QProcess

```
// Making a QProcess object
QProcess process;
// Common methods of the object and what they do
proces s.s etS tan dar dIn put Fil e(Q Pro ces s:: nul lDe vic e()); // redirect stdin to /dev/null
proces s.s etS tan dar dOu tpu tFi le( QPr oce ss: :nu llD evi ce()); // redirect stdout to /dev/null
proces s.s tar t(e xec uta ble Name, argsList); // launch the process; argsList is a QStrin gList where
                                                                                                    -
   // each entry is one discrete argument — QProcess passes
                                                                                                    -
   // them blindly to the execut able, so the order and
                                                                                                    -
   // content is dictated by the execut able, not QProcess
proces s.e rror(); // returns error code e.g. QProce ss: :Fa ile dTo Start
proces s.k ill(); // force kill the child process
proces s.e xit Sta tus(); // NormalExit or CrashExit
proces s.e xit Code(); // 0 on success
proces s.r ead All Sta nda rdE rror(); // grab error output
// Holds the thread until either the process finishes or if it hits the duration provided in Ms as a
parameter
// Returns true if the process finished before the provided duration ends
// Returns false if the process didn't finished before the provided duration
// Also returns false if the process was already done before this line
proces s.w ait For Fin ish ed( tim eou tMs);
```

A Qt class that launches and manages external processes from within your application, used wherever you would use fork/exec in plain C++. It handles starting the process, managing its input/output streams, waiting for it to finish, and inspecting the result. It is preferred over raw fork/exec in Qt codebases because it cooperates with Qt's own internal signal handling.

QProcess::nullDevice()

```
QProcess::nullDevice()
```

A static method on QProcess that returns the path to the null device (/dev/null on Linux), which discards anything written to it and returns nothing when read. It is used to silence stdin and stdout when you only care about stderr. Being a static method it is called on the class itself rather than an instance.

QProcess::FailedToStart

```
if (process.error() == QProcess::FailedToStart) {
       // executable not found or not runnable
}
```

An error code on QProcess that indicates the executable could not be launched at all, typically because it was not found on the system's PATH or the user lacks permission to run it. It is checked via process.error() after waitForFinished() returns false to distinguish between a launch failure and a timeout. Being a static value it is referenced on the class itself rather than an instance.

QProcess::NormalExit

```
if (process.exitStatus() == QProcess::NormalExit) {
       // process exited cleanly, exit code can be trusted
}
```

A status code on QProcess that indicates the process exited cleanly on its own, as opposed to being killed by a signal or crashing. It is checked via process.exitStatus() after waitForFinished() returns to confirm the process shut down normally before trusting the exit code. Being a static value it is referenced on the class itself rather than an instance.

QProcess::NotRunning

```
if (process.state() == QProcess::NotRunning) // process has exited
```

A value of the QProcess::ProcessState enum indicating the process is not currently running — either it hasn't started yet or it has already finished. Check it via .state() to detect that a process exited between polling slices.