Subject: Re: [PROPOSAL]: AsyncQueue class (a single threaded synchronization tool) for U++
Posted by Oblivion on Tue, 01 Mar 2016 23:22:31 GMT
View Forum Message <> Reply to Message

Hello Mirek,

I've got rid of VarArgs, VectorMap, DoJob(), and re-based AsyncQueue class as a thin wrapper of vector containing callbacks.
Now it can also utilize lambda callbacks in-place (e.g. this way any StartFoo() can be both used for programming the queue, and as the place to write the actual non-blocking code.), and have backward compatibility.

While U++ callbacks allow up to 5 args, using std::tuple and std::get it can be increased by the user, if necessary.

If you approve this one, I'll rewrite the document accordingly and upload the package asap...

```
class AsyncQueue : Moveable<AsyncQueue> {

   Vector<Callback> job_queue;
   bool halted;

protected:
   Callback&         AddJob()                          { return job_queue.Add(); }
   AsyncQueue&       AddJob(Callback cb)                       { job_queue.Add(cb); return *this; }
   Callback&         InsertJob(int i)                  { return job_queue.Insert(i); }
   AsyncQueue&       InsertJob(int i, Callback cb)             { job_queue.Insert(i, cb); return *this; }
   Callback&         GetJob(int i)                     { return job_queue.At(i); }
   void              RemoveJob(int i)                  { job_queue.Remove(i); }
   void              ProcessQueue()                     { if(!job_queue.IsEmpty())
job_queue.Remove(0); }
   void              ClearQueue()                     { if(!job_queue.IsEmpty()) job_queue.Clear();
halted = false; }
   bool              QueueHasJobs() const                 { return job_queue.GetCount() > 1; }
   bool              QueueIsHalted() const               { return halted; }
   bool              QueueIsEmpty() const                 { return job_queue.IsEmpty(); }
   bool              GetJobCount() const                { return job_queue.GetCount(); }
   void              Halt()                        { job_queue.Clear(); halted = true; }

public:
   virtual bool      Do()                           { if(!job_queue.IsEmpty()) GetJob(0)(); WhenDo();
return InProgress(); }

   bool              InProgress() const                 { return !job_queue.IsEmpty();  }
```

```
    bool            IsSuccess() const               { return job_queue.IsEmpty() && !halted; }
    bool            IsFailure() const               { return halted;  }

    Callback        WhenDo;

    AsyncQueue() : halted(false)    {}
    virtual ~AsyncQueue()           {}

    AsyncQueue(AsyncQueue rval_ a)                  { job_queue = pick(a.job_queue); halted =
a.halted; }
    void operator=(AsyncQueue rval_ a)              { ClearQueue();  job_queue =
pick(a.job_queue); halted = a.halted; }
};
```

What do you think?

Regards,
Oblivion