Versioning

Scott Seely

http://www.pluralsight.com/

Outline

- What is versioning?
- Versioning Data
- Versioning SOAP Endpoints
- Versioning REST Endpoints

Versioning: The thoughtful application of changes to a system that is already in production.

How do you “plan” versioning?

- **During v1 (*or ASAP if you shipped once already*)**
  - Pick a version-able identifier
  - Pick a versioning pattern
- **What are version-able identifiers?**
  - XML Data: XML Namespace
  - Things that listen at URLs: URL path
- **Numbering pattern:**
  - http://www.pluralsight.com/[service name]/[major].[minor].[build]
  - Each number increases monotonically: 1, 2, 3, 4, …, 9, 10, 11
- **Date Pattern**
  - http://www.pluralsight.com/[service name]/[yyyy]/[mm]/[dd]
  - http://www.pluralsight.com/[service name]/[yyyy].[mm].[dd]

What causes a need to version?

- **Data**
  - Add a new type
  - Change fields
  - Require fields
- **SOAP**
  - Add/remove methods
  - Change parameter names
  - In session: change the set of initiating/termination actions
- **REST**
  - Change URI structure
  - Add methods

Versioning Data

- **Reasons to version data**
  - Name chooser picked bad names (aka renames)
  - Adding/removing fields (maybe)
- **Things to handle upon versioning data**
  - Co-existing old/new clients
- **Reasons not to version data**
  - Re-ordering for esthetic reasons *(don't do this!)*

Q: Can/would you *ever* version data independent of services?

- **Yes. Agile development will do this regularly.**
- **Caveats:**
  - New fields must have sensible defaults
  - New fields must be null-able
  - New fields must be ordered at the ‘end’ of the object

Name Change

```
[DataContract(Namespace = Namespaces.V1)]
public class LineItem
{
  [DataMember]
  public int Line { get; set; }

  [DataMember]
  public int ItemId { get; set; }

  [DataMember]
  public double Price { get; set; }

  [DataMember]
  public int Qty { get; set; }

  [DataMember]
  public int PurchaseOrderId { get; set; }
}
```

```
[DataContract(Namespace = Namespaces.V2)]
public class LineItem
{
  [DataMember]
  public int Line { get; set; }

  [DataMember]
  public int ItemId { get; set; }

  [DataMember]
  public double Price { get; set; }

  [DataMember]
  public int Quantity { get; set; }

  [DataMember]
  public int PurchaseOrderId { get; set; }
}
```

New Members

- **If new member is required, define sensible default**
  - Caveat: if no sensible default exists, you have a serious bug in previous version (missed feature?)
  - If you think you have no sensible default and customers won’t change, you will suddenly find a sensible default
- **Place at end**
  - Some clients may be ordinal dependent, not name.
  - Use DataMember.Order property
- **Or inherit DTO if new item breaks old clients**

Adding New Data Member

```
[DataContract(Namespace = Namespaces.V2)]
public class PurchaseOrder
{
  [DataMember]
  public int CustomerId { get; set; }

  [DataMember]
  public List<DTO.v2.LineItem> LineItems { get; private set; }

  [DataMember]
  public int PurchaseOrderId { get; set; }

  [DataMember(Order = 100)]
  public DateTime OrderDate { get; set; }
}
```

Add New Member: Inheritance

```
[DataContract(Namespace = Namespaces.V3)]
public class PurchaseOrder : DTO.v2.PurchaseOrder
{
  [DataMember()]
  public string Comments { get; set; }
}
```

Versioning SOAP Endpoints

- **Reasons to version endpoints**
  - Name chooser picked bad names (aka renames)
  - Adding methods
  - Removing methods
  - Updating method parameters (new version of old type)
  - Adding terminating functions (might already exist…)
- **Things to handle upon versioning endpoints**
  - Update old to call out to controller properly
- **Reasons not to version endpoint**
  - Re-ordering for esthetic reasons *(don't do this!)*

Handle Versioning

- **Keep services light**
- **Keep details in separate class**
- **Use MVC type pattern**
  - DTO is *Model*
  - Service is *View*
  - “Separate class” is *Controller*
- **Benefits**
  - Minimize code churn in services
  - Keep logic mapping DTO to Business in one place

Example Implementation

```
V1:
public bool SubmitPO(PurchaseOrder purchaseOrder)
{
  return _controller.Submit(purchaseOrder);
}
```

```
V2:
public bool SubmitPO(DTO.v2.PurchaseOrder
   purchaseOrder)
{
  return _controller.Submit(purchaseOrder);
}
```

Controller knows the difference and handles any delta. SVC remains simple/closed.

Key takeaways

- Keep service code clean, simple.
- Think of service as a machine UI to the real object model.
- Let internal logic worry about mapping DTO to business object.
- Handle upgrade in one place, away from service.

Versioning REST Endpoints

- **Reasons to version REST endpoints**
  - Name chooser picked bad names (aka renames)
  - Paths don't make sense
  - Updating data types, parameters (new version of old type)
- **Things to handle upon versioning endpoints**
  - Update old to call out to controller properly
- **Reasons not to version endpoint**
  - Supporting more of Uniform Interface (GET | HEAD, PUT, POST, DELETE)

Where to version?

- **New set of services in new directory**
  - Pros: clean separation, small files, easy to read, allows for easier deletion later
  - Cons: not necessary, moves logic to new file
- **Keep everything in one file, add new UriTemplates**
  - Pros: All URLs in one place, easy to see what uses code, can see evolution in one place
  - Cons: Likely to clutter code, increase maintenance burden over time

v1 to v2 Migration

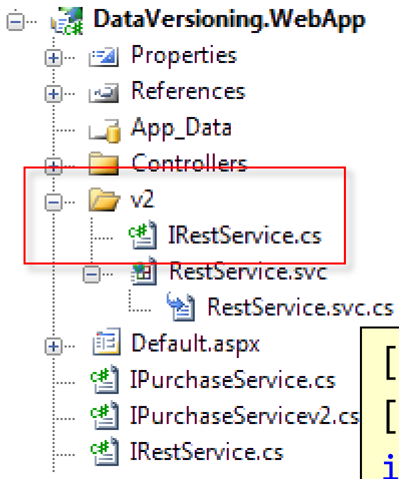
```
[OperationContract]
[WebInvoke(Method = "POST", UriTemplate="v1/PurchaseOrder")]
int CreatePurchaseOrder(DTO.PurchaseOrder purchaseOrder);

[OperationContract]
[WebInvoke(Method = "PUT", UriTemplate = "v1/PurchaseOrder/{id}")]
bool Update(string id, DTO.PurchaseOrder purchaseOrder);

[OperationContract]
[WebInvoke(Method = "DELETE", UriTemplate = "v1/PurchaseOrder/{id}")]
bool Delete(string id);

[OperationContract]
[WebGet(UriTemplate = "v1/PurchaseOrder/{id}")]
DTO.PurchaseOrder Get(string id);
```

v1 to v2 Migration


DataVersioning.WebApp
Properties
References
App_Data
Controllers
v2
IRestService.cs
RestService.svc
RestService.svc.cs
Default.aspx
IPurchaseService.cs
IPurchaseServicev2.cs
IRestService.cs

```
[OperationContract]
[WebInvoke(Method = "POST", UriTemplate = "PurchaseOrder")]
int CreatePurchaseOrderv2(DTO.v2.PurchaseOrder purchaseOrder);

[OperationContract]
[WebInvoke(Method = "PUT", UriTemplate = "PurchaseOrder/{id}")]
bool Updatev2(string id, DTO.v2.PurchaseOrder purchaseOrder);

[OperationContract]
[WebInvoke(Method = "DELETE", UriTemplate = "PurchaseOrder/{id}")]
bool Deletev2(string id);

[OperationContract]
[WebGet(UriTemplate = "PurchaseOrder/{id}")]
DTO.v2.PurchaseOrder Getv2(string id);
```

Support Plans for Versioning

- **Need an SLA between service and consumers**
- **Need to define**
  - How long a consumer can depend on a version
  - What constitutes end of life for a version
- **Need to prepare for**
  - Extensions to end of life
  - Presence of older clients
  - Work with client to update to latest

Assets to Create to Assist/Reduce Burden

- **Documentation on wire-level formats**
  - XSD, HTML documents, sample code
- **SDKs in client languages**
  - Java, Ruby, PHP, Python, .NET
  - Can actually ease migration.
    - Client uses new library, works till clean compile
    - Versioning issue becomes deployment issue
- **Support staff**
  - Dedicate staff to help support clients who are migrating code.

Summary

- Versioning: The thoughtful application of changes to a system.
- Plan ahead
  - URL structure, version names, use MVC pattern
- Changes hit in three, related areas: Data, SOAP Methods, REST
- Manage changes
- Create new endpoints for new data types
- Use DTOs
- Keep logic to translate between DTO and business in one place

For more in-depth online developer training visit

pluralsight
see what you can learn

on-demand content from authors you trust