Subject: BackgroundTask
Posted by dolik.rce on Wed, 16 Feb 2011 12:52:45 GMT
View Forum Message <> Reply to Message

As discussed in the thread about MT in theide, there is sometimes need to run some lengthy task in background, while the rest of Application stays responsive.

Ideal solution should use Thread in MT mode and ProcessEvents() in single threaded environment to achieve maximum effectiveness. Here is my first attempt to create such class:bool ProcessForeground(){
#ifdef _MULTITHREADED
 DUMP(Thread::IsShutdownThreads());
 return !Thread::IsShutdownThreads();
#else
 Ctrl::ProcessEvents();
 return true; //is it possible to detect application is being closed?
#endif
}

class BackgroundTask{
 typedef BackgroundTask CLASSNAME;
 #ifdef _MULTITHREADED
 Thread t;
 #endif
 bool running;
 void Watch(Callback task) {running=true; task(); running=false;}
public:
 void Start(Callback task);
 bool IsRunning() {return running;}
 BackgroundTask():running(false){}
 BackgroundTask(Callback task) {Start(task);}
};

void BackgroundTask::Watch(Callback task){
 running=true;
 task();
 running=false;
}

void BackgroundTask::Start(Callback task){
 #ifdef _MULTITHREADED
 t.Run(THISBACK1(Watch,task));
 #else
 PostCallback(THISBACK1(Watch,task));
 #endif
}
There is few requirements on the task and application that runs it. Task must call ProcessForeground() periodically, often enough to keep the UI responsive. If

ProcessForeground() returns false, the task must finish as soon as possible. In MT mode the application must call Thread::ShutdownThreads(), so that any running tasks know that it is time to terminate itself.

In attachment there is a full code of a testing app. Known bugs: In ST mode you must close the window twice (press Alt+F4 or cross button twice), to make it really close. Probably easy to fix. In MT, calling ShutdownThreads assumes that there are no threads expected to run even after the Background tasks are terminated. This should be rarely problem, but to have a peace in mind, I would prefer separate solution.

I'm looking forward to your comments,
Honza

File Attachments

1) bgtask.cpp, downloaded 419 times

Subject: Re: BackgroundTask
Posted by dolik.rce on Wed, 16 Feb 2011 14:32:13 GMT
View Forum Message <> Reply to Message

Slightly improved version:

```
class BackgroundTask{
 typedef BackgroundTask CLASSNAME;
 #ifdef _MULTITHREADED
 Thread t;
 #endif
 bool running;
 const int id;
 static Index<int> list;

 void Watch(Callback task);
 static int AssignId();

 friend bool ProcessForeground(int);
public:
 void Start(Callback task);
 void Stop()                              {int n=list.Find(id); if(n>=0) list.Remove(id);}
 int GetId()                             {return id;}
 bool IsRunning()                            {return running;}
 BackgroundTask():running(false),id(AssignId())   {}
 BackgroundTask(Callback task):id(AssignId())     {Start(task);}
};
Index<int> BackgroundTask::list;

void BackgroundTask::Watch(Callback task){
 running=true;
 task();
 Stop(id);
```

```
 running=false;
}

int BackgroundTask::AssignId(){
 static int sid=0;
 return sid++;
}

void BackgroundTask::Start(Callback task){
 if(IsRunning()) Stop();
 list.Add(sid);
 #ifdef _MULTITHREADED
 t.Run(THISBACK1(Watch,task));
 #else
 PostCallback(THISBACK1(Watch,task));
 #endif
}

bool ProcessForeground(int id){
#ifndef _MULTITHREADED
 Ctrl::ProcessEvents();
#endif
 return BackgroundTask::list.Find(id)>=0;
}
```

It allows stopping tasks individually based on unique ID. Also reusing the task instances should work now. The major difference is that ProcessForeground now takes task ID as argument.

Honza

EDIT: Removed file due to contained errors

Subject: Re: BackgroundTask
Posted by mirek on Wed, 16 Feb 2011 14:41:31 GMT
View Forum Message <> Reply to Message

1. I would not bother with ST here. 'background' processing with timerqueue is way too different from MT IMO to try to reconcile both methods in the single class.
2. Having said that, I do not see any advantage of using BackgroundTask over using plain Thread...

Subject: Re: BackgroundTask
Posted by dolik.rce on Wed, 16 Feb 2011 15:00:22 GMT
View Forum Message <> Reply to Message

mirek wrote on Wed, 16 February 2011 15:411. I would not bother with ST here. 'background'

processing with timerqueue is way too different from MT IMO to try to reconcile both methods in the single class.
2. Having said that, I do not see any advantage of using BackgroundTask over using plain Thread...
Let me start with the second point: The advantage is that it can be compiled in ST where Thread is not defined.

The goal of this actually IS to provide single interface for both MT and ST solutions. I am aware that it doesn't solve every possible scenario, but I think there is non-trivial group of cases where this might be handy. If extended with couple of Callbacks (WhenFinished,WhenProgress,WhenStopped), it might be a good way to manage simple tasks e.g. data loading or processing.

BTW: The above code contained couple errors, they are fixed in the attachment.

File Attachments
1) bgtask.cpp, downloaded 279 times

Subject: Re: BackgroundTask
Posted by koldo on Thu, 17 Feb 2011 17:31:36 GMT
View Forum Message <> Reply to Message

Hello

Not the same but similar is the gif animation in RasterPLayer. See Reference/AnimatedClip demo.

RasterPlayer::SetMT() method if compiled with MT, chooses ST or MT (if compiled with ST, only ST is available).

Subject: Re: BackgroundTask
Posted by dolik.rce on Thu, 17 Feb 2011 18:17:24 GMT
View Forum Message <> Reply to Message

koldo wrote on Thu, 17 February 2011 18:31Hello

Not the same but similar is the gif animation in RasterPLayer. See Reference/AnimatedClip demo.

RasterPlayer::SetMT() method if compiled with MT, chooses ST or MT (if compiled with ST, only ST is available).
Yep, very similar idea BackgroundTask could actually have the SetMT() method as well, if desired. But I think it would be a bit against the original idea (to provide uniform way to manage background tasks with most effective technology based on MT flag presence).

Honza