CONSUMING WEB SERVICES

Web Services

- Expose data and functionality to clients on the Internet via HTTP
- Provide a "Web API" intended for consumption by programs

Using Web Services

Supply Input to Web Service via:

- URL (Path / Query String)
- Request Headers
- Request Body

Output from Web Service:

- Response code
- Response Headers
- Response Body

Sample Web Service

- https://www.dallasopendata.com/Public-Safety/Dallas-Police-Active-Calls/9fxf-t2tr

Exploring Web Services

Tools:

- Command line HTTP clients (curl, wget)
- Browser Addons (Chrome: Postman)
- Online API Testing Sites

Types of Web Service APIs

- XML/RPC
- SOAP
- REST

XML/RPC

- Lightweight XML message format
- RPC - Remote Procedure Call
- An XML/RPC message represents an invocation of a method in a class typically hosted in an application server container

Sample XML/RPC Traffic

Request

```
<?xml version="1.0"?>
<methodCall>
  <methodName>examples.getStateName</methodName>
  <params>
    <param>
      <value><i4>41</i4></value>
    </param>
  </params>
</methodCall>
```

Response

```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value><string>South Dakota</string></value>
    </param>
  </params>
</methodResponse>
```

SOAP

- Based on XML/RPC
- Popular in early 2000's
- Heavyweight message format designed to be generated/consumed by SOAP RPC libraries
- SOAP messages often represent invocations of methods in a server-side class hosted by an application server

Sample SOAP Traffic

Request

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
     xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <ns1:getBalance
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:ns1="http://atmsvc">
   <acct xsi:type="xsd:string">123</acct>
  </ns1:getBalance>
 </soapenv:Body>
</soapenv:Envelope>
```

Response

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <ns1:getBalanceResponse
      soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:ns1="http://atmsvc">
   <getBalanceReturn xsi:type="xsd:double">1500.0</getBalanceReturn>
  </ns1:getBalanceResponse>
 </soapenv:Body>
</soapenv:Envelope>
```

REST

- Architecture for designing web services
- An approach to design, not a standard
- Message format typically JSON

REST

- REST services typically
  - Expose collections of "resources"
  - Provide Create/Retrieve/Update/Delete functionality
- HTTP verb controls what action is performed by a given request
  - See http://www.restapitutorial.com/lessons/httpmethods.html

Node.js HTTP Clients

The request module

- Use the **request** module on npmjs.org to reduce the effort to write HTTP clients

```
var request = require('request');

request("http://localhost:8000/",
  function(error, response, body) {
    if (!error && response.statusCode == 200) console.log(body);
  });
```

request options

The first parameter can be a URL or an object that specifies the URL, method, and other data to use in the HTTP request:

```
var options = {
    url: 'https://www.reddit.com/r/funny.json',
    method: 'GET',
    headers: {
        'Accept': 'application/json',
        'Accept-Charset': 'utf-8'
    }
};

request(options, function(err, res, body) {
   console.log(body);
});
```

Two Approches to Query Strings

1. Build query string yourself

```
var first = "Freddy", last = "O'Brien";

var url = "http://blah.com/foo?" +
   "fname=" + encodeURIComponent(first) +
  "&lname=" + encodeURIComponent(last);

request(url, function(err, res, body) { ... });
```

2. Let request module build it

```
var first = "Freddy", last = "O'Brien";

 request({
   url: "http://blah.com/foo",
   qs: { fname: first, lname: last }
}, function(err, res, body) { ... });
```

POST with request

The **request** module supports POST:

```
request({
  uri : "http://localhost:8000/",
  method : "POST",
  form : {
    foo : "bar",
    baz : "blah"
  }
}, function(error, response, body) {
  console.log(body);
});
```

Downloading HTTP Content

- The request object is a stream that supports piping
- Download a file:

```
var req = request("http://foo.com/bar.txt");
req.pipe(fs.createWriteStream('bar.txt'));
```

Streams and Pipes

- Streams can be composed in a pipeline using pipe()
- *readstream*.pipe(*writestream*)

  causes Node to stream data read from *readstream* to *writestream*
- Example: Copy a file

```
var rdstrm = fs.createReadStream("data.txt")
var outstrm = fs.createWriteStream("copy.txt")
rdstrm.pipe(outstrm)
```

Error Handling with Streams

- Errors during stream processing raise the **error** event
- If unhandled, an exception is thrown that terminates execution
- Example: Copy a file, handling errors

```
fs.createReadStream("missing-file.txt")
    .on('error', function(err) {
        console.log("Uh oh:", err);
    })
    .pipe(fs.createWriteStream('copy.txt'));
```

Downloading HTTP Content

- Download a file with error handling:

```
request("http://foo.com/bar.txt")
  .on('error', function(err) {
      console.log("Uh oh:", err);
  })
  .pipe(fs.createWriteStream('bar.txt'));
```