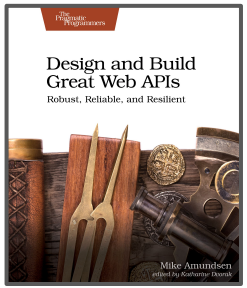
# Learning the API Design Method

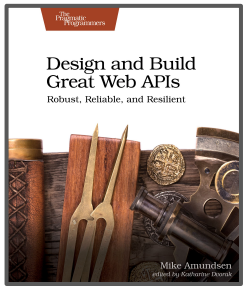
Mike Amundsen
@mamund



# Learning the API Design Method

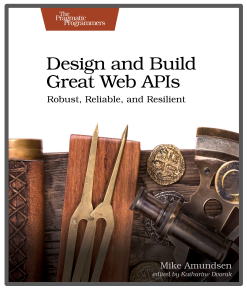
- Design the Interface
- Not the Implementation
- Five Step API Design Method





# Design the Interface





“Few people think about it or are aware of it. But there is nothing made by human beings that does not involve a design decision somewhere.”

Bill Moggridge
Interaction Design Pioneer

# Functionality, Usability, and User Experience: Three Areas of Concern

Niamh McNamara | University College Cork, Ireland | n.mcnamara@ucc.ie
Jurek Kirakowski | University College Cork, Ireland | jzk@ucc.ie

**design**

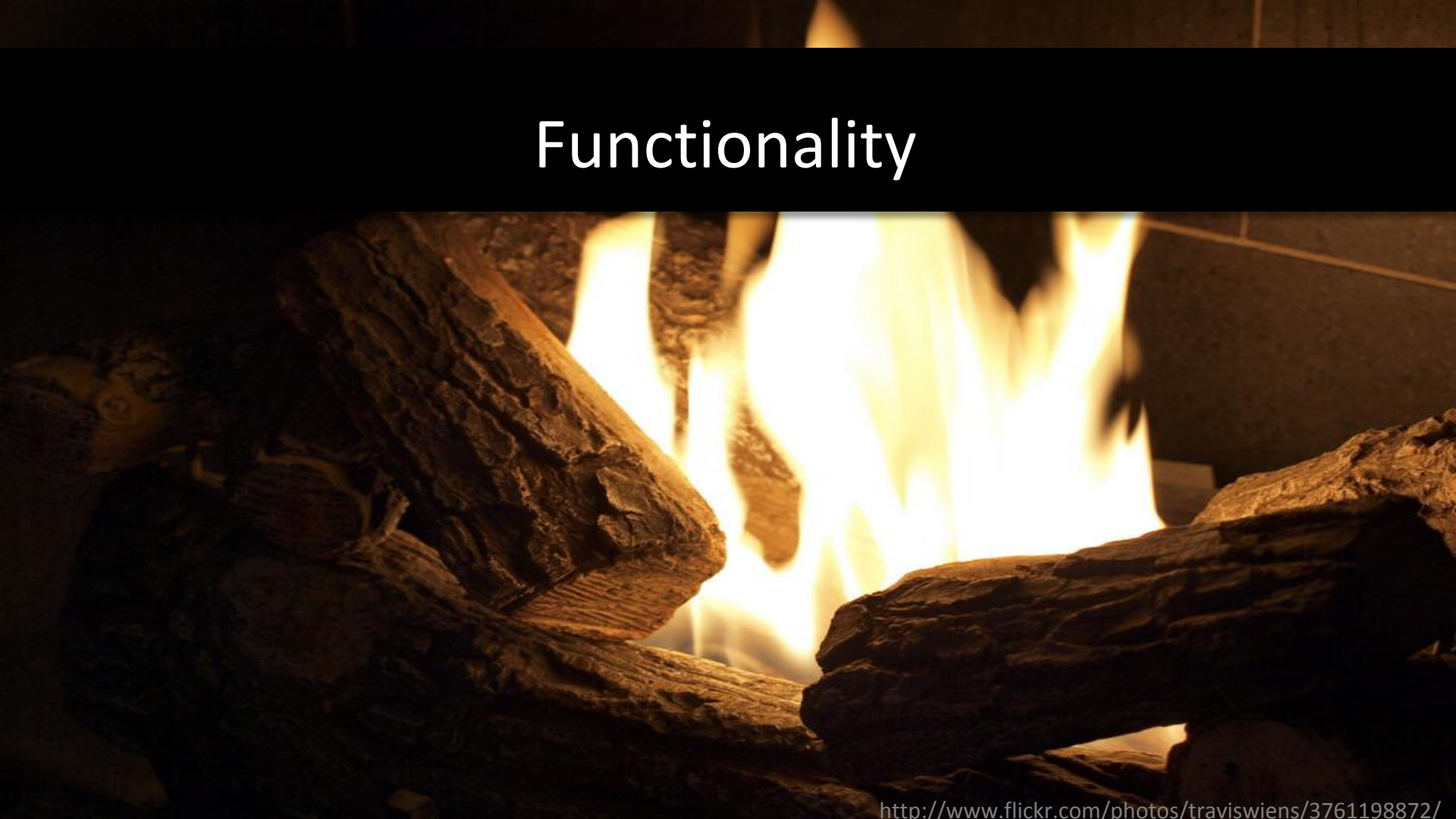
Functionality

Usability

Experience


http://www.flickr.com/photos/monicamuller/3171329060

# Functionality



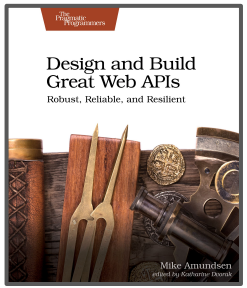
http://www.flickr.com/photos/traviswiens/3761198872/

# Usability

# Experience

# Not the Implementation

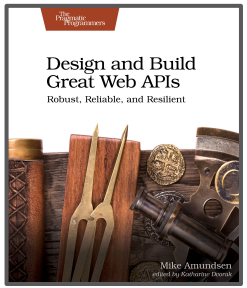




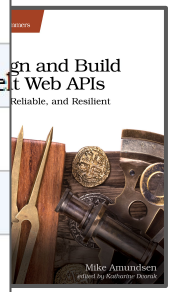
# Design Guidelines

- Craft [good/pretty/usable/stable] URIs





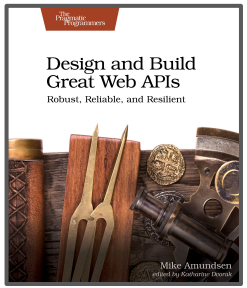
| Term | Description |
|---|---|
| Authority | A URI component that identifies the party with jurisdiction over the namespace defined by the remainder of the URI. |
| Collection | A resource archetype used to model a server-managed *directory* of resources. |
| Controller | A resource archetype used to model a procedural concept. |
| CRUD | An acronym that stands for the four classic storage-oriented functions: create, retrieve, update, and delete. |
| Developer portal | A Web-based graphical user interface that helps a REST API acquire new clients. |
| Docroot | A resource that is the hierarchical ancestor of all other resources within a REST API's model. This resource's URI should be the advertised entry point. |
| Document | A resource archetype used to model a singular concept. |
| Forward slash separator (/) | Used within the URI path component to separate hierarchically related resources. |
| Opacity of URIs | An axiom, originally described by Tim Berners-Lee, that governs the visibility of a resource identifier's composition. |
| Parent resource | The document, collection, or store that governs a given subordinate concept by preceding it within a URI's hierarchical path. |
| Query | A URI component that comes after the path and before the optional fragment. |
| Resource archetypes | A set of four intrinsic concepts (document, collection, store, and controller) that may be used to help describe a REST API's model |
| Store | A resource archetype used to model a client-managed resource repository. |
| URI path segment | Part of a resource identifier that represents a single node within a larger, hierarchical resource model. |
| URI template | A resource identifier syntax that includes variables that must be substituted before resolution. |



# Design Guidelines

- Craft [good/pretty/usable/stable] URIs
- Map domain actions to HTTP methods (CRUD)



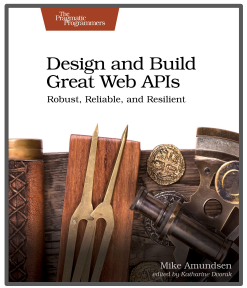


| Term | Description |
| --- | --- |
| `DELETE` | HTTP request method used to remove its parent. |
| `GET` | HTTP request method used to retrieve a representation of a resource's state. |
| `HEAD` | HTTP request method used to retrieve the metadata associated with the resource's state. |
| `OPTIONS` | HTTP request method used to retrieve metadata that describes a resource's available interactions. |
| `POST` | HTTP request method used to create a new resource within a collection or execute a controller. |
| `PUT` | HTTP request method used to insert a new resource into a store or update a mutable resource. |
| `Request-Line` | RFC 2616 defines its syntax as `Method SP Request-URI SP HTTP-Version CRLF` |
| Request method | Indicates the desired action to be performed on the request message's identified resource. |
| Response status code | A three-digit numeric value that is communicated by a server to indicate the result of a client's request. |
| `Status-Line` | RFC 2616 defines its syntax as: `HTTP-Version SP Status-Code SP Reason-Phrase CRLF` |
| Tunneling | An abuse of HTTP that masks or misrepresents a message's intent and undermines the protocol's transparency. |

# Design Guidelines

- Craft [good/pretty/usable/stable] URIs
- Map domain actions to HTTP methods (CRUD)
- Use the proper HTTP Status Codes



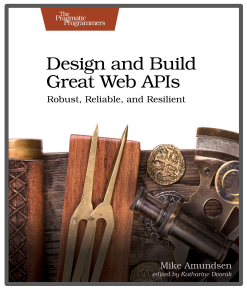


| Code | Name | Meaning |
| --- | --- | --- |
| 400 | Bad Request | Indicates a nonspecific client error |
| 401 | Unauthorized | Sent when the client either provided invalid credentials or forgot to send them |
| 402 | Forbidden | Sent to deny access to a protected resource |
| 404 | Not Found | Sent when the client tried to interact with a URI that the REST API could not map to a resource |
| 405 | Method Not Allowed | Sent when the client tried to interact using an unsupported HTTP method |
| 406 | Not Acceptable | Sent when the client tried to request data in an unsupported media type format |
| 409 | Conflict | Indicates that the client attempted to violate resource state |
| 412 | Precondition Failed | Tells the client that one of its preconditions was not met |
| 415 | Unsupported Media Type | Sent when the client submitted data in an unsupported media type format |
| 500 | Internal Server Error | Tells the client that the API is having problems of its own |

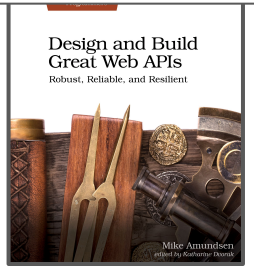
# Design Guidelines

- Craft [good/pretty/usable/stable] URIs
- Map domain actions to HTTP methods (CRUD)
- Use the proper HTTP Status Codes
- Document serialized objects as HTTP bodies





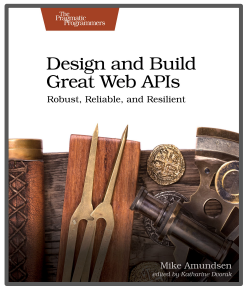
| Term | Description |
|---|---|
| Field | A named slot with some associated information that is stored in its value. |
| Form | A structured representation that consists of the fields and links, which are defined by an associated schema. |
| Format | Describes a form's presentation apart from its schematic. |
| Link | An actionable reference to a resource. |
| Link formula | A boolean expression that may serve as HATEOAS calculator's input in order to determine the availability of state-sensitive hypermedia within a form. |
| Link relation | Describes a connection between two resources. |
| Schema | Describes a representational form's structure independent of its format. |
| State fact | A Boolean variable that communicates a condition that is relevant to some state-sensitive hypermedia. |



# Design Guidelines

- Craft [good/pretty/usable/stable] URIs
- Map domain actions to HTTP methods (CRUD)
- Use the proper HTTP Status Codes
- Document serialized objects as HTTP bodies
- Use HTTP headers responsibly



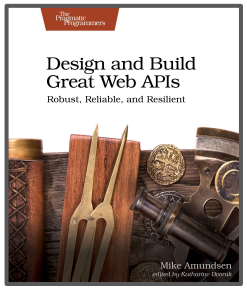


| Code | Purpose |
| --- | --- |
| `Content-Type` | Identifies the entity body's media type |
| `Content-Length` | The size (in bytes) of the entity body |
| `Last-Modified` | The date-time of last resource representation's change |
| `ETag` | Indicates the version of the response message's entity |
| `Cache-Control` | A TTL-based caching value (in seconds) |
| `Location` | Provides the URI of a resource |

# Design Guidelines

- Craft [good/pretty/usable/stable] URIs
- Map domain actions to HTTP methods (CRUD)
- Use the proper HTTP Status Codes
- Document serialized objects as HTTP bodies
- Use HTTP headers responsibly
- Describe edge cases (async, errors, authN/Z)





```
# Response
HTTP/1.1 202 Accepted ❶
Content-Type: application/xml;charset=UTF-8
Content-Location: http://www.example.org/images/task/1
Date: Sun, 13 Sep 2009 01:49:27 GMT

<status xmlns:atom="http://www.w3.org/2005/Atom">
  <state>pending</state>
  <atom:link href="http://www.example.org/images/task/1" rel="self"/>
  <message >Your request has been accepted for processing.</message>
  <ping-after>2009-09-13T01:59:27Z</ping-after> ❷
</status>
```

```
# Response
HTTP/1.1 409 Conflict
Content-Type: application/xml;charset=UTF-8
Content-Language: en
Date: Wed, 14 Oct 2009 10:16:54 GMT
Link: <http://www.example.org/errors/limits.html>;rel="help"

<error xmlns:atom="http://www.w3.org/2005/Atom">
  <message>Account limit exceeded. We cannot complete the transfer due to
  insufficient funds in your accounts</message>
  <error-id>321-553-495</error-id>
  <account-from>urn:example:account:1234</account-from>
  <account-to>urn:example:account:5678</account-to>
  <atom:link href="http://example.org/account/1234"
             rel="http://example.org/rels/transfer/from"/>
  <atom:link href="http://example.org/account/5678"
             rel="http://example.org/rels/transfer/to"/>
</error>
```

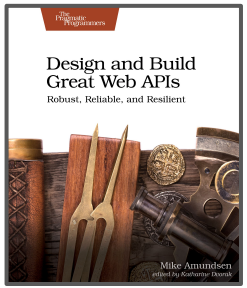
```
# Request to obtain a request token
POST /request_token HTTP/1.1 ❶
Host: www.example.org
Authorization: OAuth realm="http://www.example.com/photos",
                     oauth_consumer_key=a1191fd420e0164c2f9aeac32ed35d23,
                     oauth_nonce=109843dea839120a,
                     oauth_signature=d8e19bb988110380a72f6ca33b2ba5903272fe1,
                     oauth_signature_method=HMAC-SHA1,
                     oauth_timestamp=1258308730,
                     oauth_version=1.0 ❷
Content-Length: 0

# Response containing a request token and a secret
HTTP/1.1 200 OK
Content-Type: application/x-www-form-urlencoded

oauth_token=0e713d524f290676de8aff4073b1bb52e37f065c
  &oauth_token_secret=394bc633d4c93f79aa0539fd554937760f05987c ❸
```

# API Design Method

#mcaTravels

**@mamund**

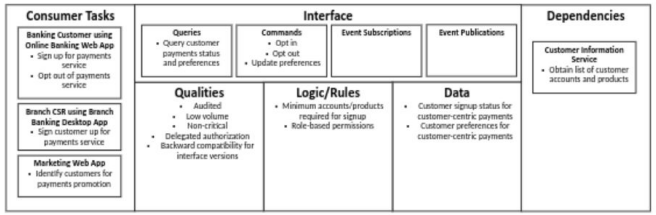
#api360

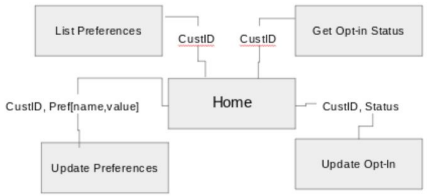


# An API Design Methodology

***A repeatable process to govern the creation of interfaces***

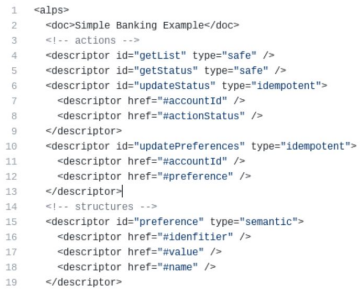
- Produce a Service Canvas
- Draw a Diagram
- Apply Vocabularies
- Create Description Document




```
<alps>
  <doc>Simple Banking Example</doc>
  <!-- actions -->
  <descriptor id="getList" type="safe" />
  <descriptor id="getStatus" type="safe" />
  <descriptor id="updateStatus" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#actionStatus" />
  </descriptor>
  <descriptor id="updatePreferences" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#preference" />
  </descriptor>
  <!-- structures -->
  <descriptor id="preference" type="semantic">
    <descriptor href="#idenfitier" />
    <descriptor href="#value" />
    <descriptor href="#name" />
  </descriptor>
```

# Produce a Service Canvas



```
<alps>
  <doc>Simple Banking Example</doc>
  <!-- actions -->
  <descriptor id="getList" type="safe" />
  <descriptor id="getStatus" type="safe" />
  <descriptor id="updateStatus" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#actionStatus" />
  </descriptor>
  <descriptor id="updatePreferences" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#preference" />
  </descriptor>
  <!-- structures -->
  <descriptor id="preference" type="semantic">
    <descriptor href="#idenfitier" />
    <descriptor href="#value" />
    <descriptor href="#name" />
  </descriptor>
```

# Design Canvas: Customer-centric Payments Management Service


## Consumer Tasks

**Banking Customer using Online Banking Web App**
- Sign up for payments service
- Opt out of payments service

**Branch CSR using Branch Banking Desktop App**
- Sign customer up for payments service

**Marketing Web App**
- Identify customers for payments promotion

## Interface

**Queries**
- Query customer payments status and preferences

**Commands**
- Opt in
- Opt out
- Update preferences

**Event Subscriptions**

**Event Publications**

## Qualities

- Audited
- Low volume
- Non-critical
- Delegated authorization
- Backward compatibility for interface versions

## Logic/Rules

- Minimum accounts/products required for signup
- Role-based permissions

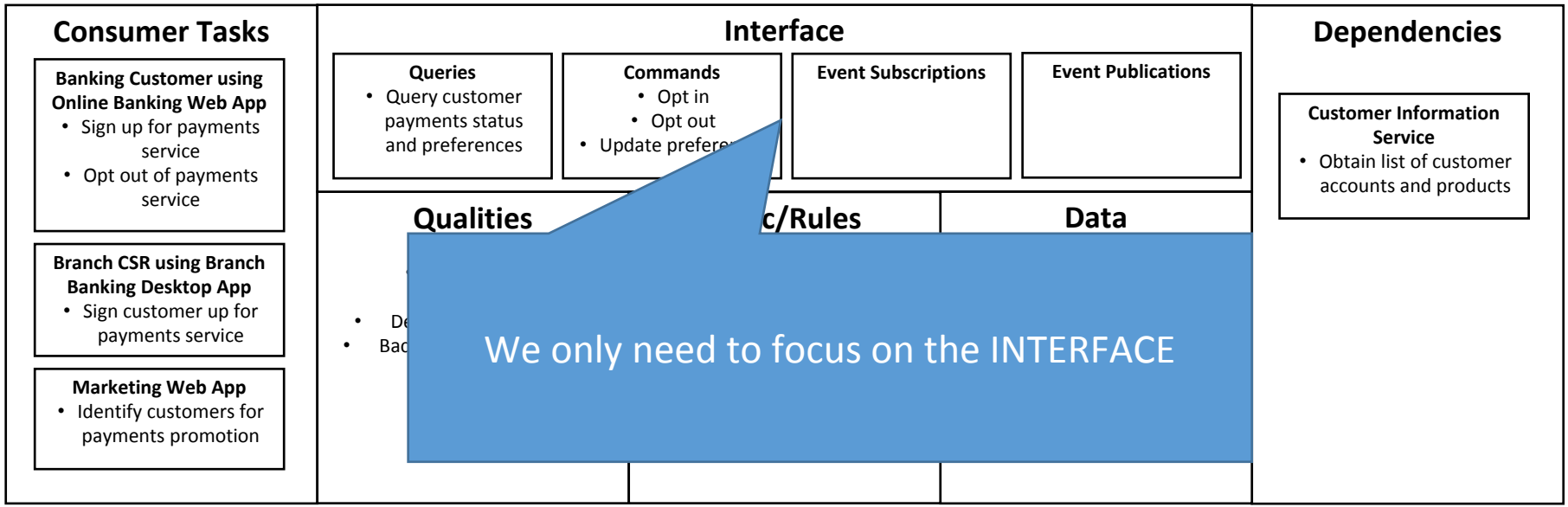
## Data

- Customer signup status for customer-centric payments
- Customer preferences for customer-centric payments

## Dependencies

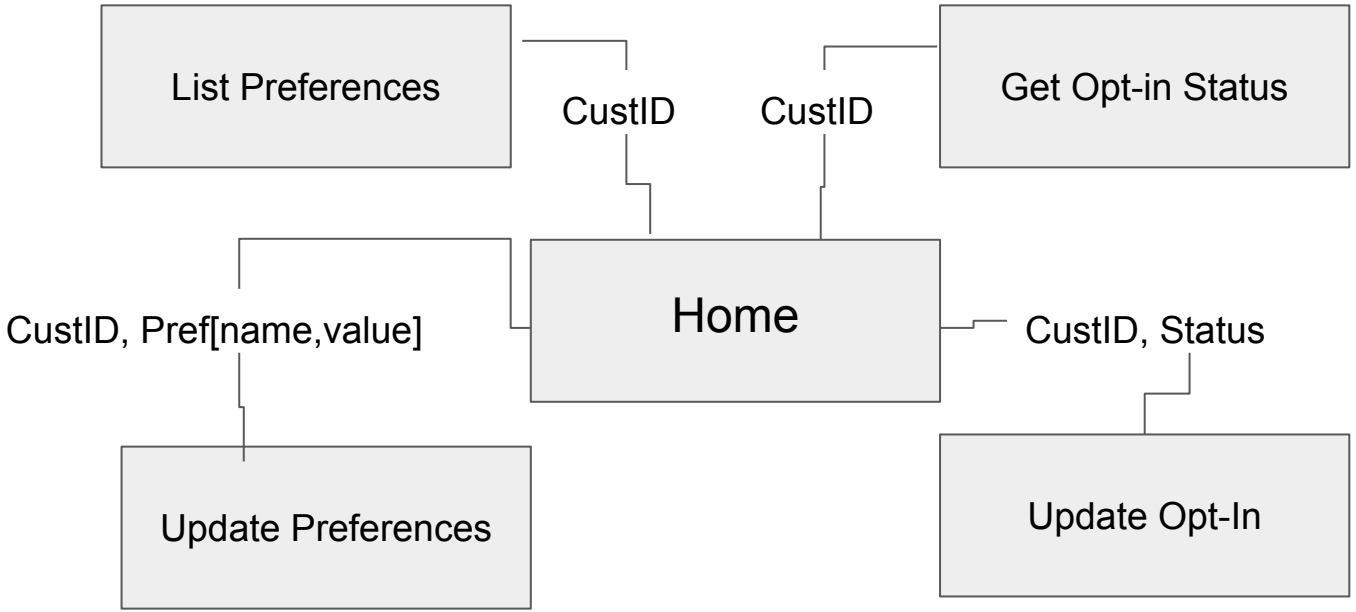
**Customer Information Service**
- Obtain list of customer accounts and products

# List all the Actions

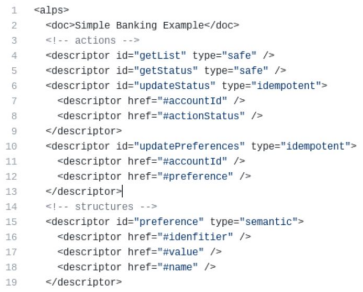


# Draw a Diagram

```
<alps>
  <doc>Simple Banking Example</doc>
  <!-- actions -->
  <descriptor id="getList" type="safe" />
  <descriptor id="getStatus" type="safe" />
  <descriptor id="updateStatus" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#actionStatus" />
  </descriptor>
  <descriptor id="updatePreferences" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#preference" />
  </descriptor>
  <!-- structures -->
  <descriptor id="preference" type="semantic">
    <descriptor href="#idenfitier" />
    <descriptor href="#value" />
    <descriptor href="#name" />
  </descriptor>
```


List Preferences
CustID
CustID
Get Opt-in Status
CustID, Pref[name,value]
Home
CustID, Status
Update Preferences
Update Opt-In

# Apply Vocabularies



```
<alps>
  <doc>Simple Banking Example</doc>
  <!-- actions -->
  <descriptor id="getList" type="safe" />
  <descriptor id="getStatus" type="safe" />
  <descriptor id="updateStatus" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#actionStatus" />
  </descriptor>
  <descriptor id="updatePreferences" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#preference" />
  </descriptor>
  <!-- structures -->
  <descriptor id="preference" type="semantic">
    <descriptor href="#idenfitier" />
    <descriptor href="#value" />
    <descriptor href="#name" />
  </descriptor>
```

# Sources for Vocabularies

- IANA Link Relation Values
- schema.org
- microformats
- Dublin Core
- Activity Streams
- Industry Vocabularies (BIAN, etc.)
- Your Enterprise Vocabularies

```
<alps>
  <doc>Simple Banking Example</doc>
  <!-- actions -->
  <descriptor id="getList" type="safe" />
  <descriptor id="getStatus" type="safe" />
  <descriptor id="updateStatus" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#actionStatus" />
  </descriptor>
  <descriptor id="updatePreferences" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#preference" />
  </descriptor>
  <!-- structures -->
  <descriptor id="preference" type="semantic">
    <descriptor href="#idenfitier" />
    <descriptor href="#value" />
    <descriptor href="#name" />
  </descriptor>
```

# Before Applying Vocabularies

- CustID,
- CustomerName,
- AccountName,
- AccountType
- Optin-Status(in, out)
- Preference(Name, Value, Prompt)
- GetStatus
- GetPreferences
- UpdateStatus
- UpdatePreferences

```
<alps>
  <doc>Simple Banking Example</doc>
  <!-- actions -->
  <descriptor id="getList" type="safe" />
  <descriptor id="getStatus" type="safe" />
  <descriptor id="updateStatus" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#actionStatus" />
  </descriptor>
  <descriptor id="updatePreferences" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#preference" />
  </descriptor>
  <!-- structures -->
  <descriptor id="preference" type="semantic">
    <descriptor href="#idenfitier" />
    <descriptor href="#value" />
    <descriptor href="#name" />
  </descriptor>
```

# After Applying Vocabularies

- BankAccount.identifier,
- Customer.familyName,
- Customer.givenName,
- BankAccount.name,
- BankAccount.category
- ActionStatus("in", "out")
- ItemList(identifier, value, name)
- **GetStatus**
- **GetPreferences**
- **UpdateStatus**
- **UpdatePreferences**

```
<alps>
  <doc>Simple Banking Example</doc>
  <!-- actions -->
  <descriptor id="getList" type="safe" />
  <descriptor id="getStatus" type="safe" />
  <descriptor id="updateStatus" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#actionStatus" />
  </descriptor>
  <descriptor id="updatePreferences" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#preference" />
  </descriptor>
  <!-- structures -->
  <descriptor id="preference" type="semantic">
    <descriptor href="#idenfitier" />
    <descriptor href="#value" />
    <descriptor href="#name" />
  </descriptor>
```

# Create a Description Document

```
<alps>
  <doc>Simple Banking Example</doc>
  <!-- actions -->
  <descriptor id="getList" type="safe" />
  <descriptor id="getStatus" type="safe" />
  <descriptor id="updateStatus" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#actionStatus" />
  </descriptor>
  <descriptor id="updatePreferences" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#preference" />
  </descriptor>
  <!-- structures -->
  <descriptor id="preference" type="semantic">
    <descriptor href="#idenfitier" />
    <descriptor href="#value" />
    <descriptor href="#name" />
  </descriptor>
```

# Description vs. Definitions

- Describing the interface doesn't define it.
- Description languages
  - ALPS
  - DCAP
  - JSON Home

```
<alps>
  <doc>Simple Banking Example</doc>
  <!-- actions -->
  <descriptor id="getList" type="safe" />
  <descriptor id="getStatus" type="safe" />
  <descriptor id="updateStatus" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#actionStatus" />
  </descriptor>
  <descriptor id="updatePreferences" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#preference" />
  </descriptor>
  <!-- structures -->
  <descriptor id="preference" type="semantic">
    <descriptor href="#idenfitier" />
    <descriptor href="#value" />
    <descriptor href="#name" />
  </descriptor>
```

# Description vs. Definitions

- Describing the interface doesn't define it.
- Description languages
  - ALPS
  - DCAP
  - JSON Home
- Definition languages
  - WSDL
  - Swagger
  - RAML
  - Blueprint

```
<alps>
  <doc>Simple Banking Example</doc>
  <!-- actions -->
  <descriptor id="getList" type="safe" />
  <descriptor id="getStatus" type="safe" />
  <descriptor id="updateStatus" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#actionStatus" />
  </descriptor>
  <descriptor id="updatePreferences" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#preference" />
  </descriptor>
  <!-- structures -->
  <descriptor id="preference" type="semantic">
    <descriptor href="#idenfitier" />
    <descriptor href="#value" />
    <descriptor href="#name" />
  </descriptor>
```

# Description vs. Definitions

- Describing the interface doesn't define it.
- Description languages
  - **ALPS**
  - DCAP
  - JSON Home
- Definition languages
  - WSDL
  - Swagger
  - RAML
  - Blueprint

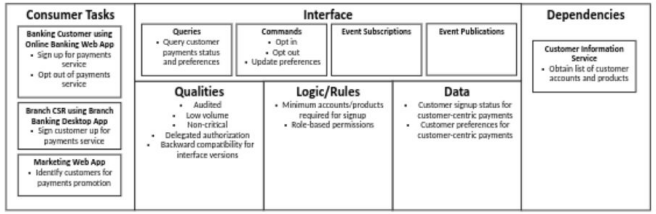
```
<alps>
  <doc>Simple Banking Example</doc>
  <!-- actions -->
  <descriptor id="getList" type="safe" />
  <descriptor id="getStatus" type="safe" />
  <descriptor id="updateStatus" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#actionStatus" />
  </descriptor>
  <descriptor id="updatePreferences" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#preference" />
  </descriptor>
  <!-- structures -->
  <descriptor id="preference" type="semantic">
    <descriptor href="#idenfitier" />
    <descriptor href="#value" />
    <descriptor href="#name" />
  </descriptor>
```

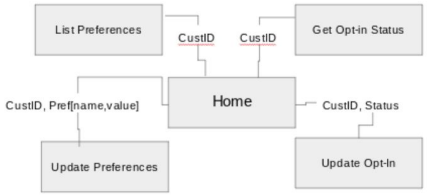
```
<alps>
  <doc>Simple Banking Example</doc>
  <!-- actions -->
  <descriptor id="getList" type="safe" />
  <descriptor id="getStatus" type="safe" />
  <descriptor id="updateStatus" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#actionStatus" />
  </descriptor>
  <descriptor id="updatePreferences" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#preference" />
  </descriptor>
  <!-- structures -->
  <descriptor id="preference" type="semantic">
    <descriptor href="#idenfitier" />
    <descriptor href="#value" />
    <descriptor href="#name" />
  </descriptor>
```

# Design Artifacts

- Service Canvas
- Diagram
- Description Document

***Check these into source control***



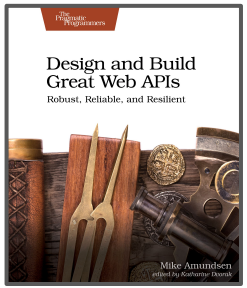




```
<alps>
  <doc>Simple Banking Example</doc>
  <!-- actions -->
  <descriptor id="getList" type="safe" />
  <descriptor id="getStatus" type="safe" />
  <descriptor id="updateStatus" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#actionStatus" />
  </descriptor>
  <descriptor id="updatePreferences" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#preference" />
  </descriptor>
  <!-- structures -->
  <descriptor id="preference" type="semantic">
    <descriptor href="#idenfitier" />
    <descriptor href="#value" />
    <descriptor href="#name" />
  </descriptor>
```

# Learning the API Design Method

- Design the Interface
- Not the Implementation
- Five Step API Design Method





# Learning the API Design Method

Mike Amundsen
@mamund