Anatomy of Chromium MessageLoop *(PUBLIC)*

kinuko@chromium.org
Apr 12, 2015
(Originally written for MessageLoop refactoring for making Thread startup non-blocking. If you find anything wrong please let kinuko@ know!)

Overview

MessageLoop is one of the very core primitives of Chromium, whose role is to process events and tasks for a particular thread. It has two types of important methods: **PostTask**[1] and **Run**, and its basic functionality (that is observable from outside the class) is to accept posted tasks and to run them on the associated thread. Normally (e.g. if it is not for testing code) the current thread's MessageLoop instance can be retrieved via MessageLoop::current() static method, but typically most production code does not (and **should not**[2]) work directly with the MessageLoop.

PostTask methods family

PostTask can be used to schedule a new task to run on the thread that is associated to the MessageLoop. It is not encouraged to directly call the MessageLoop's methods to post tasks, as we are deprecating those old PostTask interfaces in favor of newer, more generic TaskRunner. The current thread's TaskRunner can be retrieved by calling ThreadTaskRunnerHandle::Get().

Run method

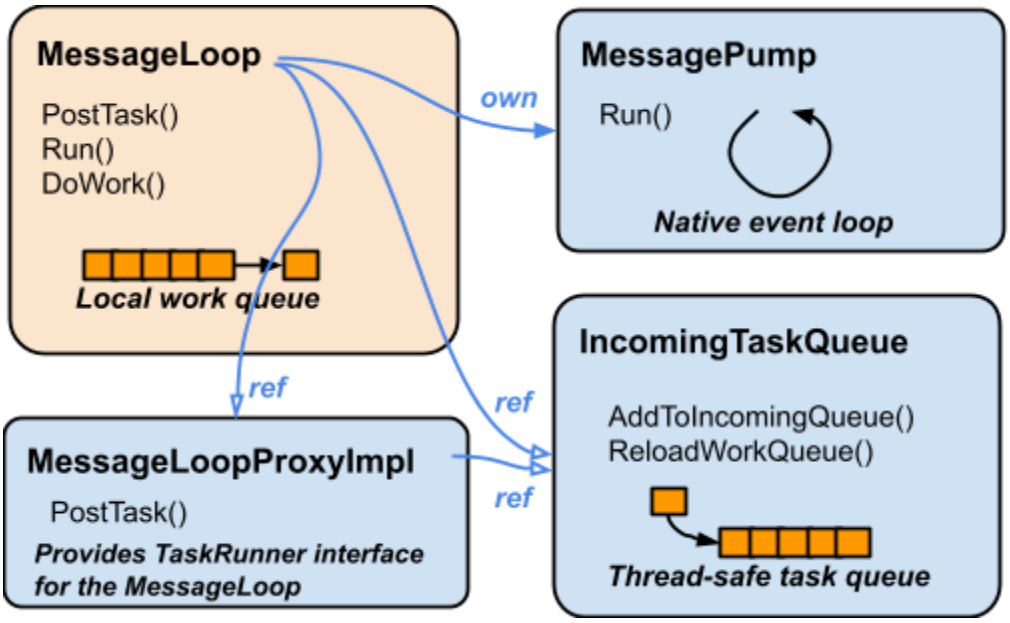
Run method tells the MessageLoop's internal message pump to start processing native events (messages) as well as executing tasks that are posted on to the MessageLoop. Chromium's thread class (base::Thread) calls this method in its thread function, so most production code does not need to call it explicitly. In testing code it is encouraged to run a message loop via RunLoop interface.

Classes that work closely with MessageLoop

MessageLoop class work very closely with a few classes, namely: MessagePump, IncomingTaskQueue and MessageLoopProxyImpl. Instances of these classes are created when MessageLoop is instantiated, and they are owned (or referenced) by the MessageLoop.

---

[1] There are actually four methods of "PostTask" family (with and without delay x nestable and non-nestable): PostTask, PostDelayedTask, PostNonNestableTask and PostNonNestableDelayedTask.
[2] As described below, directly calling MessageLoop::PostTask and MessageLoop::Run is strongly discouraged.



MessagePump

Internally a MessageLoop is paired with a MessagePump, that is a platform-specific message pump implementation and is responsible for processing native events (e.g. Windows messages, IO events etc). MessagePump provides Delegate interface, which has a set of **DoWork** methods[3] that are to be called within the platform's native message/event loop. MessageLoop implements this Delegate interface and run the posted tasks in the DoWork methods. MessagePump's single most important method is **Run**, that starts processing native events and calling MessageLoop::Delegate's DoWork methods. (MessagePump is another beast that probably needs a separate document to explain its details)

IncomingTaskQueue

IncomingTaskQueue is a thread-safe task queue that accumulates tasks for its corresponding MessageLoop. IncomingTaskQueue has two important methods, **AddToIncomingQueue** and **ReloadWorkQueue**. AddToIncomingQueue enqueues a given task to its internal thread-safe task queue, and ReloadWorkQueue swaps a given task queue with its internal task queue. Typically tasks posted to a MessageLoop are enqueued to the IncomingTaskQueue (via MessageLoopProxyImpl) by AddToIncomingQueue. The tasks queued into the IncomingTaskQueue are then eventually loaded into the MessageLoop's internal work queue by ReloadWorkQueue.

MessageLoopProxyImpl

MessageLoopProxyImpl extends MessageLoopProxy (which extends SingleThreadTaskRunner), which provides MessageLoopProxy and TaskRunner interfaces on top of IncomingTaskQueue. MessageLoopProxyImpl provides a set of **PostTask** methods, which basically just forward posted tasks to IncomingTaskQueue.

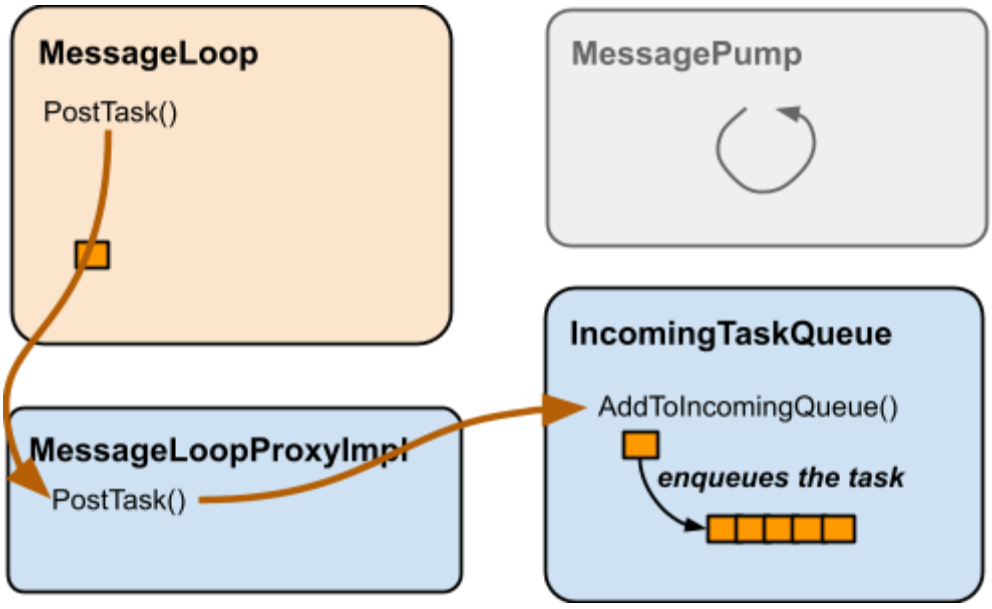
---

[3] MessagePump::Delegate defines three DoWork methods: DoWork, DoDelayedWork and DoIdleWork.

MessageLoop::message_loop_proxy() and MessageLoop::task_runner() return an instance of this class. (In another word, this class provides TaskRunner interface for its corresponding MessageLoop.)

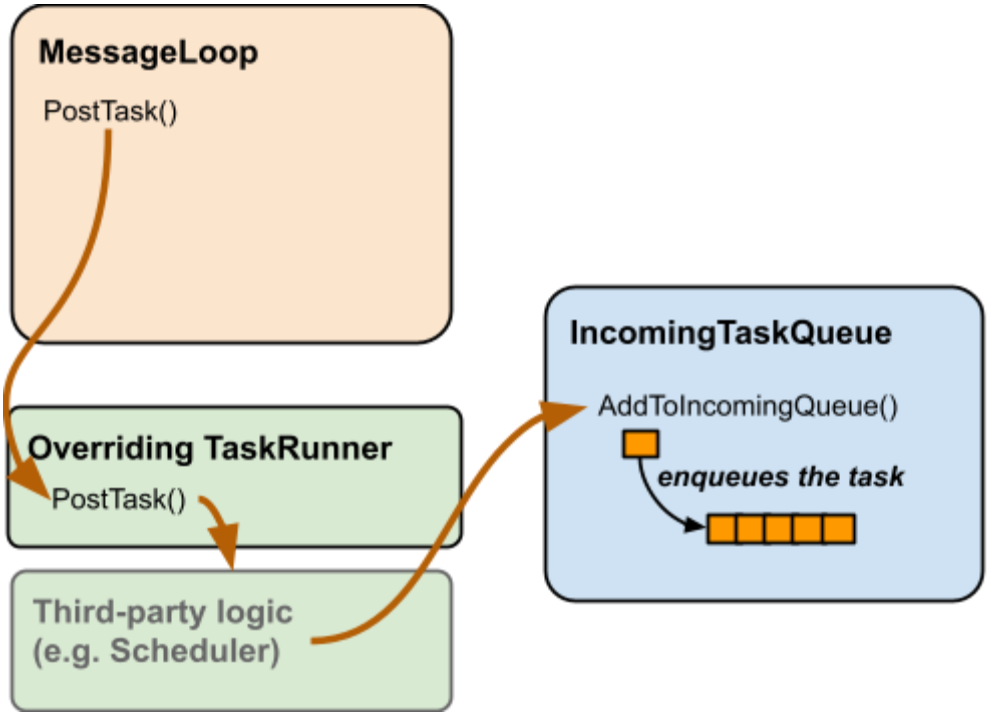
PostTask Flow

A task passed to MessageLoop::PostTask() is forward to MessageLoopProxyImpl::PostTask(), which in turn calls IncomingTaskQueue::AddToIncomingQueue() to queue the task to IncomingTaskQueue's internal thread-safe task queue.



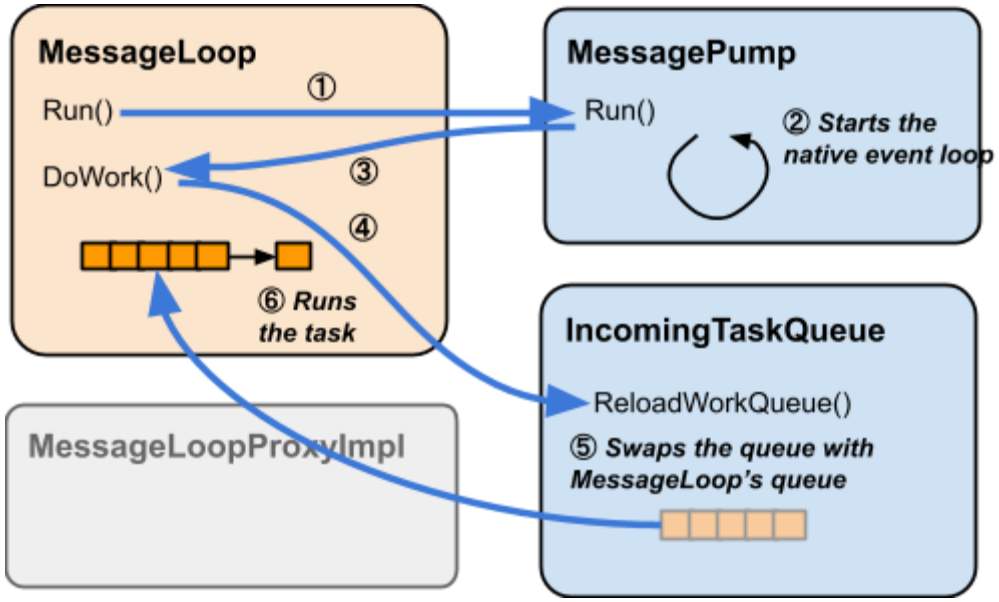
PostTask flow with TaskRunner Overriding

If we start to allow customers of MessageLoop to override its TaskRunner interface (as proposed in "Proxying MessageLoop tasks to the Scheduler") the PostTask flow with overriding TaskRunner would look like following:



Run Loop Flow

Tasks queued to the IncomingTaskQueue's internal queue are eventually loaded to the MessageLoop's work queue by ReloadWorkQueue() and then run in one of DoWork() methods family, that are called within MessagePump's native event loop. Note that the work queue reloading only happens when the local work queue becomes completely empty, this means that the cost of reloading the queue (i.e. taking a lock) is amortized by doing the reload as seldom as possible.


MessageLoop
Run()
①
DoWork()
③
④
⑥ Runs the task
MessagePump
Run()
② Starts the native event loop
IncomingTaskQueue
ReloadWorkQueue()
⑤ Swaps the queue with MessageLoop's queue
MessageLoopProxyImpl