

TaskTree ’26

Generic and declarative 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:

```
QTimer::singleShot()
QProcess
QNetworkReply + QNetworkAccessManager = QNetworkReplyWrapper
QtConcurrent::run() + QFutureWatcher<Result> = QThreadFunction<Result>
```


© The Qt Company

Recipe & Task Tree

Analogy: Cartridge and Player

Recipe (Cartridge)	QTaskTree (Player)
What tasks are to be executed	Creates and destroys tasks, taking care of their lifetime
In which order	Respects the order of task execution defined by the recipe
Execution mode (sequential or parallel)	Executes tasks in sequence or in parallel
Workflow policy	Automatically controls the execution according to the policy
Nested groups	Executes a group of subtasks
Setup and done handlers	Executes handlers to allow custom setup and collecting results
Storage<DataType>	Dynamically instantiates DataType for inter-task data exchange
Conditional expressions	Chooses different code paths depending on async task result
For loops	Iterates over containers executing a given branch

© The Qt Company

Custom Tasks & Task Adapters

Create an alias to the custom task and specify the task type and its adapter:
`using QTimerTask = QCustomTask<QTimer, QTimerTaskAdapter>`.

Use `QTimerTask` in recipes (no `QTimer` instantiated, yet!).

When running task tree sees the `QTimerTask` in a recipe, it instantiates the `QTimer` dynamically.

Custom task	Asynchronous task
`QThreadFunctionTask<ResultType>`	`QThreadFunction<ResultType>`
`QNetworkReplyWrapperTask`	`QNetworkReplyWrapper`
`QProcessTask`	`QProcess`
`QTaskTreeTask`	`QTaskTree`

© The Qt Company

Example

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

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


© The Qt Company

Inter-task data exchange

`Storage` serves for inter-task data exchange.

```
const Storage<QByteArray> storage;

const auto onDownloadDone  = [storage](const QNetworkReplyWrapper &task) {
    *storage = task.reply()->readAll();
};
const auto onScaleSetup = [storage](QThreadFunction<QImage> &task) {
    task.setThreadDataFunction(&scale, *storage);
};

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


© The Qt Company

Logical Operators, Conditional Expressions, For Loops

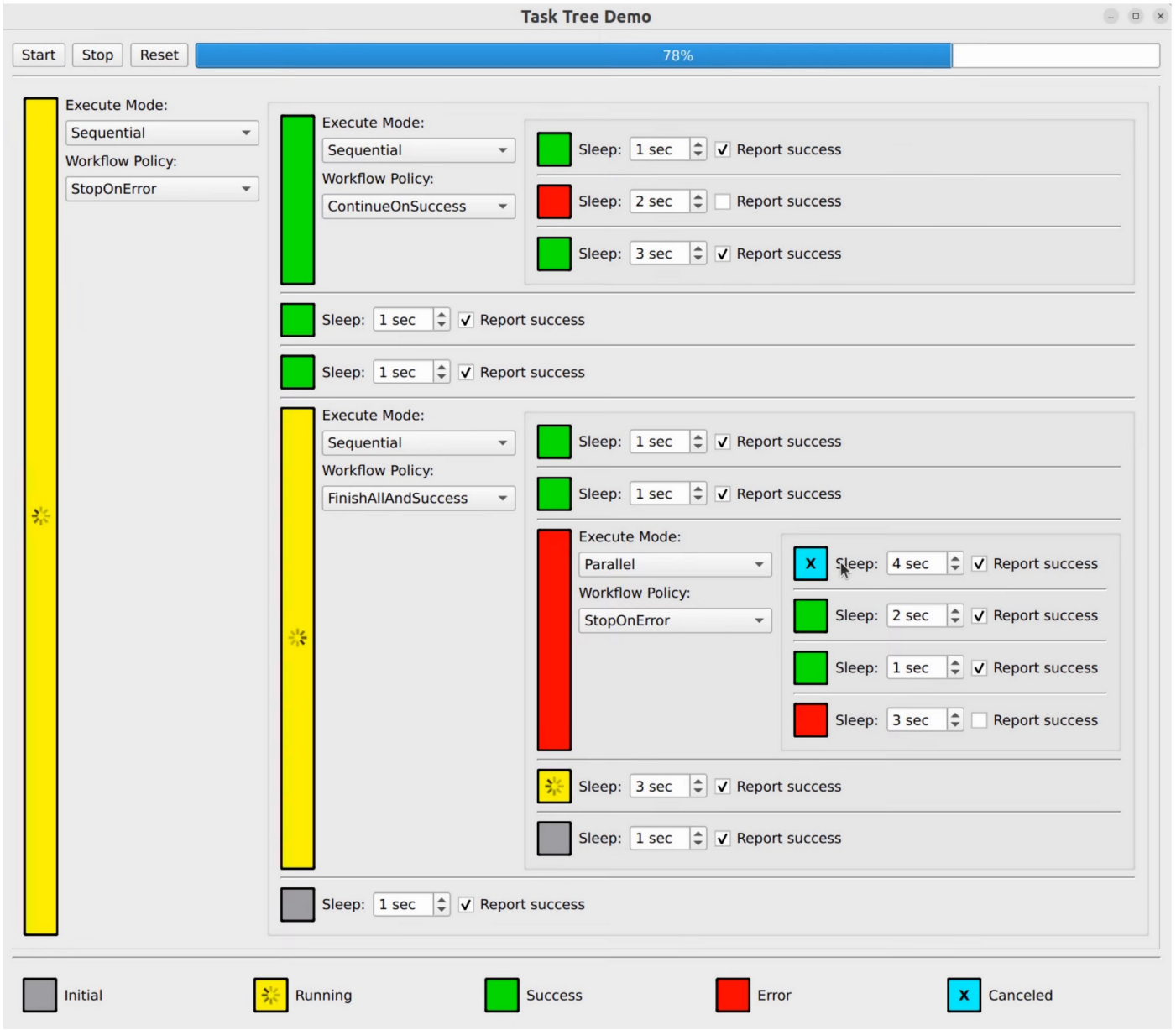
```
const Group recipe1 {
    task1 && task2, // short-circuiting conjunction
    task3 || task4, // short-circuiting disjunction
    !task           // negation
};

const Group recipe2 {
    If (condTask) >> Then {
        task1
    } >> Else {
        task2
    }
};

const Group recipe3 {
    Forever {
        task1
    }
};
```


© The Qt Company

TaskTree demo...



Qt

TaskTree

Questions?

Thank you!

Jarek Kobus / The Qt Company

More info:

https://wiki.qt.io/TaskTree