Currently, programmers have to implement the following code to check for the status of a future:

```cpp
auto fut = std::async(task);
...
#include <chrono>
switch (fut.wait_for(std::chrono::seconds(0))) {
  case std::future_status::timeout:
    std::cout << "thread still running\n";
    break;
  case std::future_status::ready:
    std::cout << "thread done, outcome available\n";
    break;
  case std::future_status::deferred:
    std::cout << "thread was deferred\n";
    break;
}
```

As checking for the state of a future is pretty common, we should simply this. Note that the concurrency TS provides a function, which however only can check, whether a future is ready:

```cpp
template <class R>
class future {
  public:
    bool is_ready() const;
};
```

Returns:

- true if the shared state is ready, otherwise false.

I see the following options:

1. provide additional helpers like this
   ```cpp
   bool is_deferred() const;
   bool is_running();
   ```
2. provide as additional interface:
   ```cpp
   std::future_status status() const;
   ```

Both options need some discussion and additional changes:
Option 1: adding is_ready(), is_deferred(), is_running()

is_running() is not necessary as it could be computed as:

\[
\text{!is_deferred()} \&\& \text{!is_ready()}
\]

But that’s again clumsy.

Also you can argue about the name here, because strictly speaking the result only claims that the result is not available yet.

So other possible names are:

\[
\begin{align*}
\text{is_unavailable()} \\
\text{is_about_being_computed()}
\end{align*}
\]

Option 2: adding status()

This extension has the benefit that we don’t have to introduce new function when we introduce new future status. Also a switch statement as at the beginning of the paper would be possible.

However, the current future status ‘timeout’ is VERY confusing because it applies to the kind of call (“wait()”) instead of the status the future has.

For this reason we should add a new name for the timeout status, e.g. “unavailable” or “running”:

```cpp
enum class future_status { 
  ready,       
  timeout,     
  deferred, 
  running = timeout  // added
};
```

Thus, the test would be:

```cpp
switch (fut.status())) {
  case std::future_status::running:  
    std::cout << "thread still running\n";
    break;
  case std::future_status::ready:    
    std::cout << "thread done, outcome available\n";
    break;
  case std::future_status::deferred: 
    std::cout << "thread was deferred\n";
    break;
}
```

or simply:

```cpp
if (fut.status() == std::future_status::deferred)
```

If we think it is useful, we can also deprecate “timeout”.

Due to the better flexibility and robustness I propose option 2.
Proposed Wording

(All against N4660)

33.6.2 Header <future> synopsis [future.syn]
In the synopsis add:

```cpp
enum class future_status {
    ready,
    timeout,
    deferred,
    running = timeout
};
```

33.6.7 Class template future [futures.unique_future]
In the declaration of
```cpp
template <class R>
class future {
    ...
}
```
after the declaration of
```cpp
bool valid();
```
add:
```cpp
future_status status() const;
```

After the definition of:
```cpp
bool valid() const noexcept;
```
add:
```cpp
future_status status() const;
```
Returns: `true` only if *this refers to a shared state.

After the definition of:
```cpp
future_status status() const;
```
Returns: `wait(chrono::seconds(0));`
OPEN or:
```
Returns:
(22.1) — future_status::deferred if the shared state contains a deferred function.
(22.2) — future_status::ready if the shared state is ready.
(22.3) — future_status::running otherwise (if the shared state contains no
defined function and is not ready.
```

OPEN: Could it throw timeout-related exceptions ???

Acknowledgments
Thanks to Jonathan Wakely, Billy O’Neal, and Detlev Vollmann for the support writing this paper.