Tekkotsu Quick Reference

ERS-7, Tekkotsu 3.0

Ethan Tira-Thompson

1 — Building

1.1 Environment Variables

Values can be set in `project/Environment.conf`, a shell environment variable, or the `make` command line. Each configuration method overrides values from prior methods.

- `OPENRSDK_ROOT` - Location of OPEN-R SDK installation (default `/usr/local/OPEN_R_SDK`)
- `TEKKOTSU_ROOT` - Location of Tekkotsu installation (default `/usr/local/Tekkotsu`)
- `MEMSTICK_ROOT` - Location of memory stick mount point (default tries to autodetect – Mac OS X users should get something in `/Volumes/...`, Cygwin users will want something like `/cygdrive/e`, Linux defaults to `/mnt/memstick`)
- `TEKKOTSU_TARGET_PLATFORM` - Can be set to either `PLATFORM_APERIOS` (the Aibo's OS, default) or `PLATFORM_LOCAL` (host machine's OS, local execution).
- `TEKKOTSU_TARGET_MODEL` - Can be set to one of the supported robot models: `TGT_ERS210`, `TGT_ERS220`, `TGT_ERS2xx` (dual boot either 210 or 220), `TGT_ERS7` (default).
- `TEKKOTSU_ALWAYS_BUILD` - If any non-empty string (default), update `libtekkotsu.a` prior to each project build. Turn off if `TEKKOTSU_ROOT` is in a read-only location, such as a shared directory in a computer lab.

See also additional options and documentation in `project/Environment.conf`

1.2 Build Targets

The following targets are available as targets to the `make` command, from within a project directory:

- `all` - Builds source into binary executables.
- `newstick` - Erases memory stick, copies fresh system files
- `install` - Builds `all`, copies project's `ms` directory to memory stick (may require previous `make newstick`)
- `update` - Builds `all`, selectively copies newer files from `ms` to memory stick (requires rsync)
- `sim` - Builds the project for local simulation, equivalent to `all` with `TEKKOTSU_TARGET_PLATFORM` set to `PLATFORM_LOCAL`
- `clean, cleanProj, cleanDeps` - Erases all, just project, or dependancy (.d) build files

The following targets are available to `make` from within the main Tekkotsu directory:

- `all` - Updates [build/.../]`libtekkotsu.a` for each supported platform
- `compile` - Updates [build/.../]`libtekkotsu.a` for only the current platform (see `TEKKOTSU_TARGET_PLATFORM`)

2 — Execution

2.1 Console

You can view text output from code running on the AIBO by using a telnet connection. You can filter the output you wish to receive by selecting a port number below. Any output sent to a port which is unconnected will be redirected to the one listed below it.

- `10002` - Output from Main process's `serr` (e.g. `serr->printf(...)`). Theoretically "blocking", but some output can still be lost in the system network buffers during a crash. Input from client is ignored.
- `10001` - Output from Main process's `sout`. Non-blocking for fast thoroughput, but will lose most recent data sent if crash occurs. Input is interpreted as ControllerGUI commands if no GUI is connected, otherwise each line of input is broadcast as a TextMsgEvent.
- `59000` - The system output console, catches all output from the operating system itself, `cout`, `cerr` or basic `printf`. Non-blocking, buffered output, *blocking input* (don't use `cin`).

2.2 Controller

You can send these commands to port 10001 (assuming tekkotsu.cfg's main.consoleMode is the default value), the GUI port 10020 (if GUI not connected), or the GUI's scripts or "Send Input" field. All commands start with '`!`'. The input field will prefix non-commands with `!input`. Use quotes around multi-word arguments.

- `!reset` - return to the root menu
- `!next` - hilight the next menu item
- `!prev` - hilight the previous menu item
- `!select` [*item*] - trigger activate of currently hilighted menu items, unless *item* is specified, in which case Controller will search from the root for an entry of the same name and trigger it if found
- `!cancel` - move up/back a menu level
- `!msg` *text* - broadcasts *text* as a TextMsgEvent
- `!root` *item* [*subitem* ...] - triggers indicated items to drill down from root to a specified subitem. Doesn't change current menu location unless final item is a menu itself.
- `!hilight` [*n1* [*n2* ...]] - Selects listed item indicies, can select multiple at once
- `!input` *text* - Passes *text* as input to the currently hilighted item(s), can be used to move directly to a submenu, specify a filename to save data into, set a corresponding variable, etc., depending on the context.
- `!set` *section.key=value* - sets a configuration variable, overriding the default found in `tekkotsu.cfg` for remainder of the current session only

3 — Model Specification

Each supported robot model is specified by an "Info" header file in `Shared`, e.g. `Shared/ERS7Info.h`, which defines the constants shown in this section. These files can also alias symbols found on other models to aid in portability between models. `Shared/RobotInfo.h` should be used to automatically include the proper header for the current target model and bring its constants into the global namespace.

3.1 Output Offsets

"Outputs" (i.e. Joints, LEDs) are refered to by index ("offset") value. These values are formed by specifying a *section* offset, plus a *specific* offset. Sections are typically general across robot models, whereas the specifics are model-dependent (but can be aliased to provide compatability).

For most joints, the positive direction is "up", and the 0 position yields a forward looking, fully extended standing posture.

- {L,R}{Fr,Bk}`LegOffset` - `NumLegs` combinations, each with `JointsPerLeg` items
  - `RotatorOffset`: positive moves "out", away from body
  - `ElevatorOffset`: positive moves up, away from body
  - `KneeOffset`: positive bends knee (slight negative possible)
- `HeadOffset` - `NumHeadJoints` items
  - `TiltOffset`: positive looks up
  - `PanOffset`: positive looks left
  - `NodOffset`: positive looks up
- `TailOffset` - `NumTailJoints` items
  - `TiltOffset`: positive raises the joint (lowers the tail itself)
  - `PanOffset`: positive points the tail to the Aibo's right
- `MouthOffset` - `NumMouthJoints` items (no specific)
- LEDs: these are all direct offsets, and do not need to be added to anything else
  - `HeadColorLEDOffset`, `HeadWhiteLEDOffset`, `ModeRedLEDOffset`, `ModeGreenLEDOffset`, `ModeBlueLEDOffset`, `WirelessLEDOffset`, `FrBackColorLEDOffset` (blue), `FrBackWhiteLEDOffset`, `MdBackColorLEDOffset` (orange), `MdBackWhiteLEDOffset`, `RrBackColorLEDOffset` (red), `RrBackWhiteLEDOffset`
  - `FaceLEDPanelOffset` - 14 items, individually unnamed. See Sony's model documentation for panel diagram.
  - `LEDABModeOffset` - this is a "virtual" LED, which changes how the AIBO interprets certain face panel LEDs
- `EarOffset` - `NumEarJoints` items

It happens that these joints can also be grouped by the *type* of joint, so there are additionally a few other offsets that can be used in order to loop across a group of joints:

- `PIDJointOffset` - `NumPIDJoints` items, servos using PID control
- `LegOffset` - `NumLegJoints` items, a subset of PID servos corresponding to the leg joints
- `LEDOffset` - `NumLEDs` items
- `BinJointOffset` - `NumBinJoints` items, solenoids, such as the ears (if any) which flip between two positions
- `NumOutputs` - total number of outputs available

LEDs are often handled in groups to display patterns. Some functions take an `LEDBitMask_t` parameter, which allows you to specify a set of LEDs in a single parameter. For any given LED offset *foo*`LEDOffset`, the corresponding bitmask constant is *foo*`LEDMask`. Alternatively, you could calculate the bitmask of *foo* by `1<<(`*foo*`-LEDOffset)`.

3.2 Reference Frames

Every PID joint has an associated reference frame, with the z axis along the axis of rotation. You can use the Output Offsets previously listed to refer to these reference frames. However, there are a few additional reference frames you may wish to refer, which do not correspond to any particular joint:

- `BaseFrameOffset` - body center, x forward, y Aibo's left, z up
- `PawFrameOffset` - `NumLegs` items, in the usual `LegOrder_t`, defines the reference frame of the passive ankle joint
- `CameraFrameOffset` - The coordinate system used for vision, origin in the center of the image, x right, y down, z into the scene
- `NearIRFrameOffset, FarIRFrameOffset, ChestIRFrameOffset` - z axis aligned with each IR beam

3.3 Buttons

Primarily used with WorldState, i.e. `state->buttons[`x`]`. Buttons marked with a * are "pressure" sensitive (range 0-1), the rest are boolean.

`LFrPawOffset`	`ChinButOffset`	`FrontBackButOffset*`
`RFrPawOffset`	`HeadButOffset*`	`MiddleBackButOffset*`
`LBkPawOffset`	`WirelessSwOffset`	`RearBackButOffset*`
`RBkPawOffset`		

3.4 Sensors

Primarily used with WorldState, i.e. `state->sensors[`x`]`

`NearIRDistOffset`	`BAccelOffset`	`PowerRemainOffset`
`FarIRDistOffset`	`LAccelOffset`	`PowerThermoOffset`
`ChestIRDistOffset`	`DAccelOffset`	`PowerCapacityOffset`
		`PowerVoltageOffset`,
		`PowerCurrentOffset`

3.5 Limits

`outputRanges` defines the range of values available for each output. `MaxOutputSpeed` defines the maximum recommended speed for each output (0 indicates no limit, e.g LEDs).

4 — Development

4.1 Events

These constants are defined by the `EventBase::EventGeneratorID_t` enumeration. For more information about the events sent by each generator, see `EventBase` documentation. Each of these constants will need to be prefixed with `EventBase::`.

`unknownEGID`,	`micPitchEGID`,	`visRawCameraEGID`,
`aiEGID`,	`motmanEGID`,	`visInterleaveEGID`,
`audioEGID`,	`pilotEGID`,	`visJPEGEGID`,
`buttonEGID`,	`powerEGID`,	`visPNGEGID`,
`erouterEGID`,	`sensorEGID`,	`visSegmentEGID`,
`estopEGID`,	`stateMachineEGID`,	`visRLEEGID`,
`locomotionEGID`,	`stateSignalEGID`,	`visRegionEGID`,
`lookoutEGID`,	`stateTransitionEGID`,	`visObjEGID`,
`mapbuilderEGID`,	`textmsgEGID`,	`wmVarEGID`,
`micOSndEGID`,	`timerEGID`,	`worldModelEGID`,
`micRawEGID`,	`visOFbkEGID`,	
`micFFTEGID`,		

The value and meaning of the event's 'source' field is defined by its generator, but the 'type' field is restricted to one of `activateETID`, `statusETID`, or `deactivateETID`.