TaskTree ’24

Generic solution for automatic management of asynchronous tasks (C++)

Jarek Kobus / The Qt Company

Asynchronous Task

Any task that can be started, and **later** it finishes with success or an error.

Later → after starting the task, the control goes back to the running event loop.

We don’t block the caller thread until the task is finished.

In order to use the task tree we need an event loop spinning.

Examples of Asynchronous Tasks

- `QTimer::singleShot()`
- `QProcess`
- `QNetworkAccessManager + QNetworkReply = NetworkQuery`
- `QtConcurrent::run() + QFutureWatcher<Result> = ConcurrentCall<Result>`

Recipe & Task Tree

Analogy: Cartridge and Player

Recipe (cartridge):

- What tasks are to be created
- In which order
- How to setup each task before start
- Collect data when each task is finished
- Execution mode (sequential or in parallel)
- Workflow policy
- More...

Task tree (player):

- Reads the recipe and creates tasks automatically
- Manages the lifetime of created tasks
- Executes continuations
- Chooses different paths depending on results of finished tasks and workflow policy
- Provides progress info
- More...

Task Adapters & Custom Tasks

Task adapter: e.g. `QTimerTaskAdapter : public TaskAdapter<QTimer>`

Access to the task by `task()` method.

Reimplement `start()` method to start a task.

Emit `done(DoneResult)` signal when the task is finished.

Custom task: e.g. `using QTimerTask = CustomTask<QTimerTaskAdapter>;`

Custom task	Asynchronous task
`ConcurrentCallTask<ResultType>`	`ConcurrentCall<ResultType>`
`NetworkQueryTask`	`NetworkQuery`
`QProcessTask`	`QProcess`
`TaskTreeTask`	`TaskTree`

Example

```
const Group recipe {
    sequential,  // execution mode, the default
    stopOnError, // workflow policy, the default
    NetworkQueryTask(...),
    Group {
        parallel,
        ConcurrentCallTask<QImage>(...),
        ConcurrentCallTask<QImage>(...)
    }
};

taskTree->setRecipe(recipe);
taskTree->start();
```

Custom task / group handlers

The constructor of custom task takes up to 2 arguments.

```
const auto onDownloadSetup = [](NetworkQuery &task) { ... };
const auto onDownloadDone  = [](const NetworkQuery &task, DoneWith result)
                               { ... };
const auto onScaleSetup = [](ConcurrentCall<QImage> &task) { ... };
const auto onScaleDone  = [](const ConcurrentCall<QImage> &task, DoneWith result)
                             { ... };
const Group recipe {
    onGroupSetup([] { ... }),
    NetworkQueryTask(onDownloadSetup, onDownloadDone),
    ConcurrentCallTask<QImage>(onScaleSetup, onScaleDone),
    onGroupDone([](DoneWith result) { ... })
};
```

Inter-task data exchange

`Storage` serves for inter-task data exchange.

```
const Storage<QByteArray> storage;
const auto onDownloadDone  = [storage](const NetworkQuery &query) {
    *storage = query.reply()->readAll();
};
const auto onScaleSetup = [storage](ConcurrentCall<QImage> &query) {
    query.setConcurrentCallData(&scale, *storage);
};

const Group recipe {
    storage,
    NetworkQueryTask(..., onDownloadDone, CallDoneIf::Success),
    ConcurrentCallTask<QImage>(onScaleSetup, ...)
};
```

Logical Operators
(new feature)

```
const Group recipe {
    task1 && task2, // short-circuiting conjunction
    task3 || task4, // short-circuiting disjunction
    !task // negation
};
```

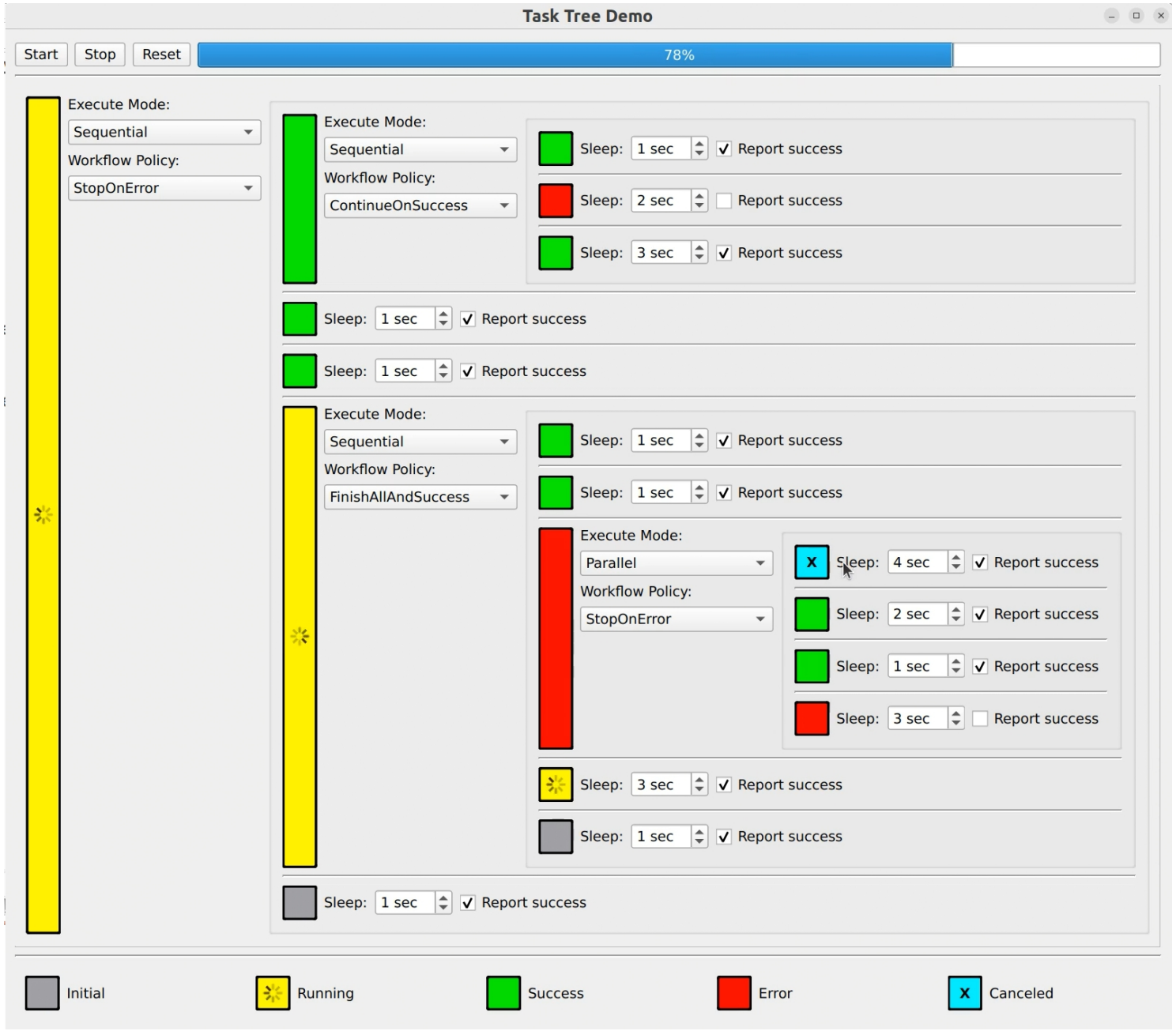

Conditional Expressions
(new feature)

```
const Group recipe {
    If (condTask1) >> Then {
        bodyTask1
    } >> ElseIf (condTask2) >> Then {
        bodyTask2
    } >> Else {
        bodyTask3
    }
};
```

For Loops
(new feature)

```
const QList<QByteArray> list = ... ;
const LoopList iterator(list);
const auto onSetup = [iterator](ConcurrentCall<QImage> &task) {
    const QByteArray currentData = *iterator;
    ...
};
const auto onDone = [iterator](const ConcurrentCall<QImage> &task) {
    qDebug() << ”current iteration” << iterator.iteration();
    ...
};
const Group recipe {
    For {
        iterator,
        ConcurrentCallTask<QImage>(onSetup, onDone)
    }
};
```

TaskTree demo...



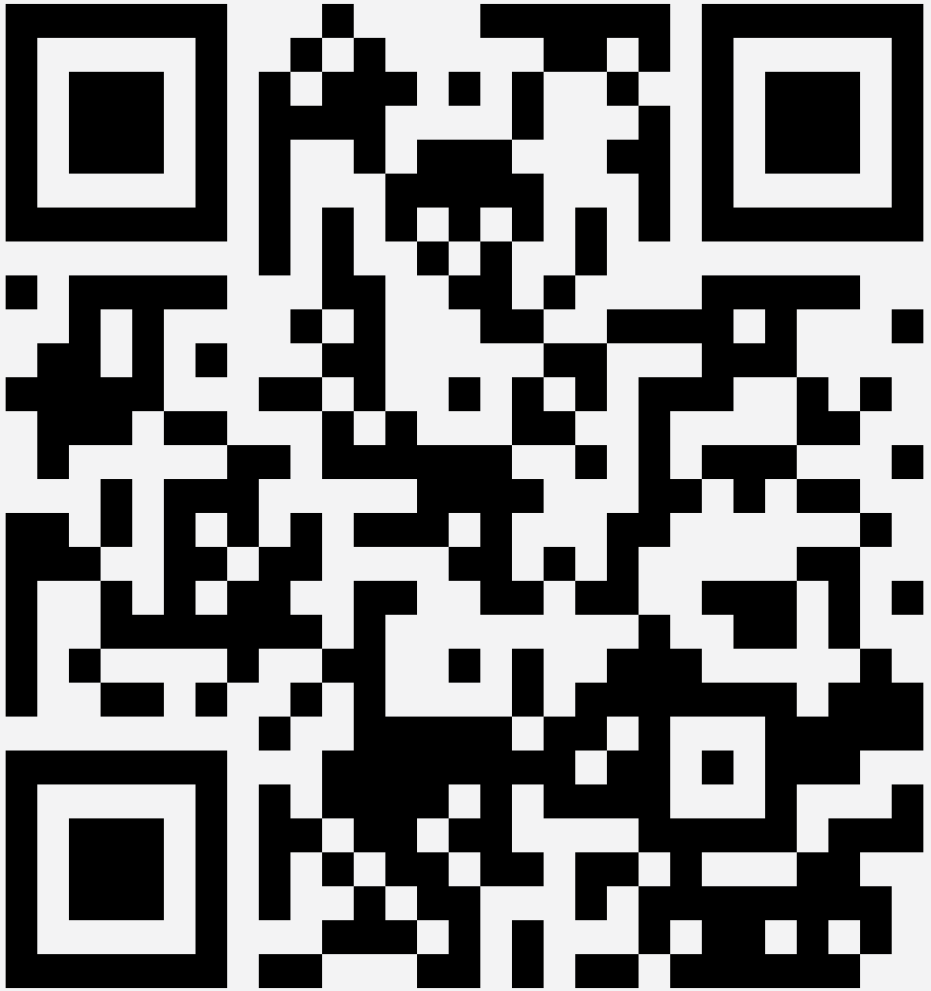
TaskTree

Questions?

Thank you!

Jarek Kobus / The Qt Company

More info:



https://wiki.qt.io/TaskTree