Using Web Services in C++

Steve Gates, Microsoft

stgates@microsoft.com

Overview

- What a typical web service looks like
- Some options for consuming
- Share a library, the C++ Rest SDK, we've been building
  - Walkthrough how to use
  - How you can get involved if interested

- We'll be looking at code
  - Snippets on slides
  - In IDE
- Cursory knowledge of HTTP, JSON, REST, and some other standards and protocols assumed

What exactly is a web service?

Or a network service?

Remote Procedure Call

- RFC 707 – "High-level framework for network-based resource sharing"
- At its simplest, the ability to run a function on another machine and get the result
- A server offers a set of callable operations to clients
- Using familiar local procedure calls, hides away
  - Parameters, message format
  - Transmission protocol details

What is a web service?

- WC3 – “A Web service is a software system designed to support interoperable machine-to-machine interaction over a network.”
- In general I think of a ‘web service’ as a method of communication between two computers over the world wide web
- Two popular variants are SOAP and REST
- Term ‘web service’ is somewhat intertwined with SOAP
- This talk will mainly focus on REST style services, or web APIs delivered over HTTP

REpresentational State Transfer

- Architectural style for designing a distributed system
- A service is RESTful if conforms to the following set of constraints:
    - Client-server separation
    - Stateless
    - Uniform interface
    - Cacheable
    - Layered system
    - Code on demand (optional)
- For a web services this means
    - Interact with resources using URIs
    - Interface using HTTP methods
    - No specific data format, but often JSON

OK what are some real examples…

Popular web services/web APIs

Service	Protocols/Standards
Google Maps	HTTP, URI, XML, JSON
Facebook Graph API	HTTP, URI, OAuth, JSON
Twitter	HTTP, URI, OAuth, JSON
Amazon S3	HTTP, URI, XML
Azure Storage	HTTP, URI, XML, JSON (tables)
Dropbox	HTTP, URI, OAuth, JSON
WordPress	HTTP, URI, OAuth, JSON

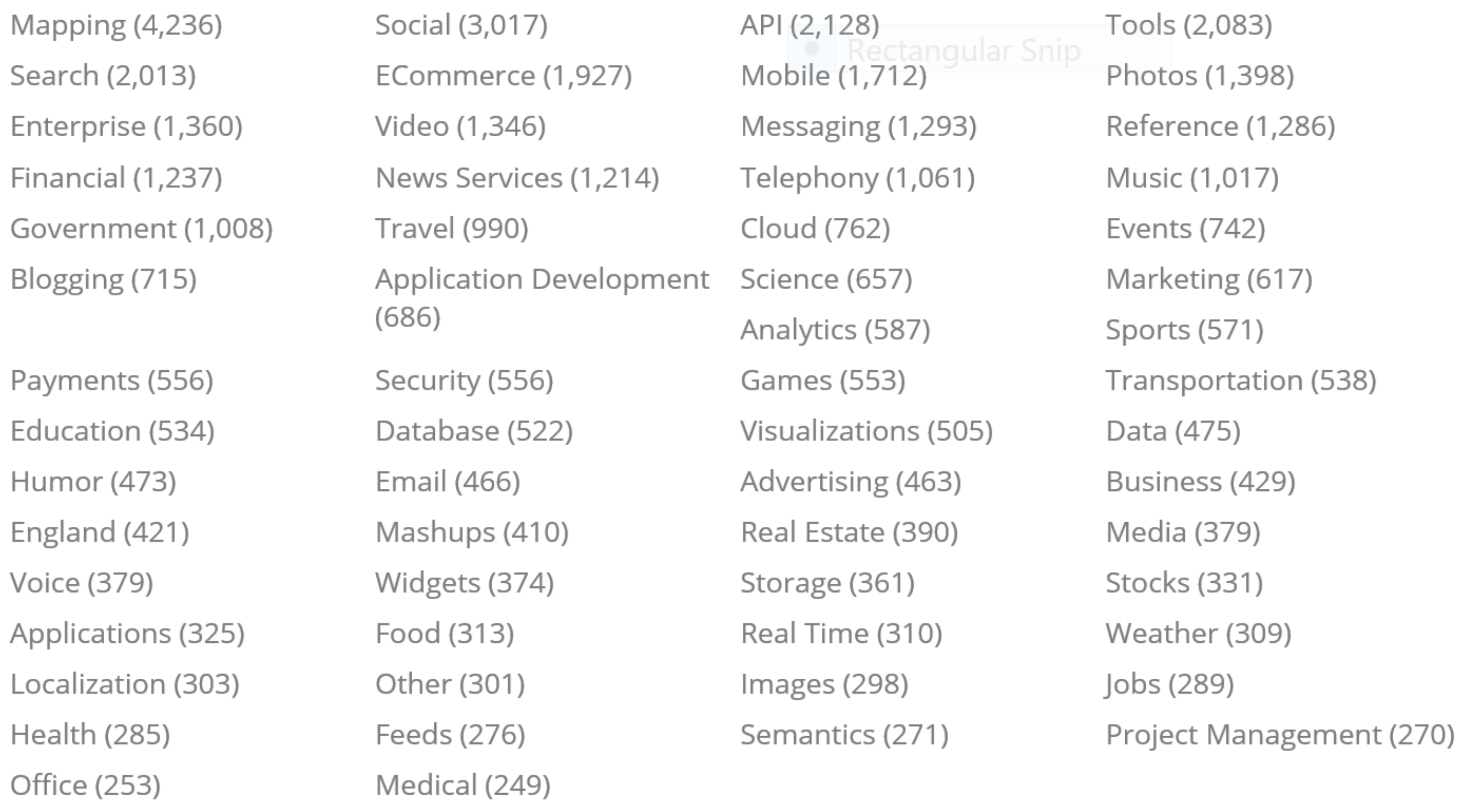
- Common protocols and standards – HTTP, URI, JSON, OAuth

Popular web services/web APIs

- Those are just a handful of web APIs, there are tons look at
  - http://www.programmableweb.com/
  - http://www.mashape.com/

Popular web services/web APIs

- Just first page of categories on ProgrammableWeb



Mapping (4,236)	Social (3,017)	API (2,128)	Tools (2,083)
Search (2,013)	ECommerce (1,927)	Mobile (1,712)	Photos (1,398)
Enterprise (1,360)	Video (1,346)	Messaging (1,293)	Reference (1,286)
Financial (1,237)	News Services (1,214)	Telephony (1,061)	Music (1,017)
Government (1,008)	Travel (990)	Cloud (762)	Events (742)
Blogging (715)	Application Development (686)	Science (657)	Marketing (617)
		Analytics (587)	Sports (571)
Payments (556)	Security (556)	Games (553)	Transportation (538)
Education (534)	Database (522)	Visualizations (505)	Data (475)
Humor (473)	Email (466)	Advertising (463)	Business (429)
England (421)	Mashups (410)	Real Estate (390)	Media (379)
Voice (379)	Widgets (374)	Storage (361)	Stocks (331)
Applications (325)	Food (313)	Real Time (310)	Weather (309)
Localization (303)	Other (301)	Images (298)	Jobs (289)
Health (285)	Feeds (276)	Semantics (271)	Project Management (270)
Office (253)	Medical (249)		

How can I use these web APIs now?

Available options

- Some services have dedicated client side SDKs (probably not in C++)
- Write directly to the exposed HTTP endpoints
  - Most languages (other than C++) have good library support
- What about C++?
  - Use existing HTTP library
  - Could use platform specific HTTP APIs when available
  - Write HTTP on top of TCP socket library
- What about if you care about asynchrony, cross platform, and C++11 style?

The C++ Rest SDK

- The C++ Rest SDK aims to fill these gaps by providing the building blocks for accessing services with high level APIs covering:
  - HTTP, URIs, JSON, OAuth, and WebSockets
- Approach is not to re-write everything, re-use what is appropriate
  - Reuse existing open source libraries
  - Build on platform APIs
- Goal is to make it easier to consume web APIs and write client SDK libraries

The C++ Rest SDK

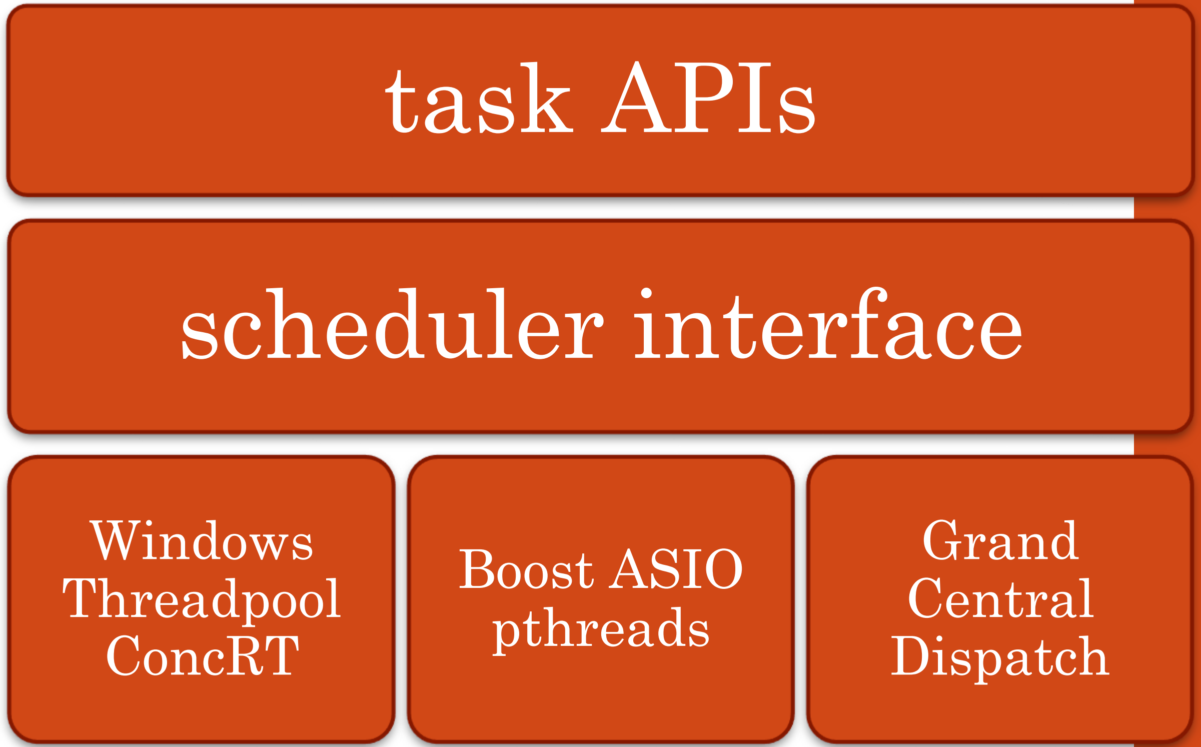
- Simple APIs
  - Having every single feature is not as important as a straight forward API
- Asynchronous
  - All I/O and potentially long running work needs to be asynchronous
- C++11 style

- All the code I show you the today runs on Windows desktop/server (XP+), Windows Store, Windows Phone, OS X, iOS, Ubuntu, and Android

Asynchrony
pplx::task

pplx::task

- Cross platform tasks from Parallel Patterns Library (PPL)
- Similar to std::future, but with continuations
- Later could be replaced with futures (N3970)
- Lots of existing presentations and resources on task based programming



pplx::task – continuations

```
pplx::task<int> intTask = start_op();


intTask.then([](int value)
{
    // Execute some work once operation has completed...
});
```

- Can also compose tasks using pplx::when_any and pplx::when_all constructs

pplx::task – exception handling

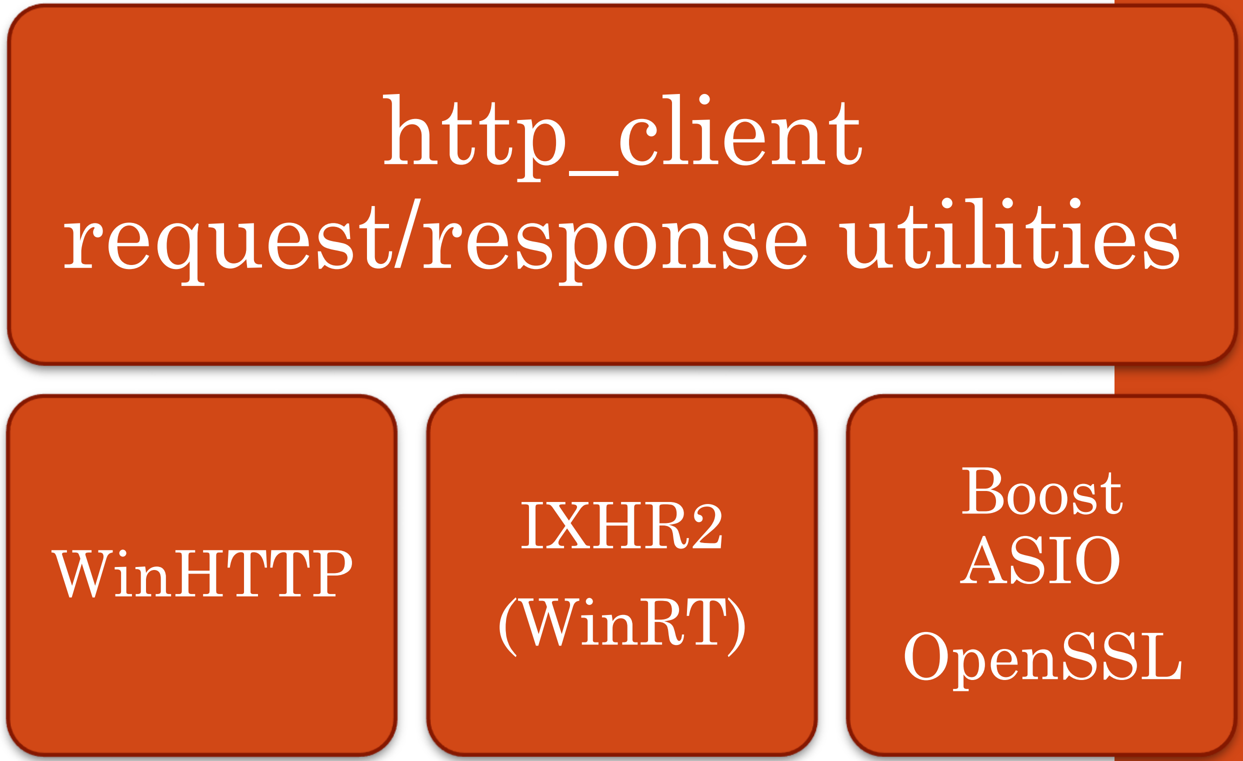
```
pplx::task<int> intTask = pplx::create_task([]() -> int
    { throw std::runtime_error("error"); });


intTask.then([](pplx::task<int> op)
{
    try
    {
        int value = op.get();
    } catch (const std::runtime_error &e)
    {  /* Perform error handling...  */ }
});
```

Task based continuation, always executes

http_client

http_client

- Support for HTTP/1.1 – client initiated request/response protocol
- Simple high level API for managing HTTP requests
- No dealing with individual connection management



http_client – hello world upload

```
http_client client(U("http://myserver.com"));

http_response response = client.request(methods::POST, U("mypath"),
U("Hello World")).get();

if (response.status_code() == status_codes::OK)

{

    // Inspect response...

}
```

- What is the ‘U’? Platform dependent string type, to allow using preferred type:
  - On Windows UTF-16, std::wstring
  - Other platforms UTF-8, std::string

http_client – hello world upload

```
// Manually build up request.
http_request req(methods::POST);
req.set_request_uri(U("mypath"));
req.headers().add(header_names::user_agent, U(“myclient"));
req.set_body(U("Hello World"));
http_response response = client.request(req).get();
```

- http_client also takes configuration for options like timeouts, chunk size, etc…

http_client – hello world download

```
http_client client(U("http://myserver.com"));
http_response response = client.request(methods::GET).get();
if (response.status_code() == status_codes::OK)
{
    const utility::string_t body = response.extract_string().get();
}
```

- http_response contains functionality for getting body as string, JSON, vector, or a stream

http_client – hello world, better

```
http_client client(U("http://myserver.com"));
client.request(methods::GET).then([](http_response response)
{
    // Check status code...
    return response.extract_string();
}).then([](const utility::string_t &body)
{
    // Use string...
});
```

http_client – request/response flow

```
http_client client(U("http://myserver.com"));
pplx::task<http_response> pendingRequest = client.request(methods::GET);
pendingRequest.then([](http_response response)
{
    pplx::task<utility::string_t> extractString = response.extract_string();
    return extractString;
})
.then([](const utility::string_t &body)
{
    // Use string...
});
```

Entire request sent, response headers arrived

Entire response arrived

It’s not that I don’t believe you, but I need to see it

Yoda Speak

http_client – efficient streaming

```
http_client client(U("http://myserver.com"));

// Create stream backed by a vector.
http_request request(methods::GET);
container_buffer<std::vector<uint8_t>> buffer;
buffer.collection().reserve(1024 * 1024 * 4);
request.set_response_stream(buffer.create_ostream());

// Send request and wait for response to arrive.
http_response response = client.request(request).get();
response.content_ready().wait();
std::vector<uint8_t> body = std::move(buffer.collection());
```

Heap allocation all at once up front

Important to move!

http_client – efficient streaming

```
// Buffer backed by a file.
streambuf<uint8_t> buffer = file_buffer<uint8_t>::open(U("myfile")).get();


// Buffer backed by raw memory.
const size_t size = 1024 * 1024 * 4;
char * raw = new char[size];
rawptr_buffer<uint8_t> buffer(reinterpret_cast<uint8_t *>(raw), size);
```

http_client – efficient streaming

- Buffer backed with contiguous storage can allow for further optimization on some platforms
- Tuning chunk size for message bodies – saves traversing up and down the stack

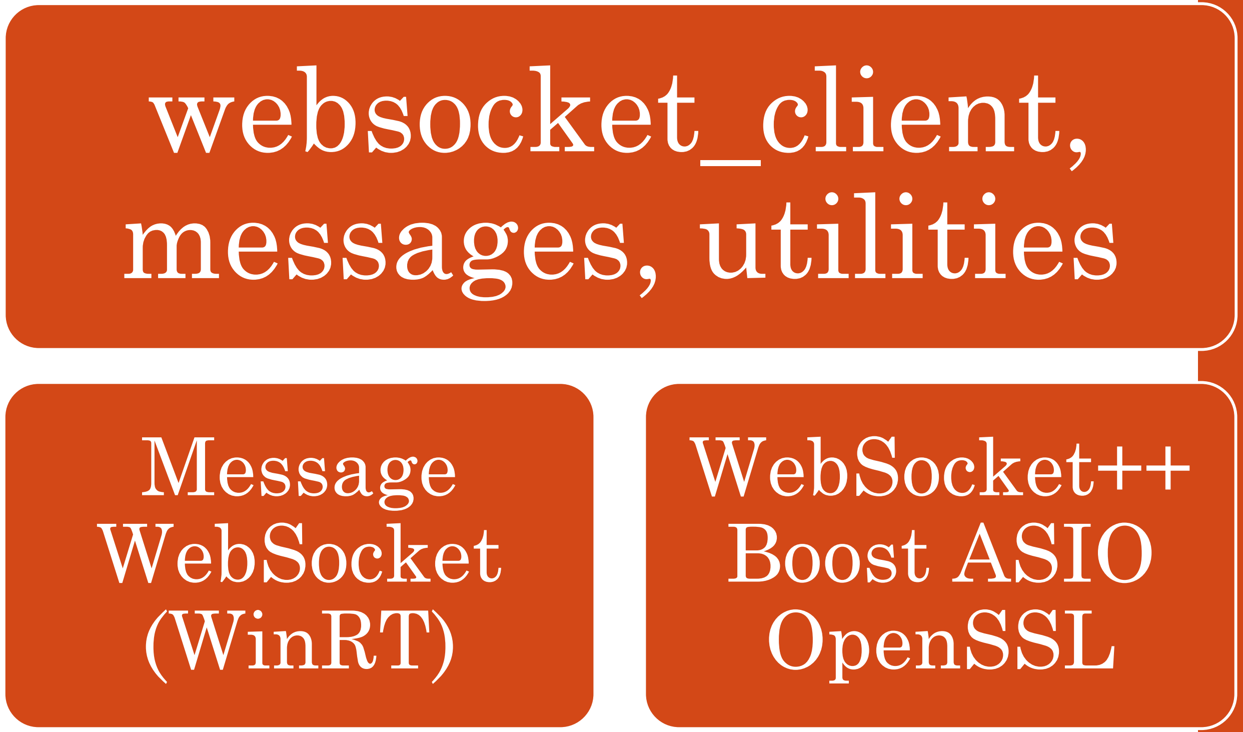
- Acquire pointer to internal storage
- Sending request body –pass directly into platform API for reading
- Receiving response body –pass directly into platform API for writing

Dropbox Upload

websocket_client

websocket_client

- Enables bi-directional communication over a persistent TCP socket after an HTTP request
- In contrast with the HTTP request/response model, servers can push messages
- Domains – frequent small messaging, gaming, interactive collaborative applications


websocket_client, messages, utilities
Message WebSocket (WinRT)
WebSocket++ Boost ASIO OpenSSL

websocket_client – opening connection

```
websocket_client client;
client.connect(U("ws://myserver.com")).then([]
{
    // Connection successfully opened...
});
// Later on when done...
client.close().then([]
{
    // Connection is closed...
});
```

- websocket_client also takes configuration for options like HTTP upgrade request headers and subprotocols

websocket_client – sending

```
// Strings
websocket_outgoing_message msg;
msg.set_utf8_message("Hello World");
pplx::task<void> sendTask = client.send(msg);

// Binary
websocket_outgoing_message msg;
streambuf<uint8_t> buffer = file_buffer<uint8_t>::open(U("myfile")).get();
msg.set_binary_message(buffer.create_istream(), 10); // Send only 10 bytes
pplx::task<void> sendTask = client.send(msg);
```

websocket_client – receiving

```
websocket_client client;

client.receive().then([](websocket_incoming_message msg)
{
    return msg.extract_string(); // Msg body could still be arriving
}).then([](std::string &data)
{
    // Use string...
});
```

- Can also access body as a stream
- Adding option for using callback for repeated message receiving

Cross platform learning

- Test automation infrastructure
  - Gated, rolling, nightly
- Make test cases cross platform by default
- Be careful about build system complexity
- Think of platforms as ‘features’
- Sometimes have to write mini-shims to hide missing libraries
- Even when not fully using cross platform capabilities, users like the option
  - Layer APIs to allow dropping down if necessary

C++ Rest SDK

- For more details the library can be located on CodePlex:
  - http://casablanca.codeplex.com/
- Release as open source under Apache 2.0 license
- We accept contributions, let me know if interested

Questions?