

# Qt and WebServices

Tim Dewhirst <[tim@kdab.com](mailto:tim@kdab.com)>

WARNING!

Acronym Heavy

- What are they?
- History
- Popular service types
- Qt
  - Clients
  - Servers

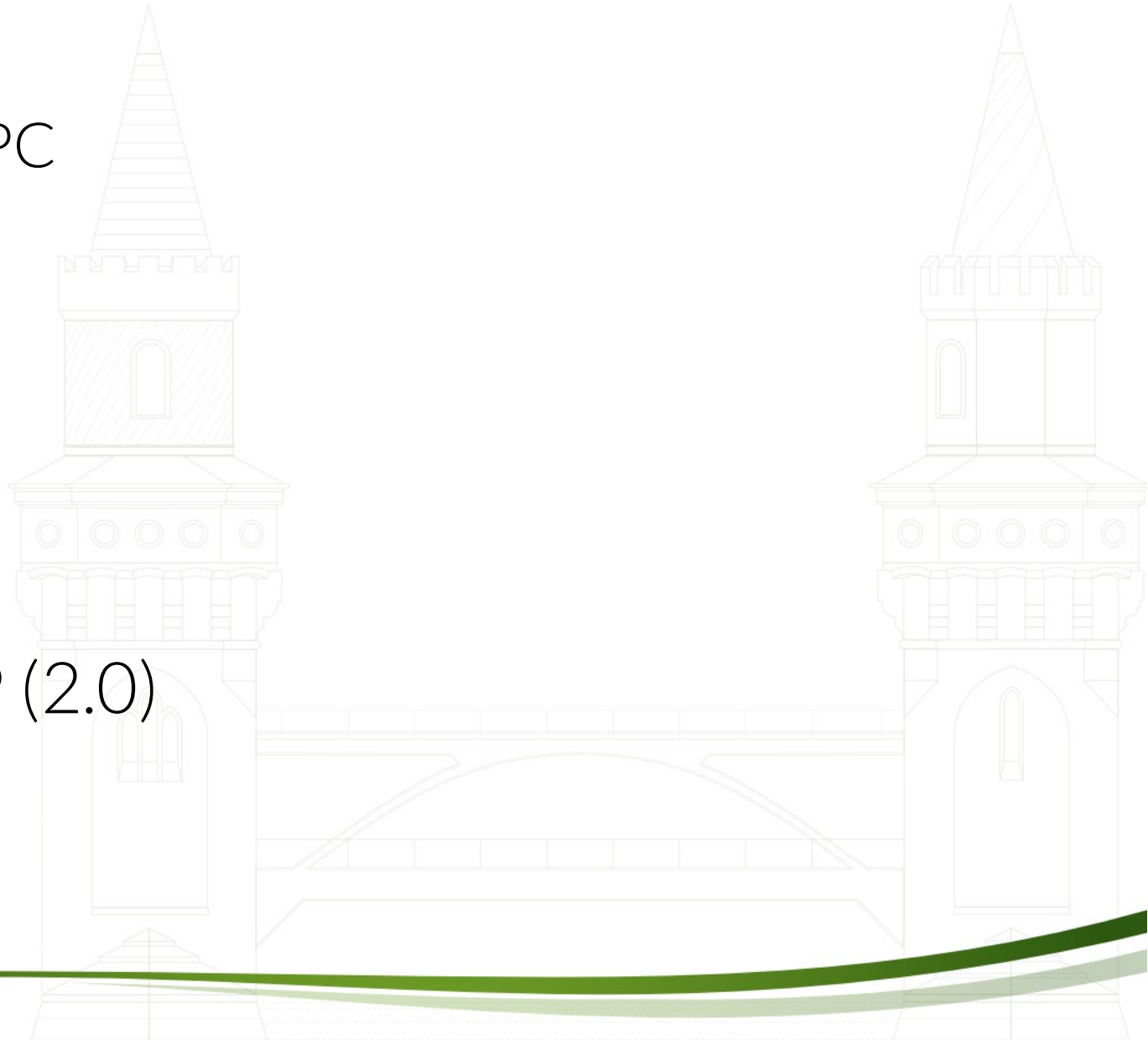


- W3C: "a software system designed to support interoperable machine-to-machine interaction over a network"
- Method of communication between two electronic devices over the web
- Normally client/server
  - WebService

- Standing on the shoulders of giants
  - Web technologies
  - Simple
  - Widely available
  - Human readable
- What about old-skool RPC?
  - CORBA, ICE, DCOM, ...
  - Custom protocols

- Mobile applications
  - Better UX
- Mash-ups
  - Sum is greater than parts (hopefully!)
- Huge informational resource
- Interfacing to services becoming increasingly important

- 1998
  - WDDX, XML-RPC
- 1999
  - SOAP
- 2000
  - REST
- 2005(1.0), 2009 (2.0)
  - JSON-RPC



- SOAP
- JSON-RPC
- REST





- SOAP
  - Evolution of XML-RPC
  - Designed by (amongst others) Don Box
- HTTP & XML
- WSDL
- Very verbose
  - You don't believe me?

```

<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tm="h
<wsdl:types>
  <s:schema elementFormDefault="qualified" targetNamespace="http://www.webserv
  <s:element name="GetWeather">
    <s:complexType>
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="CityName" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="CountryName" type="s:string" />
      </s:sequence>
    </s:complexType>
  </s:element>
  <s:element name="GetWeatherResponse">
    <s:complexType>
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="GetWeatherResult" type="s:string" />
      </s:sequence>
    </s:complexType>
  </s:element>
  <s:element name="GetCitiesByCountry">
    <s:complexType>
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="CountryName" type="s:string" />
      </s:sequence>
    </s:complexType>
  </s:element>
  <s:element name="GetCitiesByCountryResponse">
    <s:complexType>
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="GetCitiesByCountryResult" type="s:string" />
      </s:sequence>
    </s:complexType>
  </s:element>
  <s:element name="string" nillable="true" type="s:string" />
</s:schema>
</wsdl:types>
<wsdl:message name="GetWeatherSoapIn">
  <wsdl:part name="parameters" element="tns:GetWeather" />
</wsdl:message>
<wsdl:message name="GetWeatherSoapOut">
  <wsdl:part name="parameters" element="tns:GetWeatherResponse" />
</wsdl:message>
<wsdl:message name="GetCitiesByCountrySoapIn">
  <wsdl:part name="parameters" element="tns:GetCitiesByCountry" />
</wsdl:message>
<wsdl:message name="GetCitiesByCountrySoapOut">
  <wsdl:part name="parameters" element="tns:GetCitiesByCountryResponse" />
</wsdl:message>
<wsdl:message name="GetWeatherHttpGetIn">
  <wsdl:part name="CityName" type="s:string" />
  <wsdl:part name="CountryName" type="s:string" />
</wsdl:message>
<wsdl:message name="GetWeatherHttpPostIn">
  <wsdl:part name="CityName" type="s:string" />
  <wsdl:part name="CountryName" type="s:string" />
</wsdl:message>
<wsdl:message name="GetWeatherHttpGetOut">
  <wsdl:part name="Body" element="tns:string" />
</wsdl:message>
<wsdl:message name="GetWeatherHttpPostOut">
  <wsdl:part name="Body" element="tns:string" />
</wsdl:message>
<wsdl:message name="GetCitiesByCountryHttpGetIn">
  <wsdl:part name="CountryName" type="s:string" />
</wsdl:message>
<wsdl:message name="GetCitiesByCountryHttpGetOut">
  <wsdl:part name="Body" element="tns:string" />
</wsdl:message>
<wsdl:message name="GetWeatherHttpPostIn">
  <wsdl:part name="CityName" type="s:string" />
  <wsdl:part name="CountryName" type="s:string" />
</wsdl:message>
<wsdl:message name="GetWeatherHttpPostOut">
  <wsdl:part name="Body" element="tns:string" />
</wsdl:message>
<wsdl:message name="GetCitiesByCountryHttpPostIn">
  <wsdl:part name="CountryName" type="s:string" />
</wsdl:message>
<wsdl:message name="GetCitiesByCountryHttpPostOut">
  <wsdl:part name="Body" element="tns:string" />
</wsdl:message>
</wsdl:definitions>
</wsdl:portType>
<wsdl:portType name="GlobalWeatherSoap">
  <wsdl:operation name="GetWeather">
    <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Get weat
    <wsdl:input message="tns:GetWeatherSoapIn" />
    <wsdl:output message="tns:GetWeatherSoapOut" />
  </wsdl:operation>
  <wsdl:operation name="GetCitiesByCountry">
    <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Get all
    <wsdl:input message="tns:GetCitiesByCountrySoapIn" />
    <wsdl:output message="tns:GetCitiesByCountrySoapOut" />
  </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="GlobalWeatherHttpGet">
  <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Get weat
  <wsdl:input message="tns:GetWeatherHttpGetIn" />
  <wsdl:output message="tns:GetWeatherHttpGetOut" />
</wsdl:portType>
<wsdl:portType name="GlobalWeatherHttpPost">
  <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Get weat
  <wsdl:input message="tns:GetWeatherHttpPostIn" />
  <wsdl:output message="tns:GetWeatherHttpPostOut" />
</wsdl:portType>
<wsdl:portType name="GetCitiesByCountry">
  <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Get all
  <wsdl:input message="tns:GetCitiesByCountryHttpGetIn" />
  <wsdl:output message="tns:GetCitiesByCountryHttpGetOut" />
</wsdl:portType>
<wsdl:binding name="GlobalWeatherSoap" type="tns:GlobalWeatherSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
  <wsdl:operation name="GetWeather">
    <soap:operation soapAction="http://www.webserviceX.NET/GetWeather" style="
    <wsdl:input>
      <soap:body use="literal" />
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal" />
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="GetCitiesByCountry">
    <soap:operation soapAction="http://www.webserviceX.NET/GetCitiesByCountry">
    <wsdl:input>
      <soap:body use="literal" />
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal" />
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
<wsdl:binding name="GlobalWeatherSoap12" type="tns:GlobalWeatherSoap">
  <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" />
  <wsdl:operation name="GetWeather">
    <soap12:operation soapAction="http://www.webserviceX.NET/GetWeather" style="
    <wsdl:input>
      <soap12:body use="literal" />
    </wsdl:input>
    <wsdl:output>
      <soap12:body use="literal" />
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
<wsdl:binding name="GlobalWeatherHttpGet" type="tns:GlobalWeatherHttpGet">
  <http:binding verb="GET" />
  <wsdl:operation name="GetWeather">
    <http:operation location="/GetWeather" />
    <wsdl:input>
      <http:urlEncoded />
    </wsdl:input>
    <wsdl:output>
      <mime:mimeXml part="Body" />
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="GetCitiesByCountry">
    <http:operation location="/GetCitiesByCountry" />
    <wsdl:input>
      <http:urlEncoded />
    </wsdl:input>
    <wsdl:output>
      <mime:mimeXml part="Body" />
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
<wsdl:binding name="GlobalWeatherHttpPost" type="tns:GlobalWeatherHttpPost">
  <http:binding verb="POST" />
  <wsdl:operation name="GetWeather">
    <http:operation location="/GetWeather" />
    <wsdl:input>
      <mime:content type="application/x-www-form-urlencoded" />
    </wsdl:input>
    <wsdl:output>
      <mime:mimeXml part="Body" />
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="GetCitiesByCountry">
    <http:operation location="/GetCitiesByCountry" />
    <wsdl:input>
      <mime:content type="application/x-www-form-urlencoded" />
    </wsdl:input>
    <wsdl:output>
      <mime:mimeXml part="Body" />
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
<wsdl:service name="GlobalWeather">
  <wsdl:port name="GlobalWeatherSoap" binding="tns:GlobalWeatherSoap">
    <soap:address location="http://www.webserviceX.net/globalweather.asmx" />
  </wsdl:port>
  <wsdl:port name="GlobalWeatherSoap12" binding="tns:GlobalWeatherSoap12">
    <soap12:address location="http://www.webserviceX.net/globalweather.asmx" />
  </wsdl:port>
  <wsdl:port name="GlobalWeatherHttpGet" binding="tns:GlobalWeatherHttpGet">
    <http:address location="http://www.webserviceX.net/globalweather.asmx" />
  </wsdl:port>
  <wsdl:port name="GlobalWeatherHttpPost" binding="tns:GlobalWeatherHttpPost">
    <http:address location="http://www.webserviceX.net/globalweather.asmx" />
  </wsdl:port>
</wsdl:service>
</wsdl:definitions>

```

- KDSOAP
  - Get WSDL
  - generate code from WSDL
  - Compile code
- Pro:
  - Well defined interface
- Cons:
  - Tricky, slow/verbose, transfer of binary data, ...

- JSON

- JavaScript Object Notation

- Object: {}

- Array: []

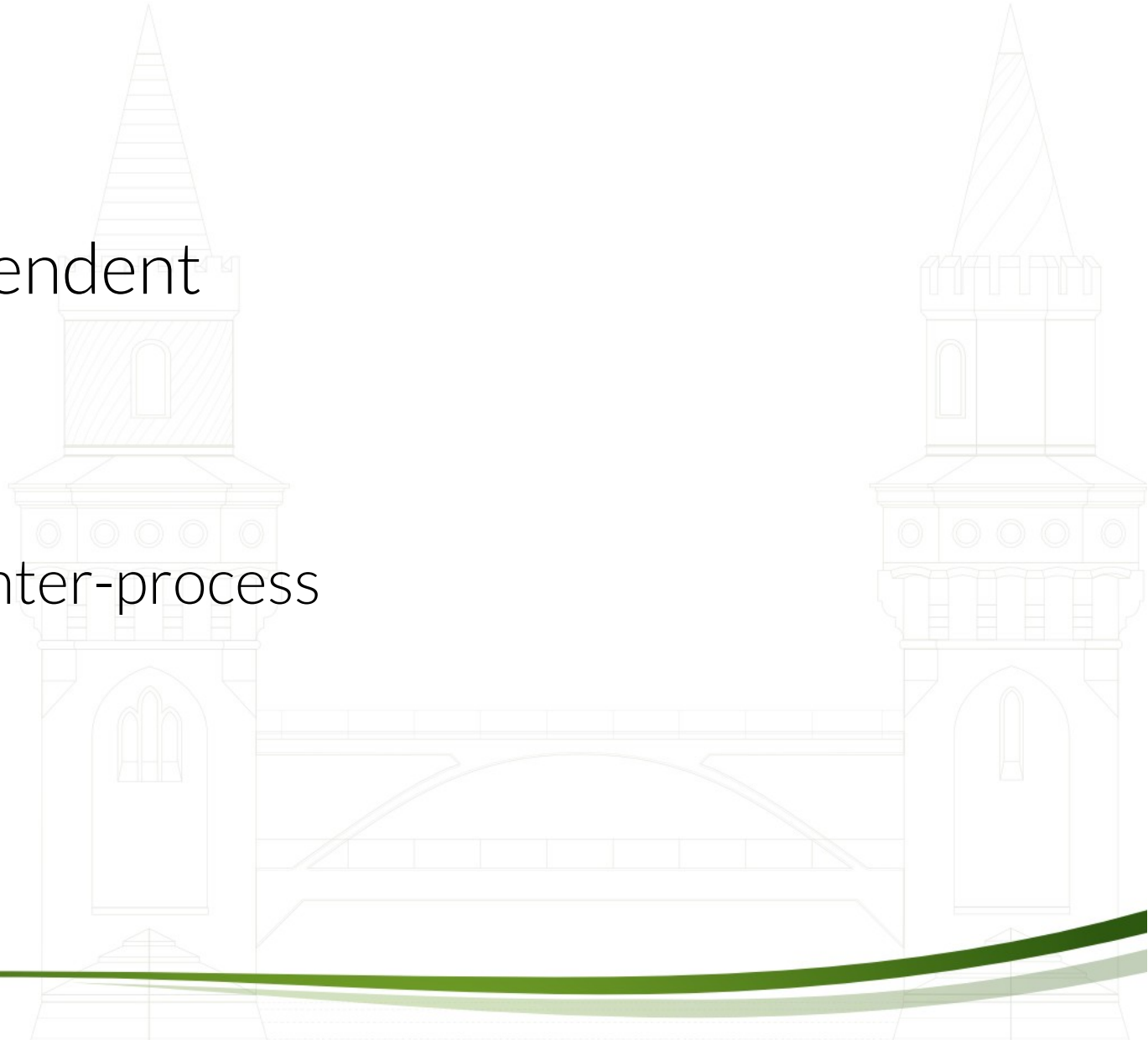
- Primitive types: number, string, bool (true, false), null

- Easy to parse

- Human readable

```
1 {  
2   "first": "John",  
3   "last": "Doe",  
4   "age": 39,  
5   "sex": "M",  
6   "salary": 70000,  
7   "registered": true,  
8   "favorites": {  
9     "color": "Blue",  
10    "sport": "Soccer",  
11    "food": "Spaghetti"  
12  }  
13 }
```

- Lightweight
- Stateless
- Transport independent
  - Raw sockets
  - HTTP
  - Same process, inter-process



- REpresentational State Transfer
- RESTful constraints:
  - Client-server
  - Stateless
  - Cacheable
  - Layered system
  - Code on demand (optional)
  - Uniform interface

- Re-use v's extension
  - REST uses HTTP verbs:
    - GET
    - POST
    - PUT
    - DELETE
  - e.g. SOAP extends with arbitrary methods
- Resources



	GET	POST	PUT	DELETE
<a href="http://foo.com/images/">http://foo.com/images/</a> Collection resource	Get all images in the set	Create a new resource in the set	Replace all resources in the set	Remove all images in the set
<a href="http://foo.com/images/1">http://foo.com/images/1</a> Element resource	Get image '1'	?	Replace image '1'	Delete image '1'

- Type of data
  - Resource dependent
  - <http://foo.com/images/1> → image/png
  - XML
  - JSON
  - HTML
  - ...

- Structured data as JSON
- eval in JavaScript context
- JSONP
  - Workaround for same origin policy
  - Supply a function e.g. foo()
  - Returned JSON will be e.g. `foo && foo( { ... } )`

- blipfoto
  - social photography website
  - RESTful/JSON API
  - View images
  - Upload images, blog
  - Rate images
  - Leave comments

- QNetworkAccessManager
- XML
  - DOM
  - Streams
- JSON
  - Qt4: qjson, ...
  - Qt5: builtin



- We're done, right?
- Authentication
  - XAUTH
  - OAUTH
  - HMAC
- Encryption
  - SSL



- Home-rolled
  - Dying out
- OAUTH
  - 1.0: started in 2006, RFC 5849 published 2010
  - 2.0: RFC 6749 published 2012

- What is OAUTH?

- Method of granting access to resources to a client that is not the resource owner
- e.g. allow a photo printing application to access your images stored online
- No need to give credentials to client (printing application)



- OAUTH is fiddly; read closely!
- Additional tools:
  - SHA1, SHA256
  - HMAC
  - MD5
  - nonce (would be so much better as n-once)
  - SHA1, MD5 already in Qt

## • HMAC: Hash-based Message Authentication Code

```
1 template < typename HASH >
2 struct HMAC
3 {
4     static QByteArray create( QByteArray secret, QByteArray message )
5     {
6         if ( secret.size() > HASH::block_size )
7             secret = HASH::hash( secret );
8
9         QByteArray opad( HASH::block_size, 0x5c );
10        QByteArray ipad( HASH::block_size, 0x36 );
11
12        for ( int i=0; i<secret.size(); ++i )
13        {
14            ipad.data()[i] ^= secret.data()[i];
15            opad.data()[i] ^= secret.data()[i];
16        }
17
18        return HASH::hash( opad.append( HASH::hash( ipad.append( message ) ) ) );
19    }
20
21    static QByteArray name() { return "HMAC-" + HASH::name(); }
22 };
```

- nonce

```
1 QByteArray generateNonce()
2 {
3     static bool static_initialized = false;
4     if ( !static_initialized )
5     {
6         qsrand( QDateTime::currentDateTime().toTime_t() );
7         static_initialized = true;
8     }
9
10    QByteArray result( QByteArray::number( QDateTime::currentDateTime().toTime_t() ) +
11                      QByteArray::number( qrand() ) );
12    result = QCryptographicHash::hash( result, QCryptographicHash::Md5 );
13    result += result;
14    result = result.toHex().mid( 0, 43 );
15
16    return result;
17 }
```

- Steps:

- Register client with webservice: shared secret and ID
- Use secret and ID to get temp. token
- Redirect to verification site → user puts in credentials
- Callback URL called
- Use information to get full tokens
- Access service!

- XAUTH

- Similar to OAUTH
- Avoids need for a validation/callback URL
- → typically pass username/password
- → typically only for 'official' applications

- Every Webservice is different
- Read the supplied documentation closely
  - Wireshark/sniffer
- Popular: facebook, twitter, ...
- Examples:
  - Blipfoto
  - Twitter

```
1 void Blipfoto::signin( const QString& user, const QString& pass )
2 {
3     QNetworkReply* reply = NULL;
4     QUrl url;
5
6     // 1. get timestamp
7     QString timestamp = QString::number( getTimestamp() );
8     if ( timestamp.isEmpty() )
9     {
10        qDebug() << "Blipfoto::signin: failed to get timestamp";
11        return;
12    }
13
14    // 2. create signature for the call to get identity token
15    QString nonce = createNonce();
16    QString sig = QCryptographicHash::hash(
17        (nonce + timestamp + API_SECRET).toLatin1(),
18        QCryptographicHash::Md5 ).toHex();
19
20    // 3. make call
21    ParameterList params;
22    params << Parameter( "api_key", API_KEY )
23        << Parameter( "format", "XML" )
24        << Parameter( "nohttperror", "1" )
25        << Parameter( "pass", pass )
26        << Parameter( "email", user.toUtf8().toPercentEncoding() )
27        << Parameter( "nonce", nonce )
28        << Parameter( "timestamp", timestamp )
29        << Parameter( "signature", sig );
30
31    url = QUrl::fromEncoded( (BASE_URL + "/get/trusttoken/?%1").arg( params.join() ).toUtf8() );
32    reply = nam().get( QNetworkRequest( url ) );
33    QVariantMap m = makeBlockingRPCRequest( reply, new IdentityTokenProcessor ).toMap();
```

```
1 void Twitter::startAuthentication()
2 {
3     if ( authenticated() )
4     {
5         qDebug() << "Twitter::startAuthentication: already authenticated";
6         return;
7     }
8
9     // 1. obtain a request token
10    URLArgumentMap args;
11    args[ "oauth_callback" ] = CALLBACK;
12    args[ "oauth_consumer_key" ] = CONSUMER_KEY;
13    args[ "oauth_nonce" ] = generateNonce();
14    args[ "oauth_signature_method" ] = HMAC< SHA1Functor >::name();
15    args[ "oauth_timestamp" ] = QByteArray::number( QDateTime::currentDateTime().toTime_t() );
16    args[ "oauth_version" ] = "1.0";
17
18    QNetworkRequest r = createRequest( "POST",
19                                     "https://api.twitter.com/oauth/request_token",
20                                     args,
21                                     URLArgumentMap(),
22                                     CONSUMER_SECRET );
23    QNetworkReply* reply = nam().post( r, QByteArray() );
24
25    impl_>listener.reset( new QNetworkReplyListener( reply ) );
26    connect( impl_>listener.data(), SIGNAL( complete( const QByteArray& ) ),
27            impl_, SLOT( onRequestToken( const QByteArray& ) ) );
28    connect( impl_>listener.data(), SIGNAL( failed() ),
29            impl_, SLOT( onRequestTokenFailed() ) );
30 }
```



```
1 // create a request
2 QNetworkRequest createRequest( const QByteArray& method,
3                               const QByteArray& uri,
4                               URLArgumentMap args,
5                               const URLArgumentMap& payload = URLArgumentMap(),
6                               const QByteArray& consumerSecret = QByteArray(),
7                               const QByteArray& tokenSecret = QByteArray() )
8 {
9     // 0. create base
10    QByteArray signatureBaseString( createSignatureBaseString( method, uri, args, payload ) );
11
12    // 1. HMAC-SHA1, stash in args
13    QByteArray signature( createSignature( consumerSecret, tokenSecret, signatureBaseString ) );
14    args[ "oauth_signature" ] = signature;
15
16    // 2. build authorization header
17    QByteArray header = "OAuth ";
18    for ( int i=0; i<args.size(); ++i )
19    {
20        if ( i != 0 )
21            header += ", ";
22
23        QByteArray key = args.keys().at( i );
24        header += key.toPercentEncoding() + "=\"" + args[ key ].toPercentEncoding() + "\"";
25    }
26
27    QNetworkRequest request = QNetworkRequest( QUrl( uri ) );
28    request.setRawHeader( "Authorization", header );
29
30    return request;
31 }
```

```
1 // from http://dev.twitter.com/pages/auth#intro
2 QByteArray createSignatureBaseString( const QByteArray& method,
3                                       const QByteArray& uri,
4                                       URLArgumentMap args,
5                                       const URLArgumentMap& payload )
6 {
7     QByteArray result = method;
8     result += '&';
9     result += uri.toPercentEncoding() + '&';
10
11     QByteArray parameters;
12
13     // join payload on
14     args.unite( payload );
15
16     // sort keys
17     QList< QByteArray > sortedKeys = args.keys();
18     qStableSort( sortedKeys );
19
20     // foreach key, add key & value
21     for ( int i=0; i<sortedKeys.size(); ++i )
22     {
23         if ( i != 0 )
24             parameters += "&";
25
26         parameters += sortedKeys.at( i ).toPercentEncoding() + "=" + args[ sortedKeys.at( i ) ].toPercentEncoding();
27     }
28     result += parameters.toPercentEncoding();
29
30     return result;
31 }
32
33 QByteArray createSignature( const QByteArray& consumerSecret, const QByteArray& tokenSecret, const QByteArray& data )
34 {
35     return HMAC< SHA1Functor >::hash( consumerSecret + '&' + tokenSecret, data ).toBase64();
36 }
```

- Why?
  - Standards, standards, standards
  - Flexible
  - Avoids lock-in
- How?
  - Introspection
  - QMetaObject system



- Invokable methods: `Q_INVOKABLE`
- Parameters, types: `QMetaMethod`
- Invoke methods by name
- JSON → `QVariant`
- `QNetworkAccessManager`
- `QTcpServer`
- `QObject` subclass as a 'callable'

- Scope
  - Embedded
  - Pull from application
  - Not amazon
- Authentication?
  - OAUTH, XAUTH if required
  - SSL & username, password

- WebServices are very widespread
- Simple
  - Fiddly, maybe!
- Flexibility

Thank You!

- JSON: <http://json.org>
- JSON-RPC: <http://www.jsonrpc.org/>
- REST:
  - [http://www.ics.uci.edu/~fielding/pubs/dissertation/rest\\_arch\\_style.htm](http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm)
  - [http://en.wikipedia.org/wiki/Representational\\_state\\_transfer](http://en.wikipedia.org/wiki/Representational_state_transfer)
- HMAC:
  - <http://tools.ietf.org/html/rfc2104>
  - [http://en.wikipedia.org/wiki/Hash-based\\_message\\_authentication\\_code](http://en.wikipedia.org/wiki/Hash-based_message_authentication_code)