Subject: Re: RE: Job package: A scope-bound worker thread for non-blocking operations.
Posted by mirek on Tue, 10 Oct 2017 14:10:09 GMT
View Forum Message <> Reply to Message

My try.

```
#include <Core/Core.h>

using namespace Upp;

template <class Ret>
class AsyncWork {
 template <class Ret>
 struct Imp {
  CoWork co;
  Ret    ret;

  template< class Function, class... Args>
  void       Do(Function&& f, Args&&... args) { co.Do([=]() { ret = f(args...); }); }
  const Ret& Get()                            { return ret; }
 };

 template <>
 struct Imp<void> {
  CoWork co;

  template< class Function, class... Args>
  void       Do(Function&& f, Args&&... args) { co.Do([=]() { f(args...); }); }
  void       Get()                            {}
 };

 One<Imp<Ret>> imp;

public:
 template< class Function, class... Args>
 void  Do(Function&& f, Args&&... args)    { imp.Create().Do(f, args...); }

 void  Cancel()                            { if(imp) imp->co.Cancel(); }
 Ret   Get()                               { ASSERT(imp); imp->co.Finish(); return imp->Get(); }
 Ret   operator~()                         { return Get(); }

 AsyncWork(AsyncWork&&) = default;
 AsyncWork()                               {}
 ~AsyncWork()                              { if(imp) imp->co.Cancel(); }
};
```

```
template< class Function, class... Args>
AsyncWork<std::result_of_t<std::decay_t<Function>(std::decay_t<Args>...)>>
Async(Function&& f, Args&&... args)
{
 AsyncWork<std::result_of_t<std::decay_t<Function>(std::decay_t<Args>...)>> h;
 h.Do(f, args...);
 return pick(h);
}

CONSOLE_APP_MAIN
{
 auto h = Async([] { return "Hello world"; });
 DDUMP(h.Get());

 DDUMP(~Async([](int x) { return x * x; }, 9));

 auto x = Async([] { throw "error"; });
 try {
  x.Get();
 }
 catch(...) {
  DLOG("Exception caught");
 }
}
```

I guess it is quite similar now.
