ChatGBT AThreads Basics

info

- In Qt, QThread is a class that provides a platform-independent way to manage threads. It allows you to create and manage threads within your Qt application. Threads are a way to execute code concurrently, allowing tasks to run independently of the main application thread. This is useful for tasks that might block the main thread, such as time-consuming computations or I/O operations.

use

Step 1 - Subclass QThread

- To create a new thread, you need to subclass the QThread class and override the run() method. The run() method will contain the code that you want to execute in the new thread.

```
#include <QThread>

class MyThread : public QThread {
    Q_OBJECT
public:
    void run() override {
        // Code to be executed in the thread
        // ...
    }
};
```

Step 2 - create a start thread

- To use the custom thread class, create an instance of it, and call its start() method. This will initiate the execution of the run() method in a separate thread.

```
MyThread thread;
thread.start();
```

Step 3 - Communication Between Threads:

- Communication between the main thread and the custom thread can be achieved using signals and slots. You can emit signals from the thread and connect them to slots in the main thread to update the UI or exchange data.

```
class MyThread : public QThread {
    Q_OBJECT
signals:
```

```
    void threadSignal();

public:
    void run() override {
        // Code to be executed in the thread
        // ...

        // Emit a signal when the thread work is done
        emit threadSignal();
    }
};
```

- main thread:

```
MyThread thread;
QObject::connect(&thread, &MyThread::threadSignal, this,
&MainWindow::handleThreadSignal);
thread.start();
```

Step 4 - Thread ellimination

- Threads should be terminated gracefully. You can use the quit() method to stop the thread's event loop and then call wait() to wait for the thread to finish.

```
thread.quit();
thread.wait();
```

Outro

- Remember that although QThread provides a convenient way to manage threads, Qt also offers higher-level constructs like QRunnable and QtConcurrent that might suit your needs better depending on the complexity of your threaded tasks.

When working with threads, it's important to consider thread safety, synchronization, and avoiding race conditions. Additionally, starting and managing threads comes with its own set of considerations, such as memory management and avoiding resource leaks.