Space)**

2. **Class name completion (Ctrl + Shift + Space)**
Code Templates

ES comes with a number of code templates

- write a keyword
- hit enter
- subsequent keywords are filled in automatically

Examples

- do .. end
- from ... until ... loop ... end
- across ... as ... loop ... end

Special case → across loop + hitting space rather than enter

- across ... as ... all ... end
Refactoring Tools

- EiffelStudio only supports two refactorings:
  - Renaming
  - Pull Up routine

- Works only on compiling system
Other useful stuff

• Take a look at menu
  • Edit
  • Edit → Advanced

• Make use of
  • Line numbers
  • Pretty print
  • Commenting
  • …

• Learn some of keyboard shortcuts 😊
Things we ignore for the moment

• Project settings
  • Shared with all other teams
  • Thus you should not modify them

• Profiler:
  http://docs.eiffel.com/book/eiffelstudio/profiling

• Record Replay:
Further Resources

• Official EiffelSoftware websites:
  • http://www.eiffel.com/
  • http://dev.eiffel.com/
  • http://docs.eiffel.com/

• Have a look in the DOSE wiki

• Make use of the Eiffel Mailing list:
  • http://tech.groups.yahoo.com/group/eiffel_software/
  • Search the archive for answers
  • Feel free to post any kind of (Eiffel-related) question
THE END