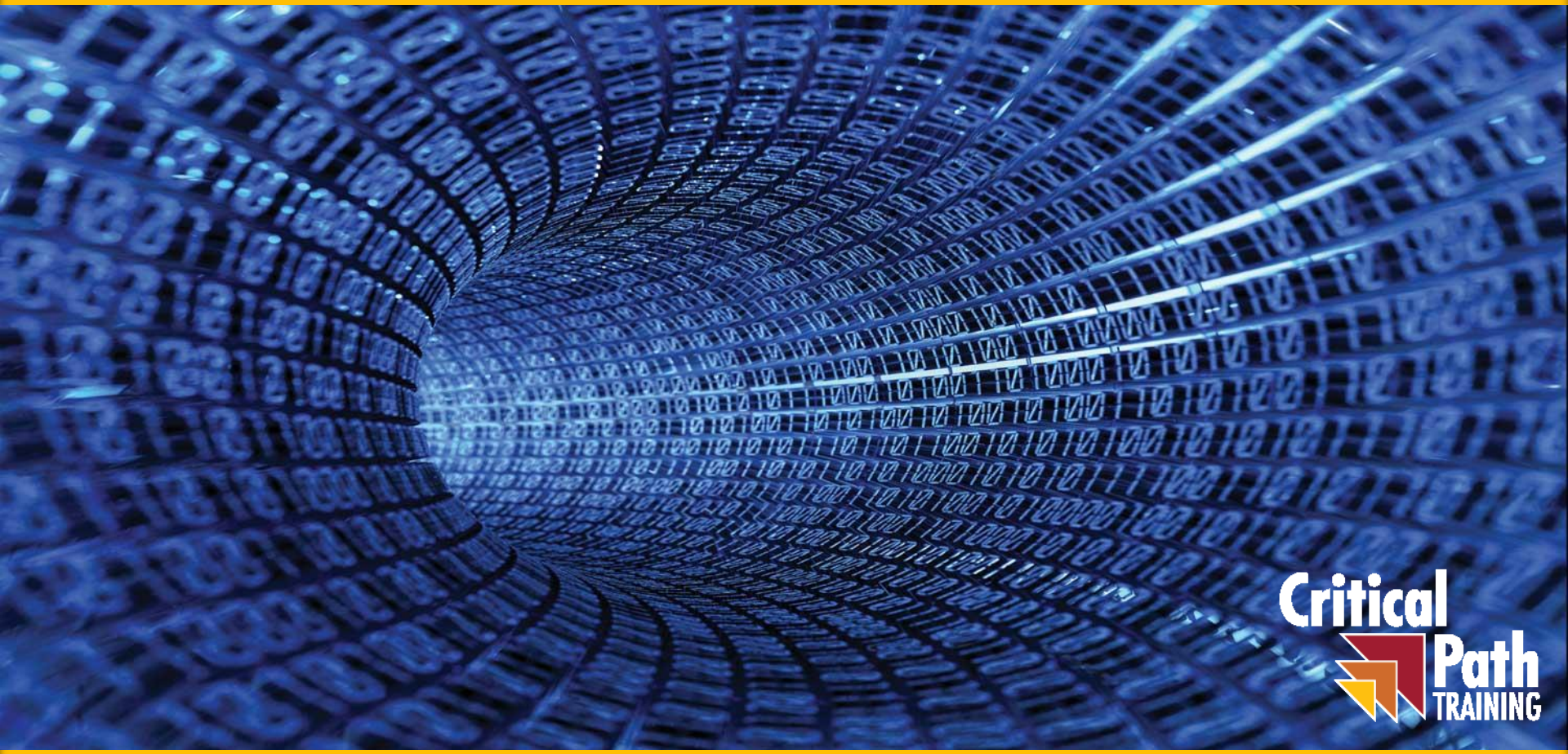
Programming the SharePoint REST API



Agenda

- The SharePoint REST API
- Creating REST URIs for SharePoint Objects
- Consuming OData Results from SharePoint
- Paging SharePoint List Items
- Adding and Updating Items

RESTful Web Services

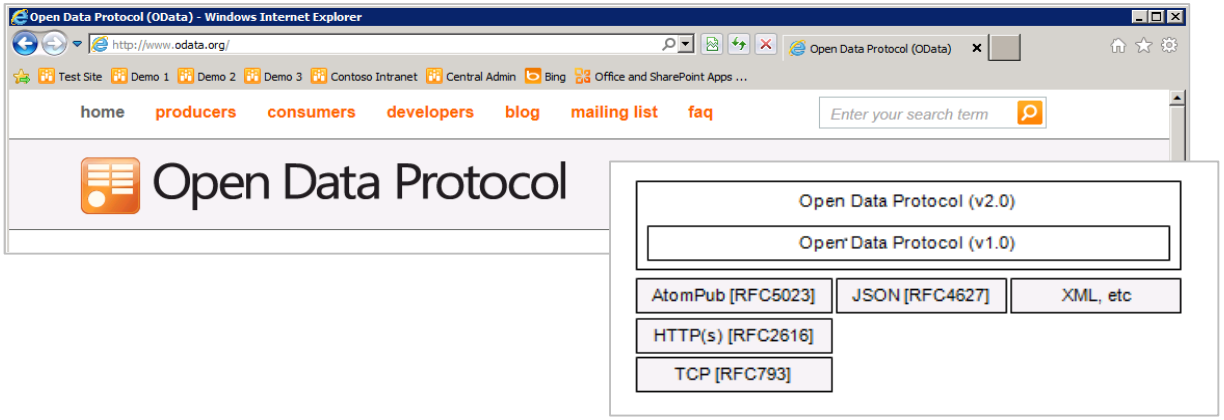
- RESTful Web Service
  - implemented using the principles of REST
  - REST URI = [base URI] + [resource path] + [query options]
  - Calls based on standard HTTP verbs (**GET**, **POST**, **PUT**, **DELETE**)
  - Passes data to and from client using representations
  - Can be designed to implement custom APIs and/or standard APIs

- Data passed across network using representations
  - Representations model resources – but they're different
  - Based on common formats: HTML, XML, ATOM and JSON
  - Based on specific Internet media types

OData Primer

- What is OData?
  - A standardized REST API interface for common CRUD operations
  - Defined by Open Data Protocol specification
  - OData services becoming more popular on Internet (e.g. NetFlix)
  - SharePoint 2010 introduced a REST API for dealing with list items
  - SharePoint 2013 introduces new and expanded REST API

for an excellent resource go to
http://www.odata.org



OData Query Option Parameters

- **$select**
  - http://services.odata.org/OData/OData.svc/Products?$select=Price,Name
- **$filter**
  - http://services.odata.org/OData/OData.svc/Products?$filter=startswith(CompanyName, 'Alfr')
- **$orderby**
  - http://services.odata.org/OData/OData.svc/Products?$orderby=Rating
- **$top**
  - http://services.odata.org/OData/OData.svc/Products?$top=5
- **$skip**
  - http://services.odata.org/OData/OData.svc/Products?$skip=5
  - http://services.odata.org/OData/OData.svc/Products?$skip=5&$top=5
- **$expand**
  - http://services.odata.org/OData/OData.svc/Categories?$expand=Products

Using the $filter Parameter

Logical Operators		
Eq	Equal	`/Suppliers?$filter=Address/City eq 'Las Vegas'`
Ne	Not equal	`/Suppliers?$filter=Address/City ne 'London'`
Gt	Greater than	`/Products?$filter=Price gt 20`
Ge	Greater than or equal	`/Products?$filter=Price ge 10`
Lt	Less than	`/Products?$filter=Price lt 20`
Le	Less than or equal	`/Products?$filter=Price le 100`
And	Logical and	`/Products?$filter=Price le 200 and Price gt 3.5`
Or	Logical or	`/Products?$filter=Price le 3.5 or Price gt 200`
Not	Logical negation	`/Products?$filter=not endswith(Description,'milk')`
Arithmetic Operators		
Add	Addition	`/Products?$filter=Price add 5 gt 10`
Sub	Subtraction	`/Products?$filter=Price sub 5 gt 10`
Mul	Multiplication	`/Products?$filter=Price mul 2 gt 2000`
Div	Division	`/Products?$filter=Price div 2 gt 4`
Mod	Modulo	`/Products?$filter=Price mod 2 eq 0`
Grouping Operators		
( )	Precedence grouping	`/Products?$filter=(Price sub 5) gt 10`

$filter Parameter String Functions

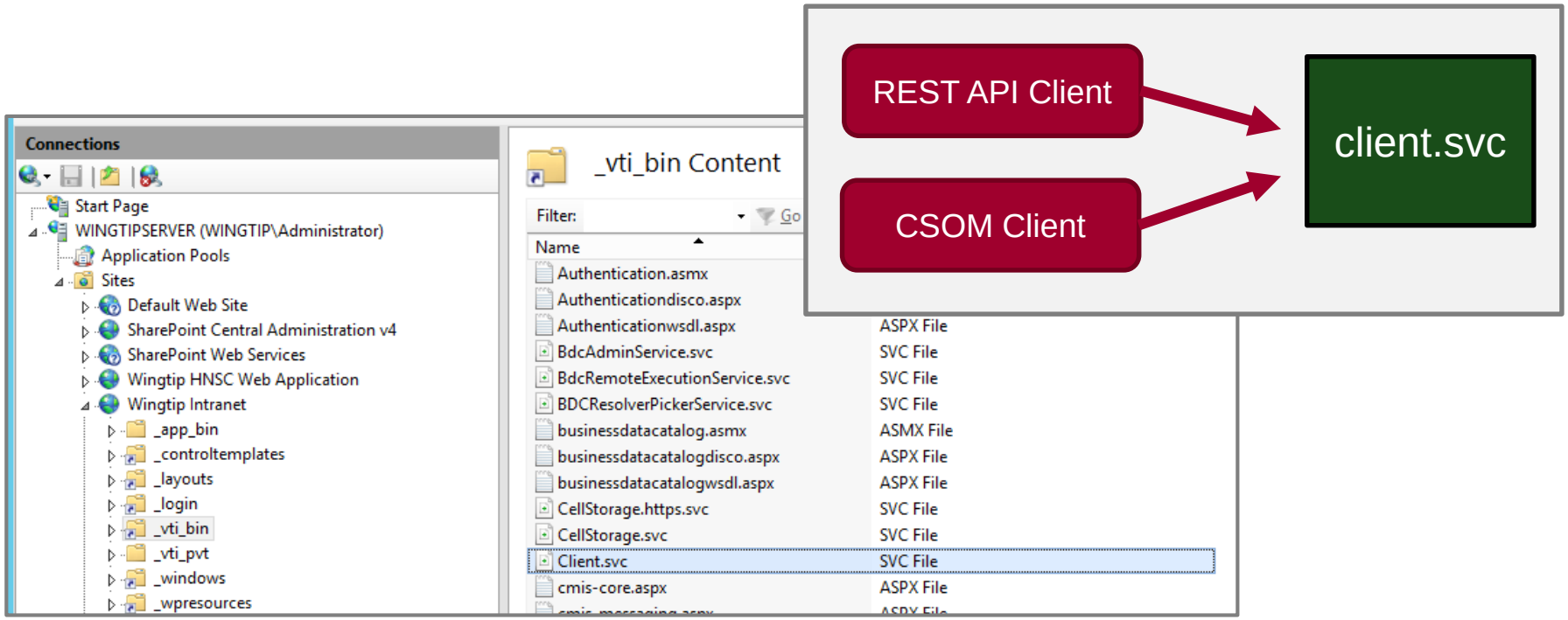
String Functions	
bool substringof(string po, string p1)	Customers?$filter=substringof('Alfreds', CompanyName) eq true
bool endswith(string p0, string p1)	Customers?$filter=endswith(CompanyName, 'Futterkiste') eq true
bool startswith(string p0, string p1)	Customers?$filter=startswith(CompanyName, 'Alfr') eq true
int length(string p0)	Customers?$filter=length(CompanyName) eq 19
int indexof(string p0, string p1)	Customers?$filter=indexof(CompanyName, 'lfreds') eq 1
string replace(string p0, string find, string replace)	Customers?$filter=replace(CompanyName, ' ', '') eq 'AlfredsFutterkiste'
string substring(string p0, int pos)	Customers?$filter=substring(CompanyName, 1) eq 'lfreds Futterkiste'
string substring(string p0, int pos, int length)	Customers?$filter=substring(CompanyName, 1, 2) eq 'lf'
string tolower(string p0)	Customers?$filter=tolower(CompanyName) eq 'alfreds futterkiste'
string toupper(string p0)	Customers?$filter=toupper(CompanyName) eq 'ALFREDS FUTTERKISTE'
string trim(string p0)	Customers?$filter=trim(CompanyName) eq 'Alfreds Futterkiste'
string concat(string p0, string p1)	Customers?$filter=concat(concat(City, ', '), Country) eq 'Berlin, Germany'

Remote Communications with SharePoint

- In SharePoint 2003 and SharePoint 2007
  - SOAP-based web services (e.g. Lists.asmx)
- In SharePoint 2010
  - Client-side Object Model (CSOM)
  - REST API for list items accessible through **ListData.svc**
- In SharePoint 2013
  - Expanded CSOM Support
  - New SharePoint REST API replaces **ListData.svc**
- In SharePoint 2016
  - REST API improved with greater support for ODATA 4.0

SharePoint REST API Architecture

- REST API entry point is client.svc
  - In SharePoint 2010, client.svc only used by CSOM
  - In SharePoint 2013, client.svc used by CSOM and REST API



SharePoint REST URLs and the _api Alias

- SharePoint REST API provides _api alias
  - The **_api** alias maps to **_vti_bin/client.svc**
  - Alias used to make SharePoint REST API URLs cleaner
  - Alias serves to decouple URLs from underlying architecture
- This URL works but it is not recommended
  - **http://intranet.wingtip.com/_vti_bin/client.svc/web**
- SharePoint REST API URLs should be created with _api
  - **http://intranet.wingtip.com/_api/web**

Anatomy of a SharePoint REST URL

- SharePoint REST made up of three parts
  - Base URI

    `http://intranet.wingtip.com/_api`
  - Target SharePoint Object

    `web`
  - Query String Parameter options

    `?$select=Id,Title,MasterUrl`

```
http://intranet.wingtip.com/_api/web/?$select=Id,Title,MasterUrl
```

Mapping SharePoint Objects to URLs

SharePoint Object	Object mapping
Site Collection	`site`
Site	`web`
Lists collection	`web/lists`
List by ID	`web/lists(guid'402cd788-9c5c-4931-92d6-09f18efb368c')`
List by Title	`web/lists/getByTitle('Customers')`
List property	`web/lists/getByTitle('Customers')/Title`
List items collection	`web/lists/getByTitle('Customers')/items`
List item	`web/lists/getByTitle('Customers')/items(1)`
List item property	`web/lists/getByTitle('Customers')/items(1)/FirstName`

Testing REST Calls Through the Browser

http://intranet.wingtip.com/_api/web/?$select=Id,Title,MasterUrl

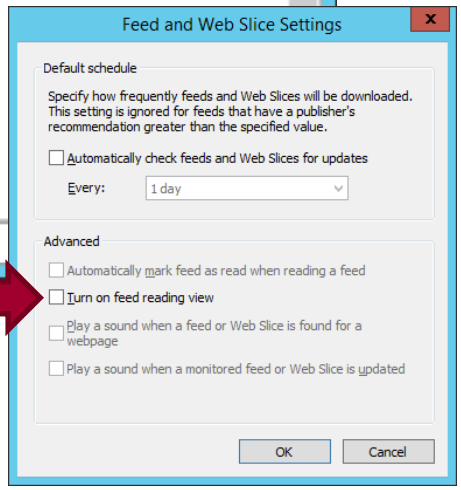
intranet.wingtip.com

File Edit View Favorites Tools Help

Wingtip Intranet | Search Center | BI Center | Publishing Site | Central Admin


```
<?xml version="1.0" encoding="utf-8" ?>
- <entry xml:base="http://intranet.wingtip.com/_api/" xmlns="http://www.w3.org/2005/Atom"
    xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
    xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
    xmlns:georss="http://www.georss.org/georss" xmlns:gml="http://www.opengis.net/gml">
    <id>Web</id>
    <category term="SP.Web" scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
    <link rel="edit" href="Web" />
    <title />
    <updated>2014-10-22T18:16:04Z</updated>
  - <author>
      <name />
    </author>
  - <content type="application/xml">
    - <m:properties>
        <d:Id m:type="Edm.Guid">1bbbc073-d181-4135-9893-75004ba752fb</d:Id>
        <d:MasterUrl>/_catalogs/masterpage/seattle.master</d:MasterUrl>
        <d:Title>Wingtip Intranet</d:Title>
      </m:properties>
    </content>
  </entry>
```


It can be helpful to disable the Internet Explorer option feed reading view



OData Support in SharePoint 2016

- SharePoint Online supports ODATA v4.0
  - OData v4.0 support added in December of 2014
- SharePoint 2013 On-premises supports ODATA v3.0
  - SharePoint 2013 OOB only supports verbose metadata format
  - PowerShell script must be run to enable all ODATA formats
- SharePoint 2016 On-premises supports ODATA v4.0
  - SharePoint 2016 OOB supports all ODATA formats

ODATA Formats and the Accept Header

- Verbose (aka Full Metadata)

  `accept: application/json; odata=verbose`

- Minimal Metadata

  `accept: application/json; odata=minimalmetadata`

  `accept: application/json`

- No Metadata

  `accept: application/json; odata=nometadata`

Agenda

- ✓ The SharePoint REST API
- ➢ Creating REST URIs for SharePoint Objects
- Consuming OData Results from SharePoint
- Paging SharePoint List Items
- Adding and Updating Items

Service Root URI of the App Web

- Creating the App Web's Service Root URI
  - Use URL relative to **Pages** folder

```
var restURI = "../_api/web/?$select=Id,Title,Url"
```

  - Use URL created from **SPAppWebUrl** query string parameter

```
var restURI = getQueryStringParameter("SPAppWebUrl") +
              "/_api/web/?$select=Id,Title,Url"
```

  - Use URL created from **_spPageContextInfo.webAbsoluteUrl**

```
var restURI = _spPageContextInfo.webAbsoluteUrl +
              "/_api/web/?$select=Id,Title,Url"
```

Finding the Service Root of the Host Web

		Will it ever work?	Will it work when host web is not at top of domain	Will it work when app installed at tenancy scope
	`var restURI = getQueryStringParameter("SPHostUrl") + "/_api/web/?$select=Id,Title,Url";`	No	No	No
	`var restURI = "/_api/web/?$select=Id,Title,Url";`	Yes	No	No
	`var restURI = "../../_api/web/?$select=Id,Title,Url"`	Yes	Yes	No

	`var appWebUrl = getQueryStringParameter("SPAppWebUrl");`
	`var appWebUrlLength = appWebUrl.length;`
	`var appSuffix = "/AppsGoneWild";`
	`var appSuffixLength = appSuffix.length;`
	`var hostWebUrlTranslated = appWebUrl.substring(0, (appWebUrlLength - appSuffixLength));`
	
	`var restURI = hostWebUrlTranslated + "/_api/web/?$select=Id,Title,Url";`
		Yes	Yes	No

	`var restURI = "../_api/SP.AppContextSite(@target)/web/" + "?$select=Id,Title,Url" + "&@target='" + getQueryStringParameter("SPHostUrl") + "'";`	Yes	Yes	Yes

Reliable URIs for SharePoint REST Calls

- For the app web

```
var restURI = "../_api/web/?$select=Id,Title,Url"
```

- For the host web

```
var restURI = "../_api/SP.AppContextSite(@target)/web/" +
              "?$select=Id,Title,Url" +
              "&@target='" + getQueryStringParameter("SPHostUrl") + "'";
```

Agenda

- ✓ Creating REST URIs for SharePoint Objects
- ➢ Consuming OData Results from SharePoint
- Paging SharePoint List Items
- Adding and Updating Items

Querying a List in the App Web

```
var getCustomers = function () {

  var requestUri = "../_api/web/lists/getByTitle('Customers')/items" +
                   "?$select=ID,FirstName,Title,Company,WorkPhone,HomePhone,Email" +
                   "&$orderby=Title,FirstName";

  // send call across network
  return $.ajax({
    url: requestUri,
    headers: { "accept": "application/json;odata=verbose" }
  });

};
```

Querying for Lists within the Host Web

```
var getLists = function () {

  var requestUri = "../_api/SP.AppContextSite(@target)/web/lists/" +
                   "?$filter=(Hidden eq false) and (BaseType eq 0)" +
                   "&$select=Id,Title, ItemCount, EntityTypeName, ListItemEntityTypeFullName" +
                   "&@target='" + hostWebUrl + "'";

  return $.ajax({
    url: requestUri,
    type: "GET",
    headers: { "accept": "application/json;odata=verbose" }
  });

};
```

Using the $expand Query Option

- **$expand** used to create more efficient code
  - Deferred content held back by default
  - **$expand** used to retrieve results with deferred content
  - Effectively reduces round trips

```
var getSiteGroups = function () {

  var requestUri = "../_api/SP.AppContextSite(@target)/web/siteGroups/" +
                   "?$expand=Users" +
                   "&@target='" + hostWebUrl + "'";;

  return $.ajax({
    url: requestUri,
    type: "GET",
    headers: { "accept": "application/json;odata=verbose" },
  });

};
```

DEMO

- Ted Pattison

HostWebExplorer App

Agenda

- ✓ Creating REST URIs for SharePoint Objects
- ✓ Consuming OData Results from SharePoint
- ➢ Paging SharePoint List Items
- • Adding and Updating Items

Paging with SharePoint Lists

- SharePoint does not support **$skip** for list items
  - You cannot create typical OData paging scheme with a SharePoint list
- What do you do instead?
  - Create a custom paging scheme using **$filter**
  - Create a paging scheme using **$skiptoken**

DEMO

Paging with SharePoint List Items

Agenda

- ✓ The SharePoint REST API
- ✓ Creating REST URIs for SharePoint Objects
- ✓ Consuming OData Results from SharePoint
- ✓ Paging SharePoint List Items
- ➢ Adding and Updating Items

Updating SharePoint Objects

- All write operations must pass valid request digest value
- You must include type metadata for inserts & updates
- Sometimes you must pass ETags for updates & deletes

Understanding the Request Digest

- All SharePoint write operations require Request Digest
  - Provides security mechanism to protect again replay attacks
  - Request digest known to SharePoint old timers as “Form Digest”
  - SharePoint adds request digest element with ID **__REQUESTDIGEST**
  - Request digest value passed using **X-RequestDigest** header

```
var requestHeaders = {
  "accept": "application/json;odata=verbose",
  "X-RequestDigest": $("#__REQUESTDIGEST").val()
}
```

Caching the Request Digest

- Request digest queried using **/_api/contextinfo**

```
Wingtip.Customers.DataAccess = function () {

  var requestDigest;

  var initialize = function () {

    var deferred = $.ajax({
      url: "../_api/contextinfo",
      type: "POST",
      headers: { "accept": "application/json;odata=verbose" }
    })

    deferred.then(function (data) {
      requestDigest = data.d.GetContextWebInformation.FormDigestValue
    });

  }
```

Working with List Item Type Metadata

- Each SharePoint list has a unique type for its list items

```
{
  __metadata
    etag="1"
    id=02d4852e-43b4-4023-9706-07358b789674
    type=SP.Data.CustomersListItem
```

- **type** value must be passed with all inserts and updates

```
var customerData = {
  __metadata: { "type": "SP.Data.CustomersListItem" },
  Title: LastName,
  FirstName: FirstName,
  Company: Company,
  WorkPhone: WorkPhone,
  HomePhone: HomePhone,
  Email: Email
};
```

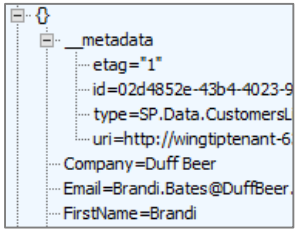
- **type** discoverable using **ListItemEntityTypeFullName** property

Adding a SharePoint List Item

```
var addCustomer = function (FirstName, LastName, Company, WorkPhone, HomePhone, Email) {

  var requestUri = "../_api/web/lists/getByTitle('Customers')/items";

  var requestHeaders = {
    "accept": "application/json;odata=verbose",
    "X-RequestDigest": $("#__REQUESTDIGEST").val()
  }

  var customerData = {
    __metadata: { "type": "SP.Data.CustomersListItem" },
    Title: LastName,
    FirstName: FirstName,
    Company: Company,
    WorkPhone: WorkPhone,
    HomePhone: HomePhone,
    Email: Email
  };

  var requestBody = JSON.stringify(customerData);

  return $.ajax({
    url: requestUri,
    type: "POST",
    contentType: "application/json;odata=verbose",
    headers: requestHeaders,
    data: requestBody,
  });

};
```

ETags and Optimistic Concurrency

- OData v2 requires items to carry ETags
  - ETag is integer value in that it identities version of item
  - ETag is automatically incremented with each update



- ETag use to support for optimistic concurrency control
  - ETag works to eliminate the “lost update” scenario
  - ETag must be tracked in order to post updates in most scenarios

```
// store item metadata values into hidden controls
$("#customer_id").val(data.d.ID);
$("#etag").val(data.d.__metadata.etag);
```

ETags and the If-Match Header

- Update and Delete operations require If-Match Header
  - Allows you to pass ETag value during an update
  - Update fails if ETag value changed due to update by other user

```
var requestHeaders = {
  "accept": "application/json;odata=verbose",
  "X-HTTP-Method": "MERGE",
  "X-RequestDigest": $("#__REQUESTDIGEST").val(),
  "If-Match": ETag
}
```

- You can pass wildcard (*) value inside If-Match Header
  - Done to disable optimistic concurrency control
  - This is commonly done with delete operations

```
var requestHeaders = {
  "accept": "application/json;odata=verbose",
  "X-RequestDigest": $("#__REQUESTDIGEST").val(),
  "If-Match": "*"
}
```

Updating a SharePoint List Item

```
var updateCustomer = function (Id, FirstName, LastName, Company, WorkPhone, HomePhone, Email, ETag) {

  var requestUri = "../_api/web/lists/getByTitle('Customers')/items(" + Id + ")";

  var requestHeaders = {
    "accept": "application/json;odata=verbose",
    "X-HTTP-Method": "MERGE",
    "X-RequestDigest": $("#__REQUESTDIGEST").val(),
    "If-Match": ETag
  }

  var customerData = {
    __metadata: { "type": "SP.Data.CustomersListItem" },
    Title: LastName,
    FirstName: FirstName,
    Company: Company,
    WorkPhone: WorkPhone,
    HomePhone: HomePhone,
    Email: Email
  };

  var requestBody = JSON.stringify(customerData);

  return $.ajax({
    url: requestUri,
    type: "POST",
    contentType: "application/json;odata=verbose",
    headers: requestHeaders,
    data: requestBody,
  });

};
```

Deleting a SharePoint List Item

```
var deleteCustomer = function (Id) {

  var requestUri = "../_api/web/lists/getByTitle('Customers')/items(" + Id + ")";

  var requestHeaders = {
    "accept": "application/json;odata=verbose",
    "X-RequestDigest": $("#__REQUESTDIGEST").val(),
    "If-Match": "*"
  }

  return $.ajax({
    url: requestUri,
    type: "DELETE",
    headers: requestHeaders,
  });

};
```

DEMO

Exploring the SharePointCRM App

Summary

- ✓ The SharePoint REST API
- ✓ Creating REST URIs for SharePoint Objects
- ✓ Consuming OData Results from SharePoint
- ✓ Paging SharePoint List Items
- ✓ Adding and Updating Items