BPC User Guide

Application Development with BPC

Last Updated 25/04/2014

Contents

Setup

Requirements

- Presently the BPC package is aimed for Linux machines only
- It was tested on the Ubuntu 10.4 distribution
- The machine should have g++ installed. For the Ubuntu distribution, “sudo apt-get install build-essential” should suffice
- Compilation is performed using Makefiles
- Editing is not limited to any specific IDE – use the one you like best

Installation

- To install the BPC core, simply decompress the .tar.gz file from the site
- To see that it compiles properly, run the “make” command inside the root folder

A New Project

- We recommend using the sample project provided with the package as a skeleton – simply rename the folder
- Any BPC project will use the core. To tell your project where it is, override the Makefile variable CORE_DIR in your Rules.mk file (or by an argument)

Compiling Your Project

- If you maintain the structure proposed in the sample application (i.e. recursive Makefiles), you will only need to run the “make” command in the root directory of your project
- The result is an .elf file, whose name is defined in your Makefiles, that can be run. Note that the core is integrated into the project, and does not need to be run separately.

Core Directories and Tests

- The core/common directory contains multiple utility classes. Feel free to use them
- The core itself was written using test driven development. Consequently, it comes with a bunch of unit-tests, inside the /tests subdirectory of every main directory
  - The unit-testing framework is called CxxTest. It comes with the core (tools/cxxtest).
- These unit tests are automatically run with every compilation. If you change the core, you might have to adjust them as well
- If you don’t care about unit-tests and just want to look at the core’s code, ignore the “T::” prefix everywhere.
- If you want to make core contributions, make sure to write tested code

Writing a Program

Program Structure

- A BPC program uses key classes defined in the core. Most notably:
  - *BProgram*: the main class. Instantiate it, register your threads to it using the *addThread* method, and then run it using the *runProgram* method
  - *BThread:* the thread interface class. Any thread must inherit from it, and implement its *entryPoint* method. It provides the *bSync* and *lastEvent* methods
  - *Event:* a class representing events, vectors of which are passed to the *bSync* method. Currently, an event is just an event code (an integer), without extra parameters

bSync

- The bSync method takes as input three *vectors* of events
  - *Vectors* are defined in the core/common directory; see in the sample application
- Just add the relevant events to the vectors
- Vectors may also be passed empty

IEvent

- We recommend, though this is not mandatory, to keep an IEvent file as in the sample application
- This file enumerates the events
- It also provides a *allEvents* method, which is convenient.
  - You may extend it with similar methods

The Core and the Threads

- At run time, each b-thread is run in a separate thread
- They communicate with the core, also run in a separate thread
- Communication is socket based
  - See the engine/EngineConfiguration.cpp file for determining the port.
- The system supports dynamic addition of new threads during run time, as well as termination of existing threads