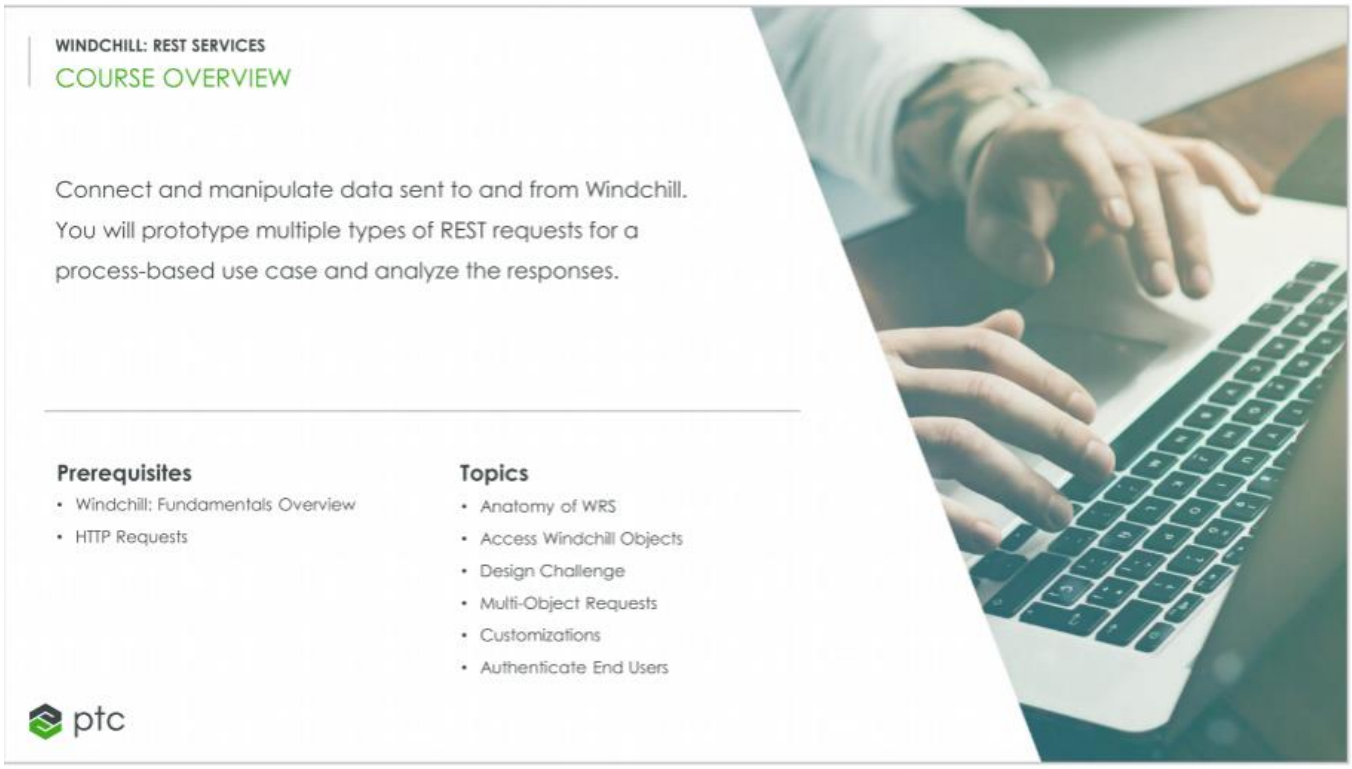
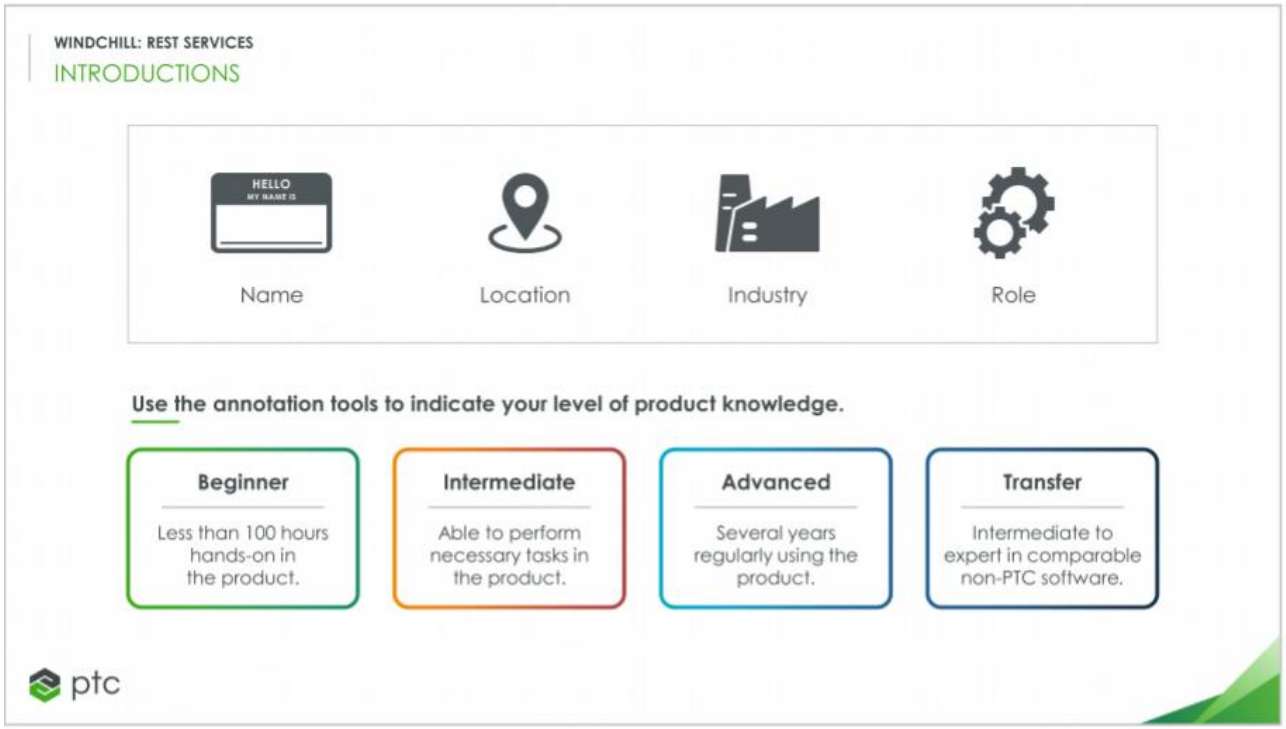
Windchill REST Services

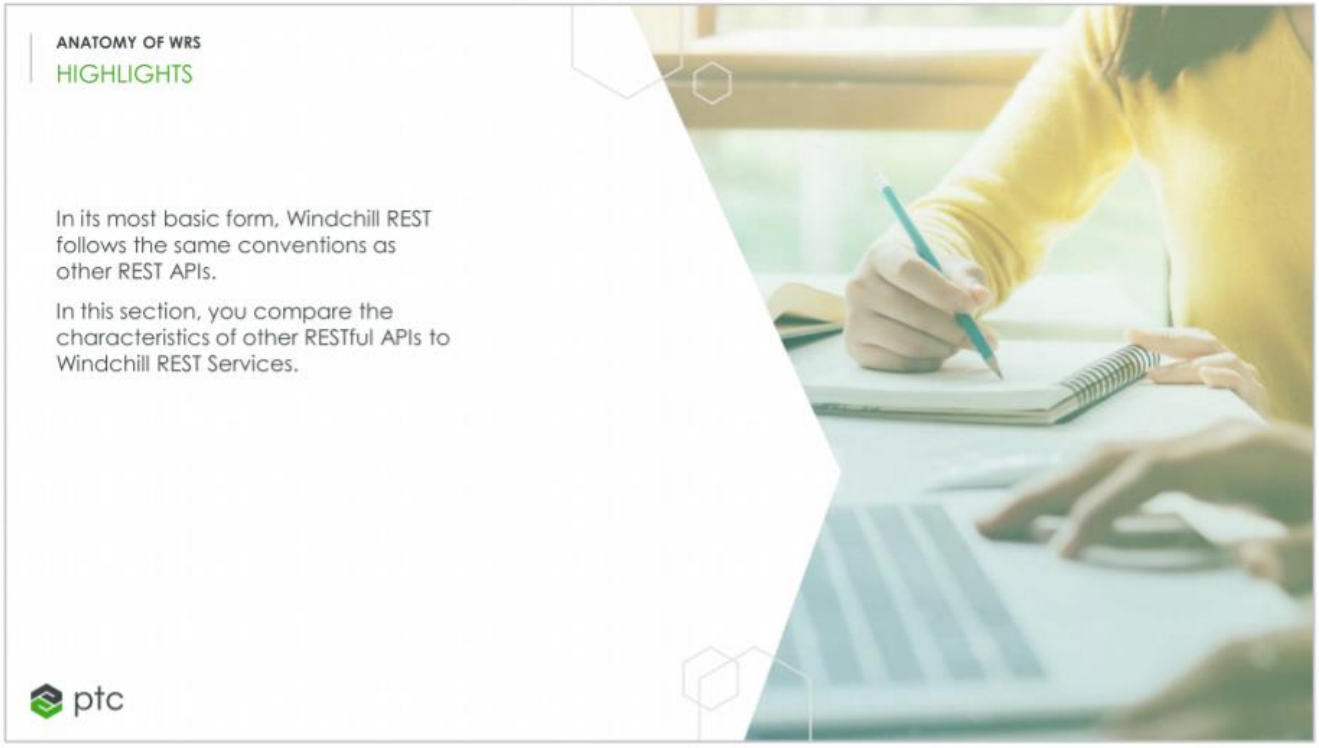
Overview



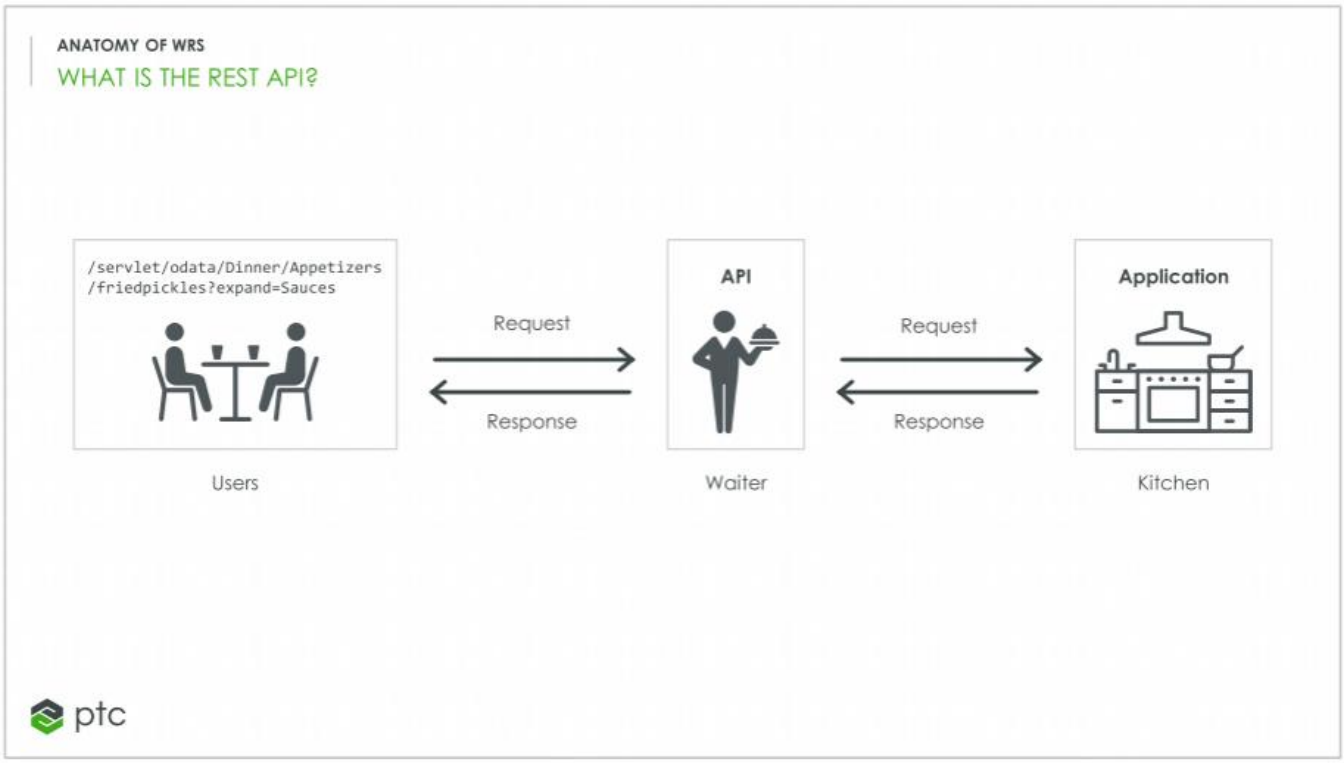


Anatomy of Windchill REST Services

Highlights



What is the REST API?



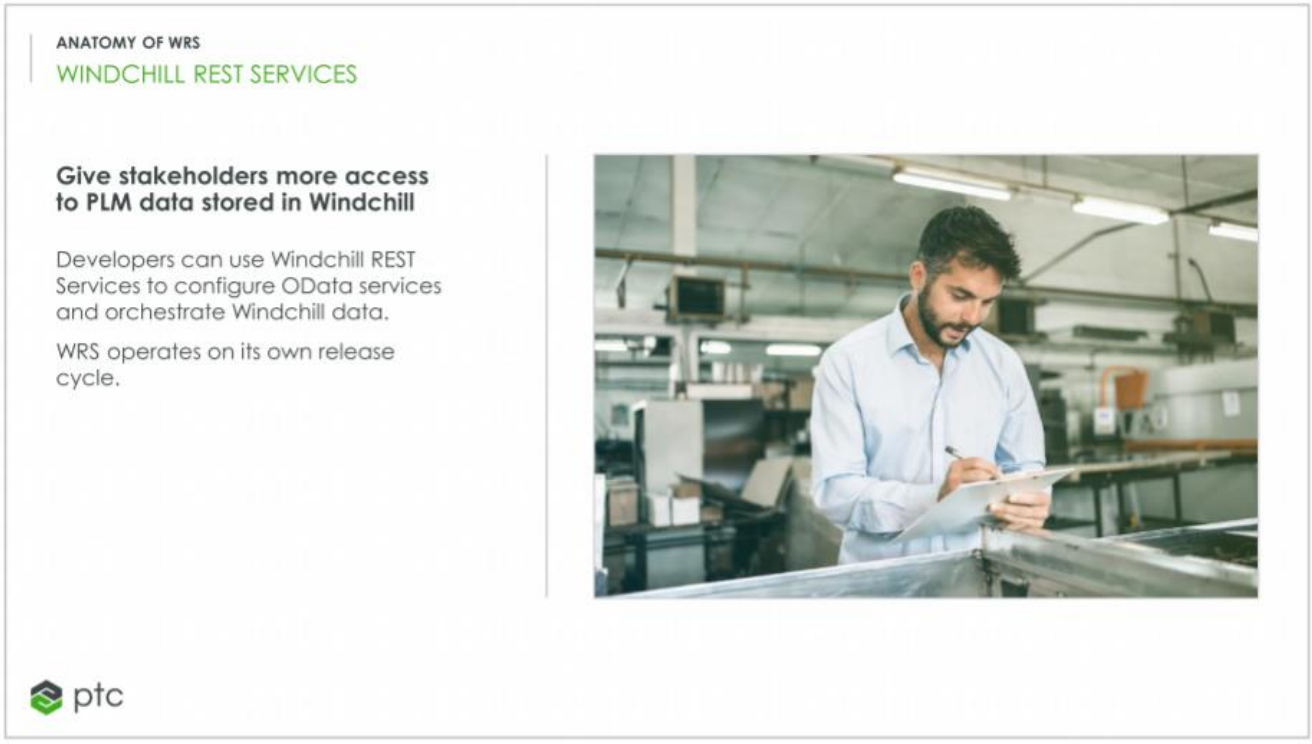
Representational State Transfer (REST) refers to a web service which transfers "representations" over the internet. You use a "representation" of a resource to transfer the resource state, which lives on the server, into an application state on the client.

In computer programming, an API (application programming interface) is a set of subroutine definitions, protocols, and tools for building application software. A client makes a request and the server responds, similar to ordering in a restaurant. You first place your order for an appetizer. Your order is your request to the waiter who similarly acts as an API. The waiter then takes your request to the kitchen, which acts as the application where your request will be processed. In the kitchen, your prepared food is the response to your request. Finally, the waiter (API) will bring your food (response) to your table.

REST resources are data on which you want to perform actions or simply read. A REST resource gives you access to that data but does not in itself do anything with that data. REST implies that you are exchanging data. Therefore, this data can be present in the database as records of tables or in any other form. This record has a unique identifier, like an identification number for an employee. A resource is accessed via a common interface based on HTTP standard methods.

With a REST API like Windchill REST Services, you define resources (the nouns) and use a uniform interface to operate them (the verbs). Open Data (OData), which Windchill REST Services is based on, is a REST-based protocol developed for querying and updating data; it's effectively a set of rules. OData is built on standardized technologies, such as HTTP, Atom/XML, and JSON.

Windchill REST Services

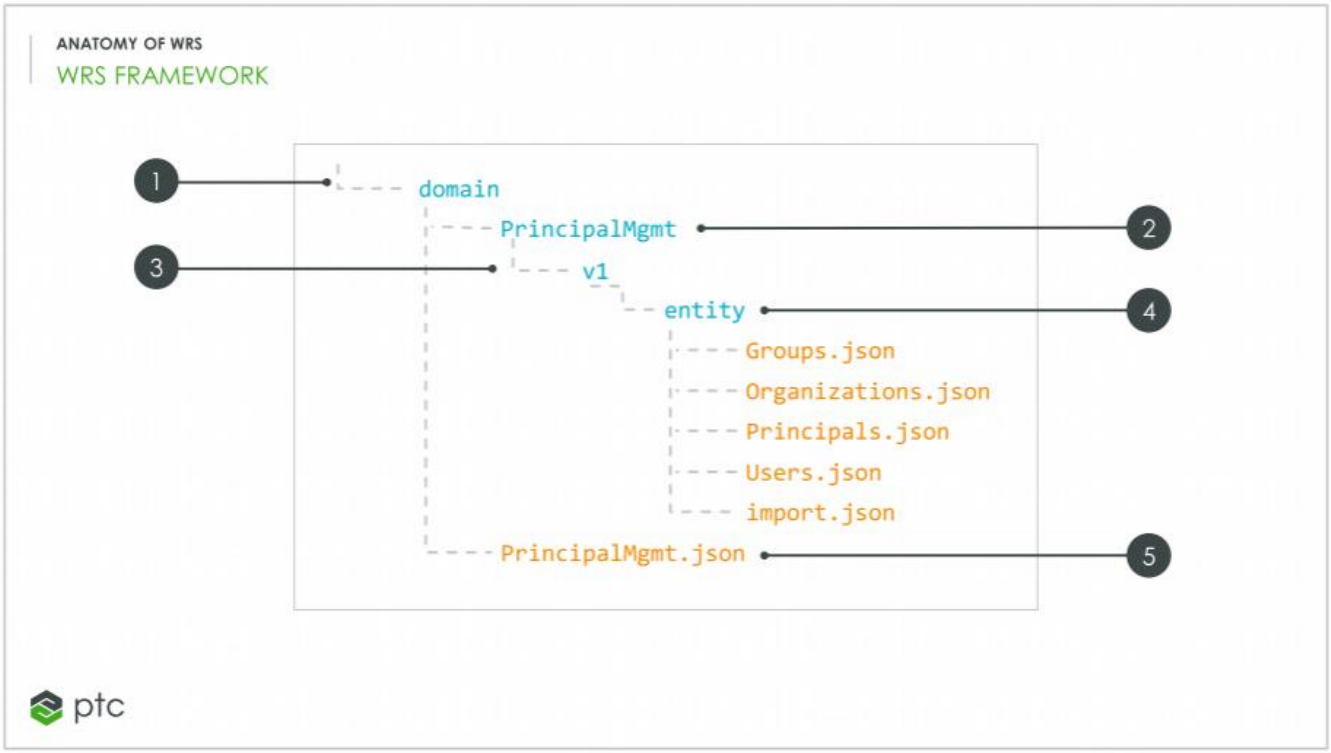


Windchill REST Services (from here referred to as its acronym, WRS) is how you communicate with Windchill and other third-party enterprise systems to read and modify Windchill data. As there is a growing need to collaborate with remote data across the Digital Thread, exposing this data using WRS can help your enterprise to develop infrastructure to analyze, manage, and communicate Windchill data across departments and with external collaborators.

WRS follows the OData protocol, so it helps to understand OData components to understand WRS. OData enables the creation of REST-based services that enable resource identification using Uniform Resource Locators (URLs). OData then uses a data model to be published and edited by web clients which use simple HTTP messages.

WRS is a Windchill module that operates on its own release cycle and is bundled with a supported version of Windchill. Install WRS through the PTC Solution Installer. Refer to the software matrices for Windchill REST Services at https://www.ptc.com/en/support/refdoc for compatibility with your Windchill solution.

WRS FRAMEWORK



An OData service in Windchill REST Services (WRS) is called a domain, but you will commonly hear them referred to as services or service endpoints. Domains expose Windchill object types and object collections as OData entity types and entity sets.

On top of that, the framework (wcOdata) also provides basic and common Windchill capabilities, such as CRUD operations on Windchill persistables, support for soft types and soft attributes, and so on.

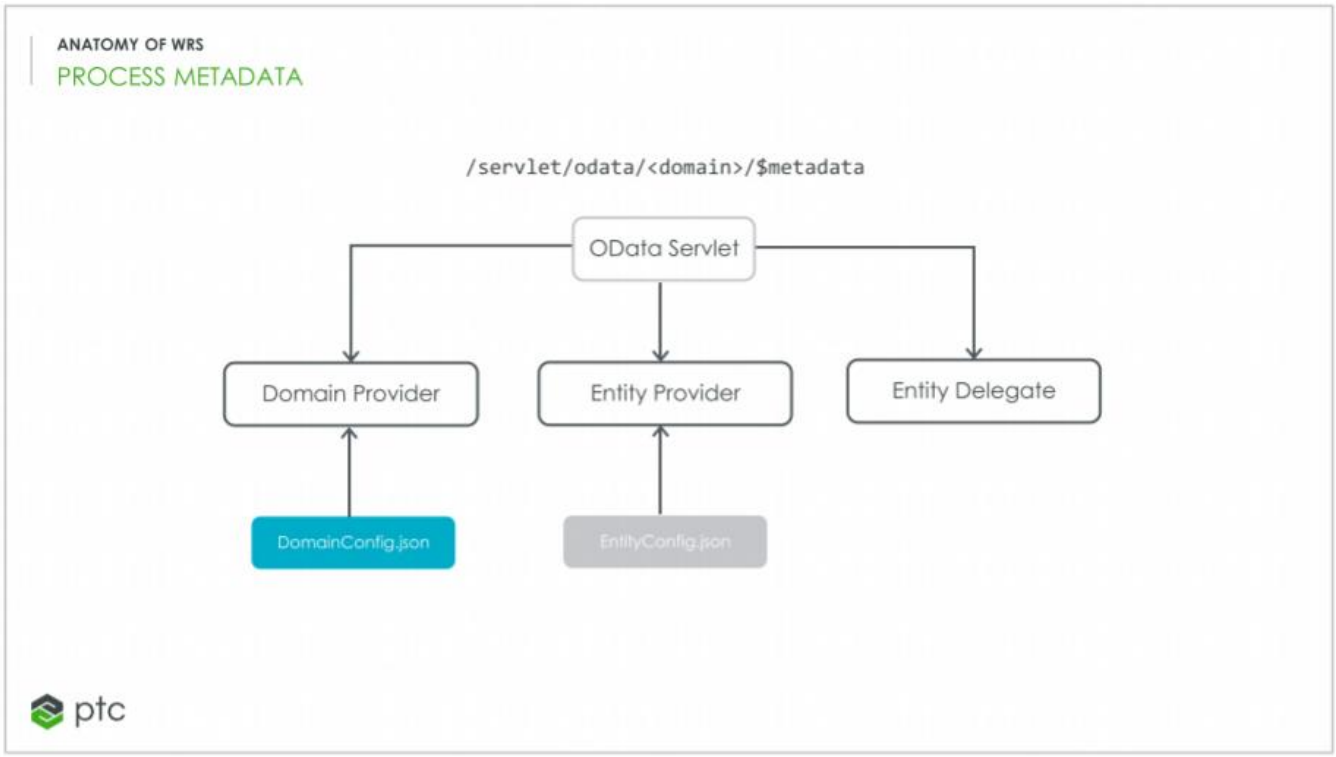
The framework reads a set of configuration files that live in the Windchill codebase. The configuration files have the info required to set up and use OData services on Windchill. Domains expose Windchill object types (including parts, documents, and change objects) and object collections as OData entity types and entity sets.

The domains are designed to enable access to smaller and functionally independent areas of this schema to RESTful clients. The servlet understands the domains. It reads domains and resource configurations and provides a view of the domains to the client.

OData standard specifies the URLs to access entities and entity sets in a domain. The URLs provide a uniform interface between domains and clients. The framework generates the Entity Data Model (EDM) for each domain when a request is made to the $metadata URL of a domain. The framework also generates the service document for each domain when requested by clients.

1. The domain is the configuration root for all domains that resides at the Windchill codebase.
2. The configuration folder contains all resources within the domain.
3. The configuration of entitles exposed by a domain is version-specific and maintained in folders for specific versions such as v1.
4. The entity folder contains configuration files for domain entities.
5. The configuration file specifies the name, identification, namespace, container name, and default version for the domain.

PROCESS METADATA



Windchill REST Services includes a set of domain configurations for specific functional areas of Windchill. The schema of Windchill consists of a broad range of functional areas. The domains are designed to enable access to smaller and functionally independent areas of this schema to RESTful clients. The servlet understands the domains; it reads domains, resource configurations, and provides a view of the domains to the client, such as Data Administration, Product Management, Document Management, and so on.

EntityConfig.json processes entity configuration. The entity delegate produces entity metadata and the CRUD operations on basic entities.

Structural Properties and Navigation Criteria

ANATOMY OF WRS

STRUCTURAL PROPERTIES AND NAVIGATION CRITERIA

These properties describe data related to the current entity type.

An entity is defined by its Structural Properties, such as a Part to a Parts List.

The value of a navigation property is the referenced entity or entity set, but only when explicitly requested.

```
        <String>READ,CREATE,UPDATE,DELETE</String>
    </Annotation>
</EntityType>
<EntityType Name="BOM">
    <Property Name="PartId" Type="Edm.String"/>
    <Property Name="PartUseId" Type="Edm.String"/>
    <Property Name="PartName" Type="Edm.String"/>
    <Property Name="PartNumber" Type="Edm.String"/>
    <NavigationProperty Name="Part" Type="PTC.ProdMgmt.Part"/>
    <NavigationProperty Name="PartUse" Type="PTC.ProdMgmt.PartUse" ContainsTarget="true
    <NavigationProperty Name="Occurrences" Type="Collection(PTC.ProdMgmt.UsageOccurrenc
    <NavigationProperty Name="Components" Type="Collection(PTC.ProdMgmt.BOM)" ContainsT
    <Annotation Term="Core.Description">
        <String>Bill Of Materials</String>
    </Annotation>
    <Annotation Term="PTC.Operations">
        <String>READ</String>
    </Annotation>
</EntityType>
<EntityType Name="MadeFromSet" BaseType="PTC.ProdMgmt.ManufacturingInformation">
```

Metadata for the BOM Type in the Product Management Domain

Metadata for the BOM Type in the Product Management Domain

An Entity Data Model (EDM) is the specification of entities that are available for a domain. The EDM contains all of the data points you can get from that domain. Each entity contains structural and navigation properties you can use to refine requests. Structural properties have values. Navigation properties, such as Uses, are references to other entities in the domain.

The EDM of a domain can be accessed by adding `$metadata` at the end of the Domain Root URL. This enables developers or the client to get more information about the entities, relationships, functions, and actions provided by the domain. For example, the URL for EDM of the Product Management domain is `https://windchill.ptc.com/Windchill/servlet/odata/ProdMgmt/$metadata`.

Navigation criteria are metadata properties that describe data related to a current entity type. If you're familiar with SAP, they work the same way in WRS. For example, `PartUse` and `Occurences` are navigation criteria for BOMs.

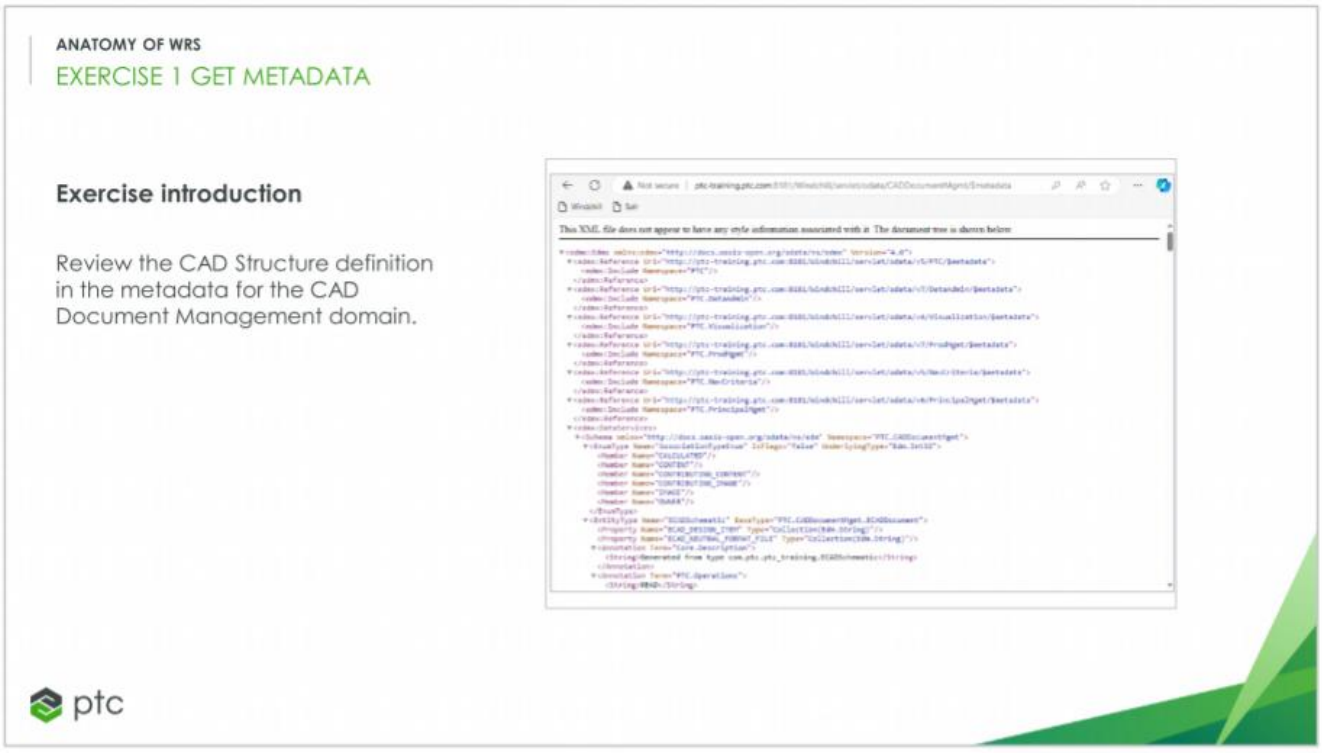
Navigation properties in OData are the reference attributes that point to another identity. A parts list is always going to be referencing a part. A part may reference a DescribedBy document, such as `{ "name": "Uses", "target": "PartUses", "type": "PartUse", "isCollection": true, "containsTarget": true, "traversal":"usedBy@wt.part.WTPartUsageLink" }`.

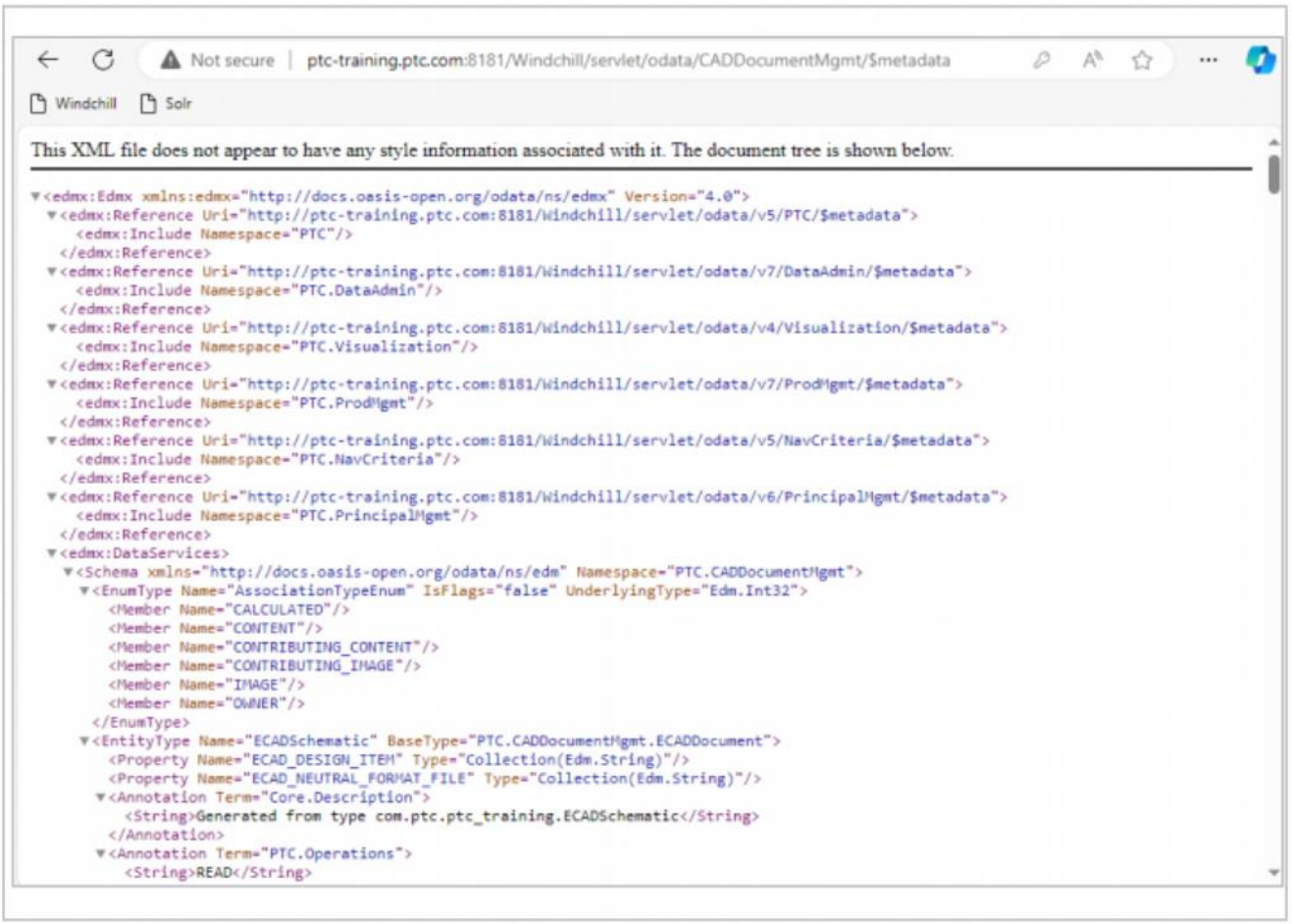
You can query for a collection of change objects, parts, or documents using the `$filter` query option on the `Context` navigation property, such as `/DocMgmt/Documents?$filter=contains(Context/Name,'<substring_of_containername>')`. The URL will return all of the available documents in all containers with a name that contains the specified substring. For example, if you specify the substring as `Engine`, then the response will contain all documents available in all containers with a name that contains "`Engine`."

The value of a navigation property is the referenced entity or entity set and only shown when explicitly requested.

In the next exercise, you will review the available entities in the CAD Document Management domain, the entities' structural properties, and navigation properties which reference other entities in the domain.

Exercise-1 GET METADATA

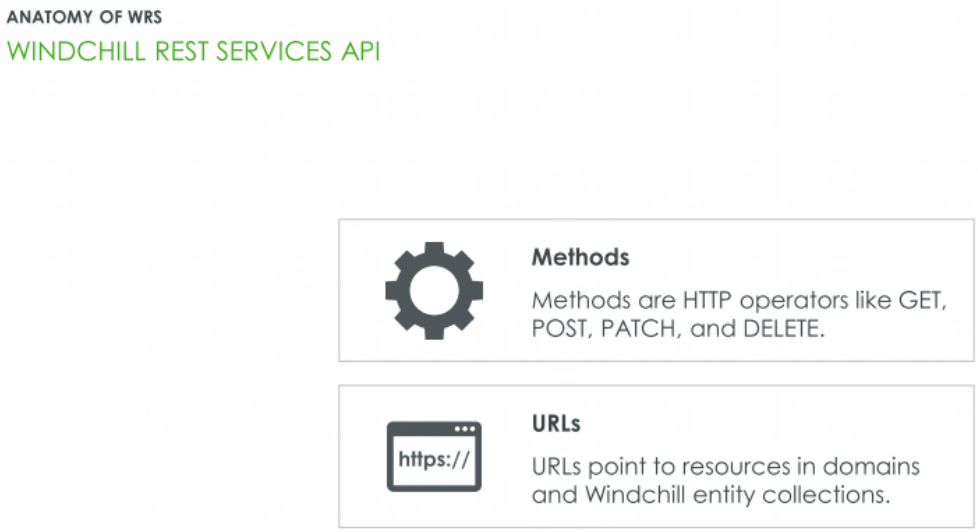




Task 1: Access the Metadata in the CAD Document Management Domain

1. Launch a web browser.
2. Go to **http://ptc-training.ptc.com:8181/Windchill/servlet/odata/CADDocumentMgmt/$metadata**.
   Note: You can paste this URL from W:\WCEC-REST-Lab-Files\WCEC-REST-Copy-Text.txt.
3. Sign in to Windchill.
   - Username: **isaha**
   - Password: **ptc**
4. Locate the `<EntityType Name="CADStructure">` definition.
5. In the definition, notice the listed Property Name values. This is just one example of the structural properties for CAD Structures in Windchill that you could request using the REST API.
6. Notice the `NavigationProperty` Name values in the `CADStructure` definition. These listed values define the relationships to other domain entities like CAD Documents and Component collections.
7. Close the browser.

Windchill REST Services API



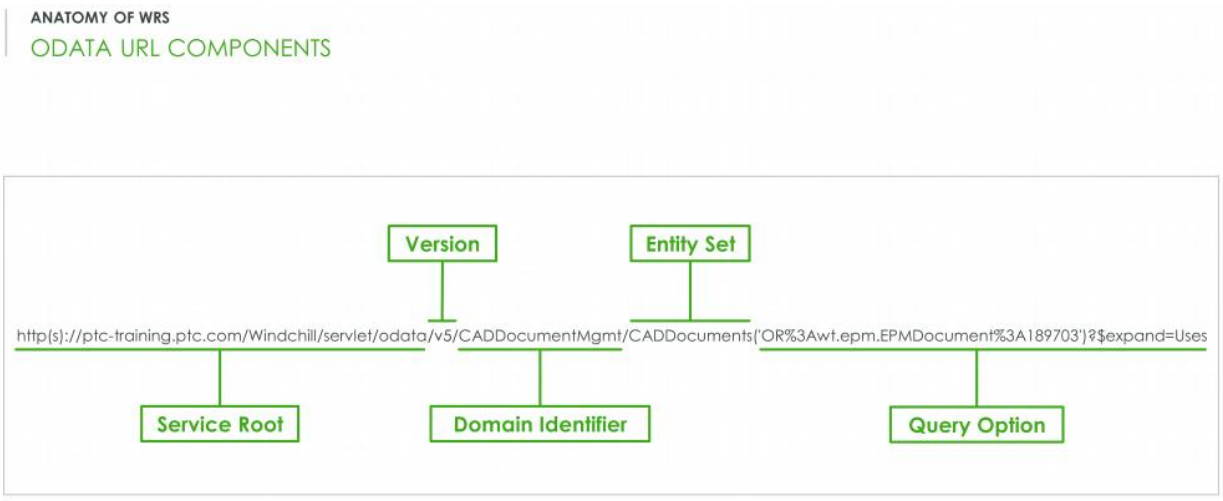
In the first exercise, you used the two primary components to WRS: a method (GET) and a URL. Methods are the standard HTTP operators; they are verbs. The URL points to a resource, which is the domain and Windchill entity collections. When you want to retrieve, modify, or delete a record, you operate on its URL using the correct HTTP method.

The Windchill server exposes the URLs of resources or entities that are known to it. The client makes different HTTP requests to those URLs to perform the create, read, update, and delete (CRUD) actions on that entity or resource that has been exposed.

Methods
These listed methods work on the resource and depend on the configuration of the REST API and the authorized capabilities of the current user. When you submit a request, you can also send along metadata in the request header to get more specific data for the user. This data should include the content type to comply with the self-descriptive messages constraints and can also contain a user agent string, accepted language string, authentication, and cache control, among others.

URLs
REST clients can access and modify the REST resources on the Windchill server through URLs.

ODATA URL Components



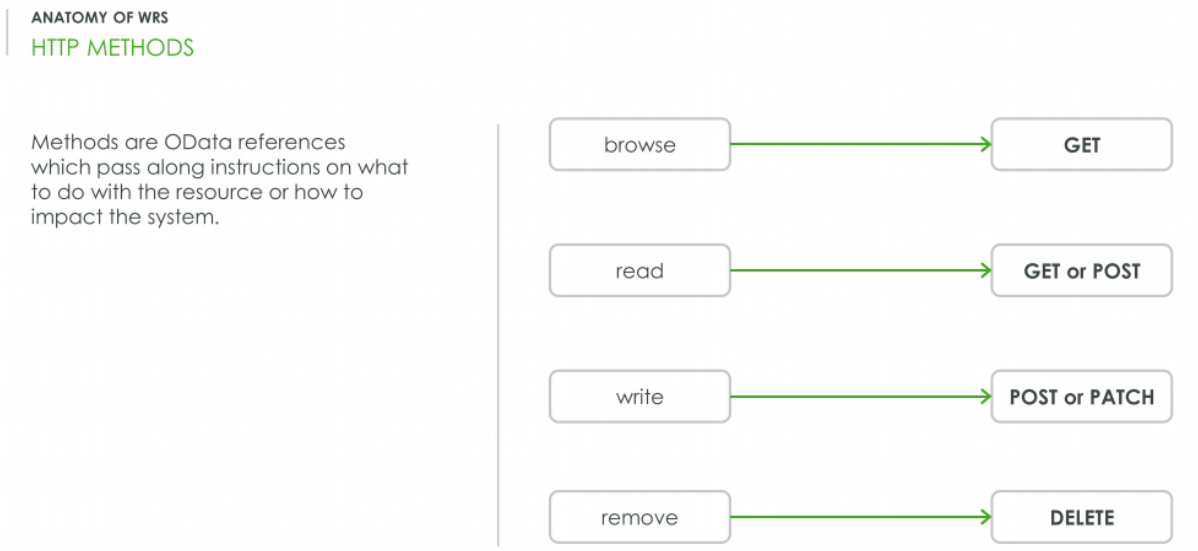
Windchill REST Services uses OData URLs to identify the resources on which you want to perform operations. A distinct characteristic of RESTful APIs like WRS is that no context is stored on the server between requests. These self-descriptive requests from a client contain all of the required information to service the request and make them easy to consume.

These URLs tell the OData framework to return JSON objects matching the set parameters. URLs contain these components:

1. Service Root – The service root includes the scheme, host, and sometimes a port number. This is the Windchill training server and architecture context.
2. Version – Like other REST APIs, WRS URLs usually include domain versions to provide a migration path for applications. It's optional to include this in the URL unless the client needs a specific version of the domain to prevent something breaking.
3. Domain Identifier – The domain identifier is part of the resource path that identifies the domain. Domains are used to access Windchill entities. In this case, the request is performed on the CAD Document Management domain.
4. Entity Set – Requests are performed on a type or collection of types in a given domain.
5. Query Options – These appended filters vary by domain and object. In this URL, the request is a specific EPMDocument with expanded navigation criteria.

The percentage-alphanumeric combinations you see are percent encoding syntax: %3A is a colon (:). You may also occasionally see a %27, which is an encoded apostrophe, and a %20 is an encoded space.

HTTP Methods



While there are dozens of HTTP methods, Windchill REST Services uses four verbs to orchestrate data: GET, POST, PATCH, and DELETE.

GET
When you type an address into your web browser, you are executing a request using the GET method. It's an on-demand, read-only function that can be repeated on the metadata of the resource. The GET method works like SELECT in SQL. It checks the Windchill type for entities contained in the entity set. In the example, you can read a part and any of its child parts you have access to in Windchill. GET is generally considered a "safe" method because no modification of the resources takes place.

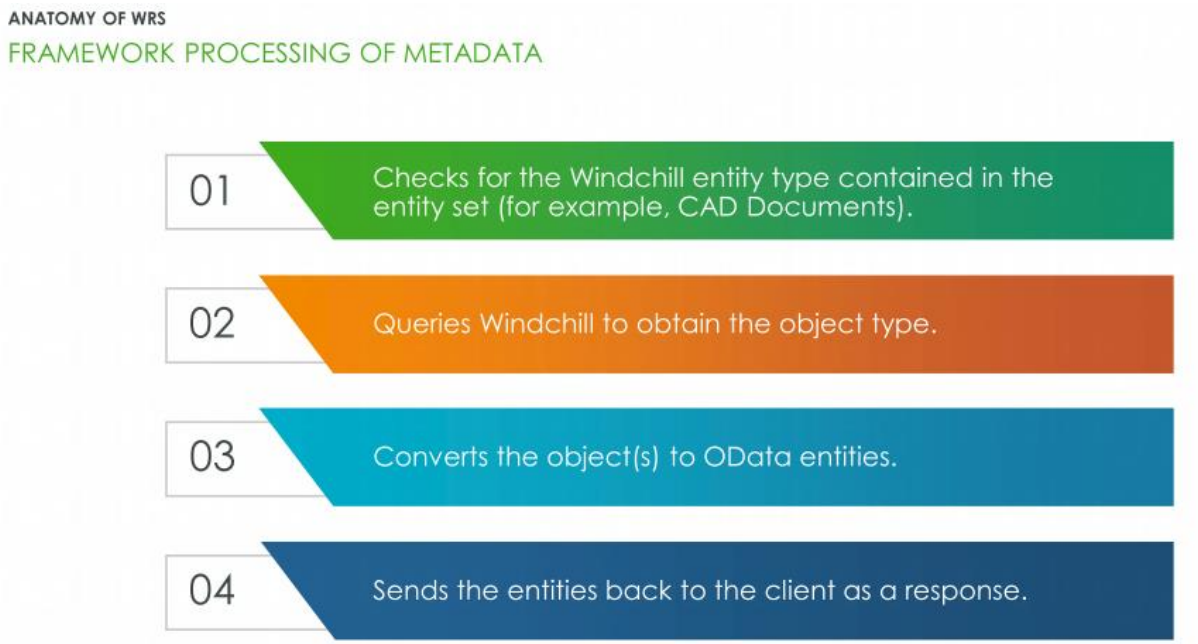
POST
If you fill out a form on a website and click the Submit button, you are sending a POST request. These methods are part of how HTTP and HTTPS requests work. This is an action; it's performing a change. POST methods are also used to create a new resource in the collection of resources. This may be an ordered read such as a multi-level part structure or process plan.

PATCH
This method is used when the client needs to update something in Windchill, like an existing manufacturing location's address. It reads the persistable and a modification to produce a new version of the resource before sending the updated representation in the response.

DELETE
Like other REST APIs, the delete method removes an entity or collection of entities. The user must have permission to delete these entities.

Not all methods work on every entity set.

Framework Processing of Metadata



In the first exercise, you looked at the body of available EDM (that is, the metadata that describes an OData service) in the CAD Document Management domain.

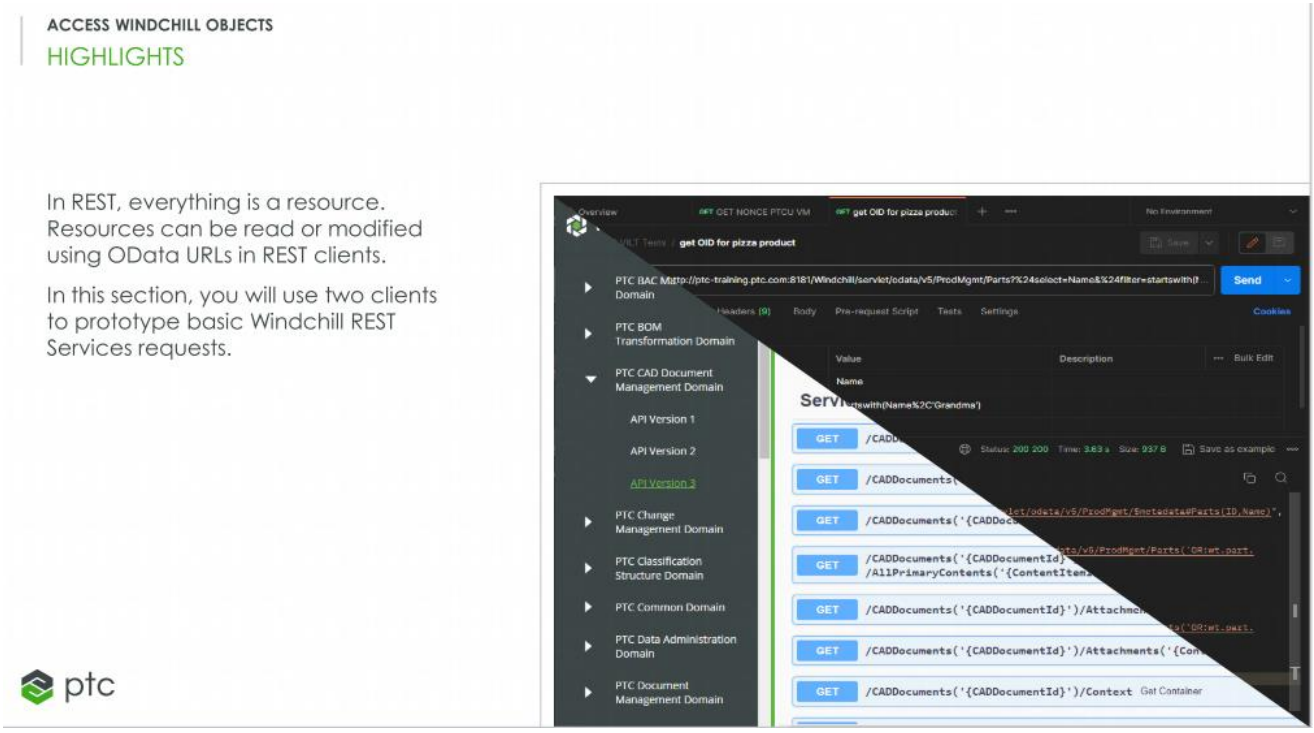
When an EDM is requested, the framework searches the domain configuration and the entities configuration for that domain. It can also append Additional information like soft attributes to base entities.

When a client requests a resource using the methods, it processes the request on entity collections and instances. Then, depending on the entity type, specialized processors will be invoked. Note that for most Windchill-backed objects, a PersistableEntityProcessor is used.

Access Windchill Objects

Overview

Highlights



WRS Testing Tools

ACCESS WINDCHILL OBJECTS

WRS TESTING TOOLS

Client	Response Format	Headers	Interactive	Params	Authentication
UI (API Catalog)	JSON	X	✓	✓	X
Browser	XML	X	X	X	✓
Command line (Curl)	stdout or other	✓	X	✓	✓
Postman (Desktop app)	JSON	✓	X	✓	✓
Third-party application code	Controlled	✓	X	✓	✓

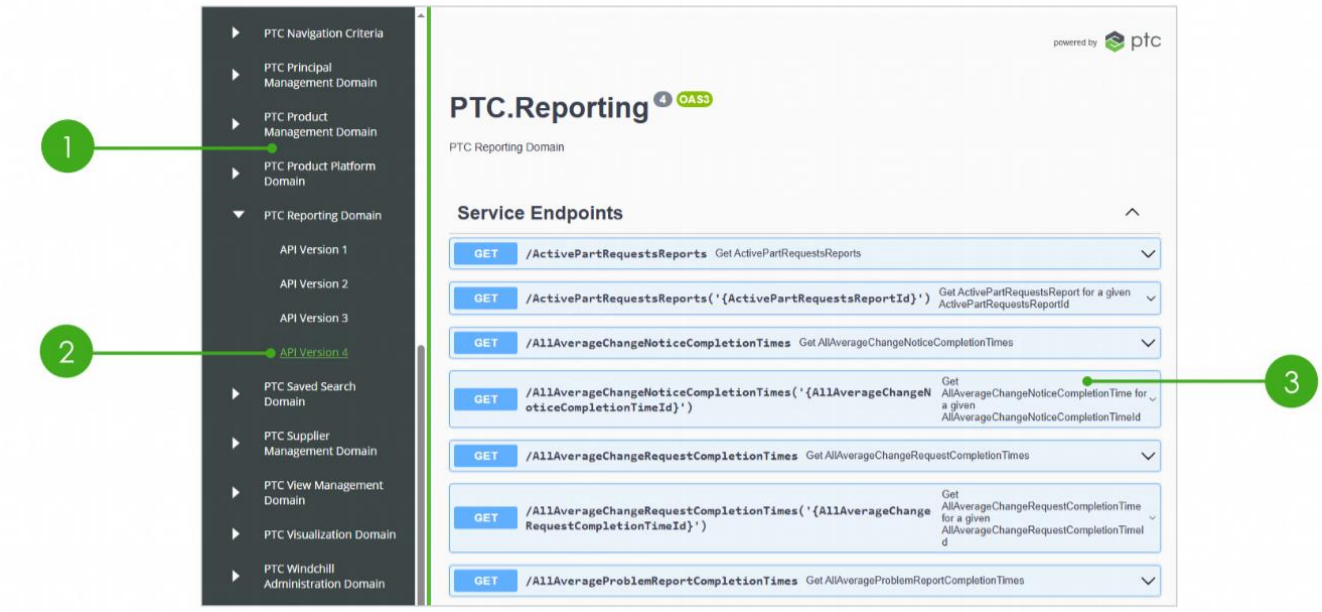
Testing your HTTP requests ahead of building an application confirms that your API is working as expected and gives you the opportunity to debug the responses that are not returning the expected values.

Third-party testing tools are useful for early prototyping. It isn't dependent upon any specific programming language or platform. If the device can send an HTTP request, it can use the Windchill REST Services API. This makes it extremely flexible, and a developer can build a prototype using any tools or technologies they want. You can execute WRS endpoints on tools like web browsers, Postman, Curl, the Yet Another Rest Client (YARC) extension in Google Chrome, and ThingWorx.

Once the API is working like you want it to, you can retrieve a copy-and-paste code to implement requests in multiple programming languages and clients.

You will use the API Catalog and Postman in the course exercises, but first you need to set a preference to make the API Catalog visible.

WRS API Catalog



The API Catalog is accessible from the Windchill Navigator if the Client Customization preference value is set to Yes. The Customization preference can only be set to visible at the Site level.

Users can interactively execute HTTP operations from the endpoints available. If the API Catalog is made visible, it is accessible to any Windchill user. There is no preference, access control list, or profile to restrict visibility to non-administrative users. It does not need to be visible to use Windchill REST Services through other clients.

The components of the API Catalog and documentation include the following:

1. Domains in Windchill REST Services can be selected in the left pane. If you do not see a domain listed, you may need to upgrade, or it may not yet be available for use. The domains listed in the catalog are extensible. You can include any custom domains in this list.
2. Windchill REST Services supports versioning just as in the OData protocol. Newer versions may include the removal of properties, changing the types of existing properties, adding or removing key properties, or reordering action or function parameters. If not included in a URL, the default version of a domain is set to the latest version. The version is repeated as a superscript at the top of the Service Endpoints pane.
3. You can interactively execute endpoints. Endpoints are connection points. They tell you the different places where you can connect to the API. Service endpoints include the method, identity set, and for many endpoints, a description. Select the arrow to expand the service endpoint and modify the parameter fields to test the response.

Exercise-2: Expose the WRS API Catalog

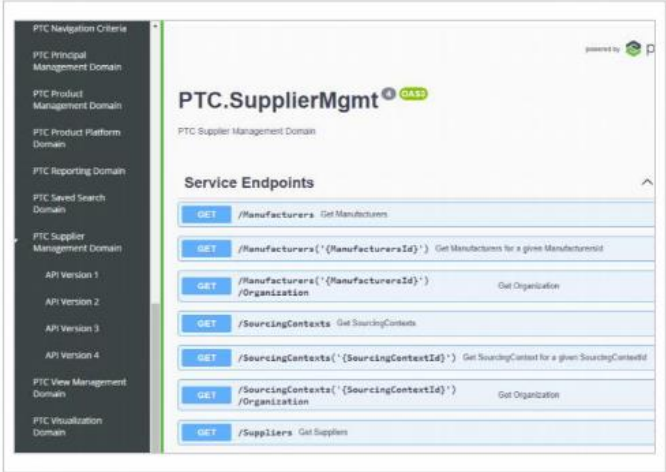
ACCESS WINDCHILL OBJECTS

EXERCISE 2 EXPOSE THE WRS API CATALOG

Exercise introduction

Switch a site-level preference to expose the WRS documentation and API catalog of service endpoints.

You will then sign in as a non-administrative user and browse the API catalog.



Task 1: Set the Client Customization Preference

1. Open a web browser and select the **Windchill** bookmark.
2. Click **Sign In**.
3. Sign in to Windchill as a site administrator.
   - Username: **wcadmin**
   - Password: **wcadmin**
4. Click the **Browse** link.
5. Browse to **Site** > **Utilities** > **Preference Management**.
6. Expand the **Client Customization** category.
7. Right-click the **Client Customization** preference and set the preference value to **Yes**.
8. Click **OK** and close the web browser.

Task 2: Access the Documentation and API Catalog

1. Open a browser and select the **Windchill** bookmark.
2. Sign in to Windchill.
   - Username: **achen**
   - Password: **ptc**
3. Click the **Browse** link.
4. Select the **Customization** icon.
   Note: You may need to refresh the browser to see this icon.
5. Select the **Documentation** link.
6. Select the **OData REST APIs** link in the API section.
7. In the WRS tab, notice the list of available Windchill domains on the left side.
8. Select the **PTC Supplier Management Domain** and notice the documentation in the right pane.
9. Select the latest **API Version** of the PTC Supplier Management Domain. Notice the list of available service endpoints.
10. Expand the **GET /SourcingContexts** endpoint.
11. Click **Try it out**, then click **Execute**.
12. Review the response body. Notice the names of the different sourcing contexts. These are different supply chain environments in the system.
13. Do not close the browser.

Query Parameters

ACCESS WINDCHILL OBJECTS

GET THE RESULTS YOU WANT WITH QUERY PARAMETERS

Query parameters can be optionally used in multiple situations, such as:

- When you access the entity sets defined in the domain.
- When you navigate to a collection of entities from a given entity.
- When you expand navigation properties of entities.

`$select`
Like SQL `select`

`$filter`
Like SQL `where` clause
EQ, NE, GT, LT, GE, LE, AND, OR, NOT, startswith, endswith, contains

`$top`
Only allows a specified number of entities to be returned in an entity set

`$skip`
Allows skipping specified number of entities from the top of an entity set

`$orderby`
Sort objects in asc or desc order, attachments, primary content, reps, containers

`$expand`
Allows expansion of related entities into the representation of an entity and nesting

WRS URLs can query parameters to get really specific when it comes to fetching or posting bits of data. The OData protocol specifies the various system query options that endpoints should accept. These can be used to filter, order, map, or paginate data. WRS supports various kinds of query options for querying data.

Query options are appended to a URL after a question mark (?) character and are separated by ampersand (&) characters. Each option consists of a name with a dollar sign ($) prefix and its value, separated by an equal sign (=). For example: `OData/Products?$top=2&$orderby=Name`. A number of logical operators and functions are defined for use when filtering data. For example: `OData/Products?$filter=Price lt 10.00 and startswith(Name,'M')` requests products with a price smaller than 10 and a name starting with the letter M.

Query options are not the same in each domain or service or entity. Different properties, like navigation properties and structural properties, can change the input here.

`$select`
Select operates like the SQL SELECT clause. It specifies the related resources to be included in line with retrieved resources and can be entered as a comma-separated list. It's often used in conjunction with a navigation property.
Example: `http://host/Windchill/servlet/odata/ProdMgmt/Parts?$select=Name,Number`

`$filter`
Filter operates like the SQL `WHERE` clause on collections (EQ, NE, GT, LT, GE, LE, AND, OR, NOT, startswith, endswith, contains). The methods startswith, endswith, and contains are not supported for object ID properties.
Example 1: `/ProdMgmt/Parts?$filter=Name eq 'CYL001'`
Example 2: `/ProdMgmt/Parts?filter=ID eq 'OR:WT.part.WTPart:115022'`

$top
Top returns only a specified number of entities in a response.
Example: `/ProdMgmt/Parts?$top=5`

`$skip`
Skip allows skipping a specified number of entities from the top of an entity set.
Example: `ProdMgmt/Parts?$skip=5`

`$orderby`
This query option sorts a collection of change objects, documents, or parts in ascending or descending order. It can also set a response to sort by attachments, primary content, representations, context, or organization. The parameter is processed after the result set is returned, so using this parameter can be resource-intensive and should not be used if it is not required.
Example: `/ProdMgmt/Parts('<OID>')/PartDocAssociations?$filter=RelatedCADDoc/ID ne 'DrwOID' and RelatedCADDoc/State/Value eq 'RELEASED'&$orderby=ID&$count=true`

`$expand`
The $expand option can specify related entities into the representation of an entity. The value of $expand can be nested or a comma-separated list of navigational properties.
Example (expand the representation to include the part's uses): `ProdMgmt/Parts?$expand=Uses`
Nested example: `ProdMgmt/Parts?$expand=Uses($expand=Uses)`
`Example: ProdMgmt/Parts?$filter=startswith(Name,'Axle')&ptc.search.latestversion=true`

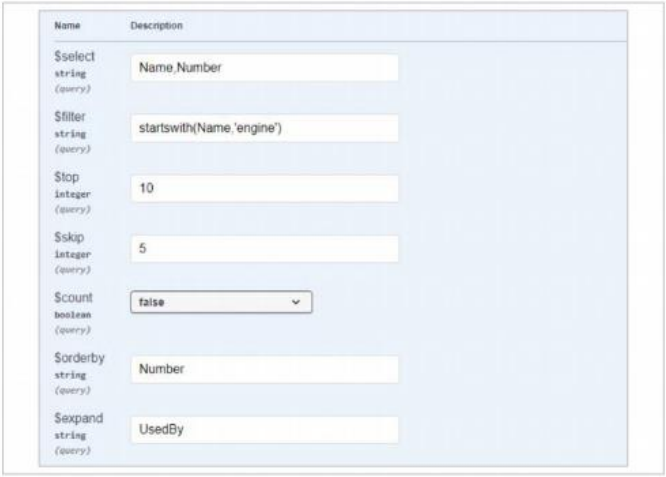
Exercise-3 Create a request in the API catalog:

ACCESS WINDCHILL OBJECTS

EXERCISE 3 CREATE A REQUEST IN THE API CATALOG

Exercise introduction

Use the WRS API Catalog and query options to retrieve a list of engine* Windchill parts from the PTC Motorcycle product.



Task 1: Get Parts with Query Options

1. In the WRS browser tab, browse to **PTC Product Management Domain** > **API Version 7**.
2. Scroll to the **GET /Parts (Get Parts)** service endpoint and expand it.
   Note: The endpoint appears immediately after the /ManufacturerParts endpoints.
3. Select **Try it out**.
4. Type the parameters:
   - $select: **Name,Number**
   - $filter: **startswith(Name,'engine')**
   - $top: **10**
   - $skip: **5**
   - $orderby: **Number**
   - $expand: **UsedBy**
5. Execute the request.

Task 2: Review the Response

1. Review the responses and request information:
   - The Curl request can be copied and pasted in the cURL command line interface.
   - The Request URL can be copied and used in another client or in a web interface.
   - The Response body contains all the information in your system that meets the request.
   - The code 200 is the status code. This means the request was successful.
   - Response headers contain the parameters of how to interpret the request.
   - The Responses are example values you can append as query parameters.
2. Review the Response body and name:value pairs of the object ID, Name, and Number for each WTPart.
3. Scroll to the Code/Description area. Notice that the media type is application/json.
4. In the Responses Description, click **Schema** and expand the PTC.ProdMgmt.Part entity properties.
5. Expand any PTC.EnumType schema description in the list.

6. Do not close the browser.

Additional information

- Query parameters for any REST requests must be typed exactly. "UsedBy" will return applicable results while parameters like "usedby" or "used By" will return a 4xx error.
- Use proper encoding for parameter values, such as encoding spaces as %20.

Results:

Responses

Curl

```
curl -X 'GET' \
  'http://ptc-training.ptc.com:8181/Windchill/servlet/odata/v7/ProdMgmt/Parts?%24select=Name%2CNumber&%24filter=startswith%28Name%2C
  -H 'accept: application/json'
```

Request URL

```
http://ptc-training.ptc.com:8181/Windchill/servlet/odata/v7/ProdMgmt/Parts?
%24select=Name%2CNumber&%24filter=startswith%28Name%2C%20%27engine%27%29&%24top=10&%24skip=5&%24count=false&%24orderby=Number&%24e
xpand=UsedBy
```

Server response

Code	Details
200	Response body

Server response

Code	Details
200	Response body

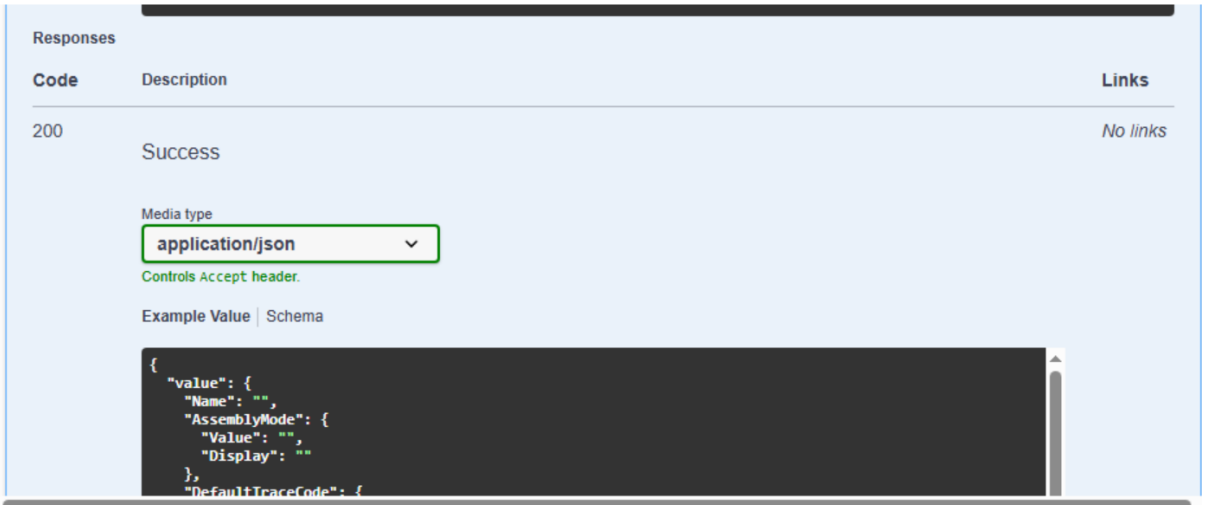

```
{
  "@odata.context": "http://ptc-training.ptc.com:8181/Windchill/servlet/odata/v7/ProdMgmt/$metadata#Parts(ID,Name,Number)",
  "value": [
    {
      "@odata.id": "http://ptc-training.ptc.com:8181/Windchill/servlet/odata/v7/ProdMgmt/Parts('OR:wt.part.WTPart:1392337')",
      "ID": "OR:wt.part.WTPart:1392337",
      "Name": "engine_housing_sm",
      "Number": "0000000163",
      "UsedBy": [
        {
          "CreatedOn": "2023-07-14T14:19:57Z",
          "ID": "OR:wt.part.WTPart:1473884",
          "LastModified": "2023-07-14T14:20:01Z",
          "AlternateNumber": null,
          "AssemblyMode": {
            "Value": "separable",
            "Display": "Separable"
          "TypeIcon": {
            "Path": "http://ptc-training.ptc.com:8181/Windchill/wtcore/images/part.gif",
            "Tooltip": "Part"
          },
          "Version": "A.1 (Manufacturing)",
          "VersionID": "VR:wt.part.WTPart:1473866",
          "View": "Manufacturing",
```


Download

Response headers

```
connection: Keep-Alive
content-encoding: gzip
content-type: application/json;odata.metadata=minimal
date: Wed,12 Nov 2025 05:52:08 GMT
keep-alive: timeout=5,max=100
odata-version: 4.0
ptc-applied-container-context: Global
server: Apache
transfer-encoding: chunked
vary: Accept-Encoding,User-Agent
x-content-type-options: nosniff
x-do-not-compress-this: 1
x-frame-options: SAMEORIGIN
x-ptc-connected: 1
```

Responses



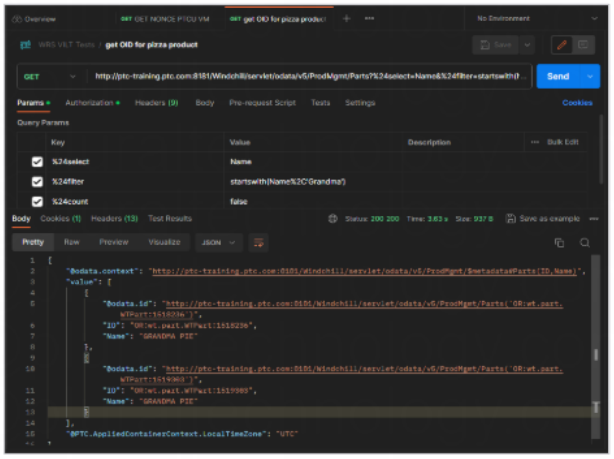
Postman

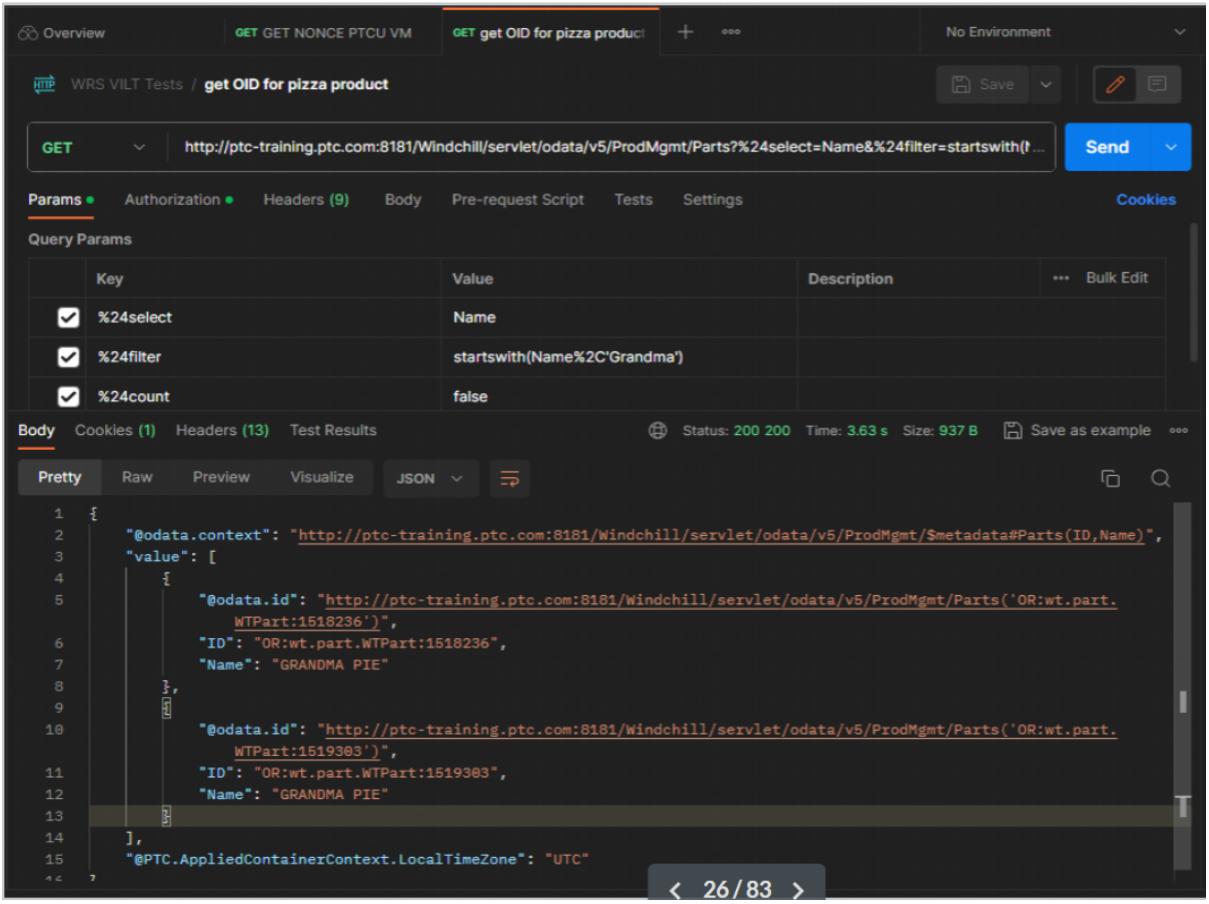
ACCESS WINDCHILL OBJECTS

POSTMAN

You can use third-party testing tools to configure your request to your specificity: method, headers, and body.

Requests can be shared or saved in collections and repeated for use in applications.





Postman is a powerful tool to prototype requests in Windchill REST Services.

You set up the proper inputs, like the header, URL, and body, then call the service.

HTTP Response Headers

ACCESS WINDCHILL OBJECTS

HTTP RESPONSE HEADERS

REST requests are stateless, meaning everything must go with the request itself to interpret the message.

Details are sent as headers, which contain name/value pairs written in JSON, a string that converts to a native JavaScript object when transmitted across a network.

Key	Value
Authorization	Basic d2NhZG1pbjp3Y2FkbWlu
Postman-Token	<calculated when request is sent>
Content-Type	application/json
Content-Length	<calculated when request is sent>
Host	<calculated when request is sent>
User-Agent	PostmanRuntime/7.31.3
Accept	*/*
Accept-Encoding	gzip, deflate, br
Connection	keep-alive
CSRF_NONCE	{{CSRF_NONCE}}

In the last exercise, you may have noticed that WRS uses response headers which appear below the response body in the API Catalog. Headers contain name/value pairs (sometimes key/value) sent to the server to interpret the message. Statelessness is a common architectural constraint of RESTful APIs. Since the requests are self-descriptive, any request from a client contains all of the required information to service the request. This makes requests easy to consume, too.

In Windchill REST Services, there are several headers that are necessary to make many WRS requests:

- Authorization: Access control rules apply in WRS.
- Content-Type is the format of the data being sent to the server. When working with Windchill, this is always JSON.
- CSRF token, or a nonce token, which contains the Key ID that authenticates the request. Any request without a valid CSRF token and proper permissions in Windchill will fail.

Other headers may be present but hidden by the tool used to prototype.

HTTP Request Body

The body of a request can contain any input parameters needed for a successful request. These parameters must be sent in JSON format.

In the example, the API expects the following three input parameters to create a new document: the Name of the object, a description, the content title, and the container in which it will reside once the request is processed. Your REST API client must send the parameters in JSON format.

HTTP Status Codes

ACCESS WINDCHILL OBJECTS

HTTP STATUS CODES

When you send a WRS request, the server responds with the request completion time and a numeric status code.

If the request was successful, the status code is 200 or 200 OK.

If you receive any other status code, the response indicates that the request failed and *may* explain why it failed.

200 or 2xx	Success
400 Bad Request	Generic user error
401 Unauthorized	Good request, use WWW-authenticate
500 Internal Server Error	Generic failure

REST Services error messages are generic. It is up to the clients (REST applications) to define friendly error messages based on the status code returned.

Exercise-4: Read & Review Responses in Postman

ACCESS WINDCHILL OBJECTS

EXERCISE 4 READ AND REVIEW RESPONSES IN POSTMAN

Exercise introduction

As Anna Chen, populate a service endpoint with query options and read the response.

Task 1: Copy the URL from the API Catalog

1. Return to the WRS API Catalog browser tab and the Get Parts service endpoint.
2. Copy the Request URL listed in the endpoint from the previous exercise.

Task 2: Set the URL in Postman

1. Launch **Postman** from the desktop.
2. In the Untitled Request tab, ensure the method is set to **GET**.
3. Paste the URL from the previous task in the address bar. Remove any trailing spaces.
4. In the **Params** tab, notice the query options are automatically populated.
5. Notice that the URL, keys, and values contain encoding syntax.

Task 3: Set the Headers

1. Select the **Authorization** tab.
2. Select **Basic Auth** as the Type.
3. Authorize the following user:
   - Username: **achen**
   - Password: **ptc**
4. Select the **Headers** tab.
5. In the Key column, type **Content-Type**.
6. In the Value column, type **json** and select **application/json** from the list.

Task 4: Execute the Request

1. Send the request.
2. Review the Status field to determine if the request was successful.
3. Review the response in the Body pane. Notice that it is identical to the response generated in the API Catalog.
4. Close the web browser signed in to Windchill as achen.

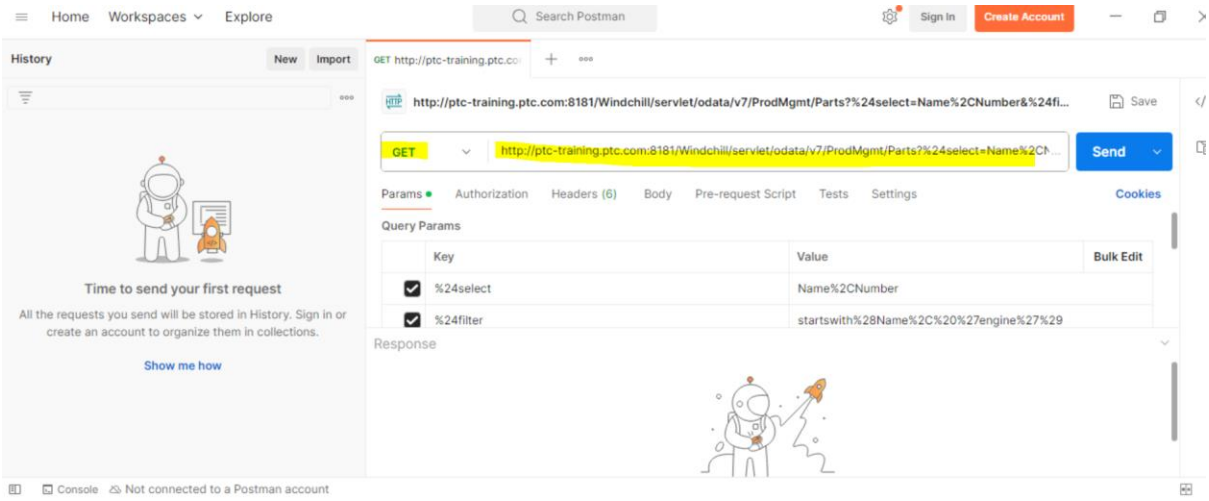
Additional information

You are using the Lightweight version of the Postman client. If you have Postman credentials and want to sign in to save a collection of requests, select the Windows Start menu > System Settings > Default apps. Change the web browser option to a browser other than Internet Explorer, then sign in from the Postman client.

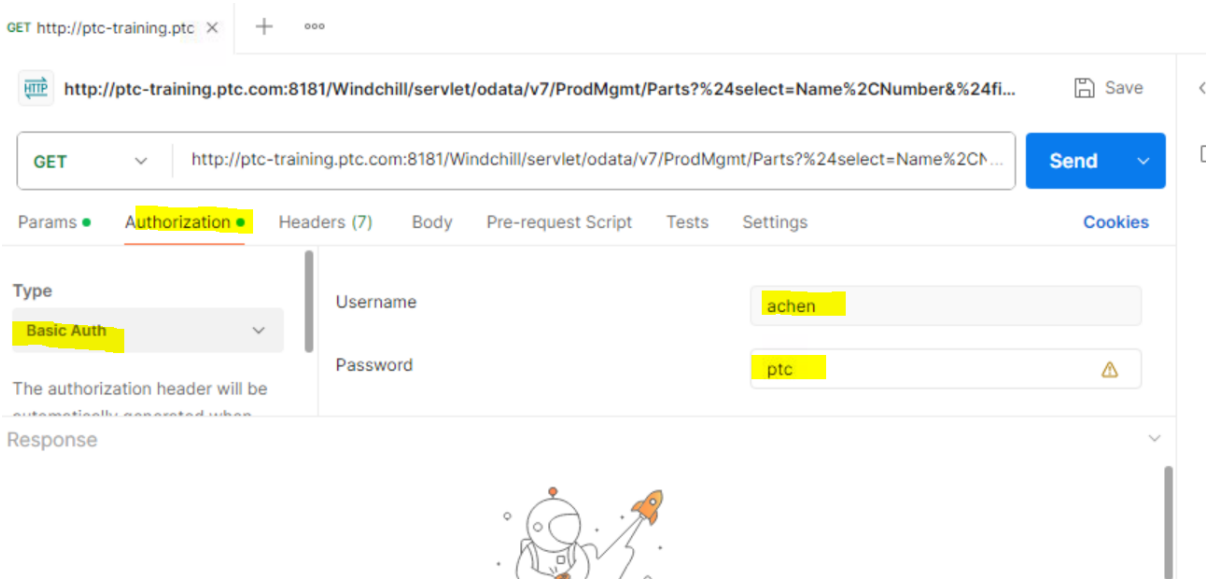
Results:

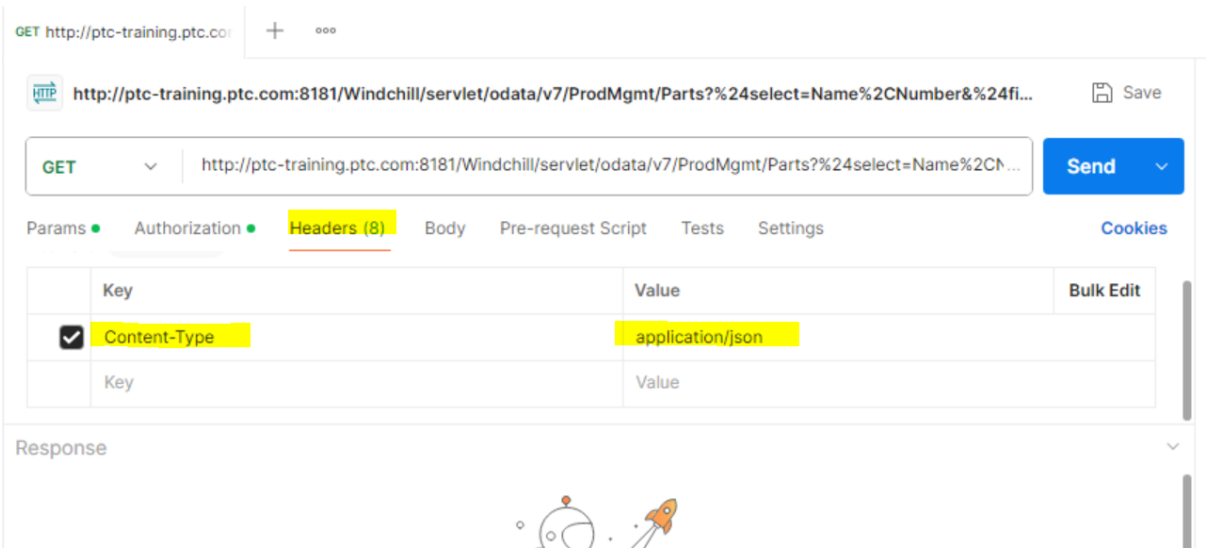
Task-1 Result Request URL:

Task-2 Result:

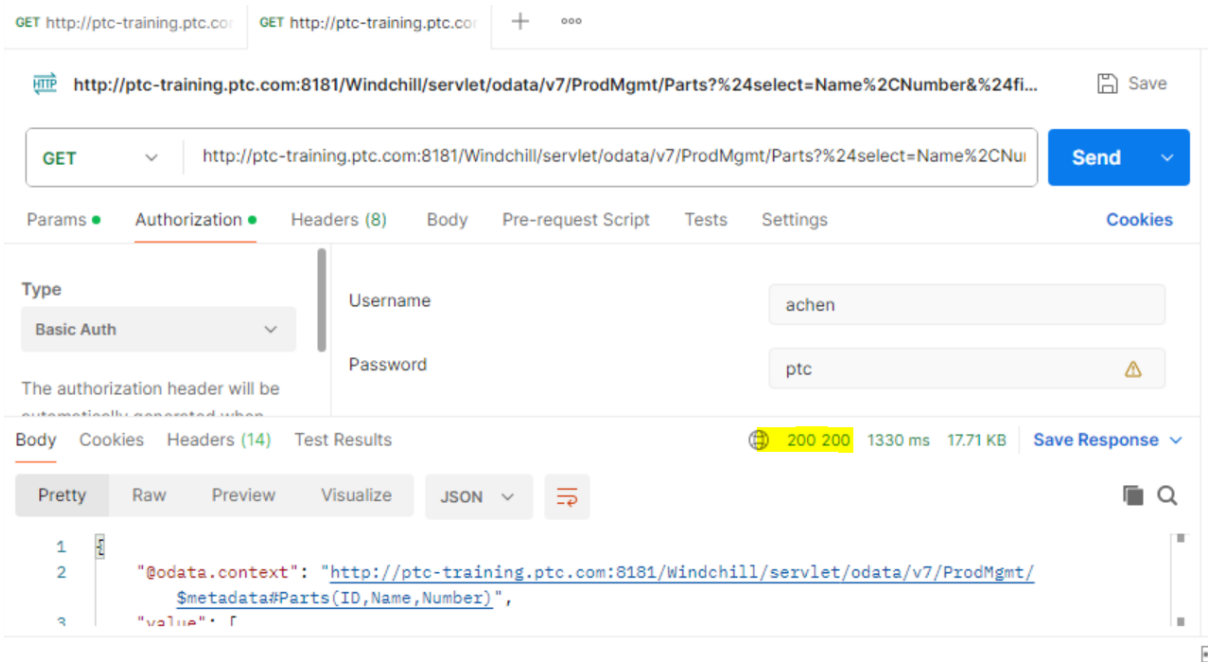


Task-3 Results:

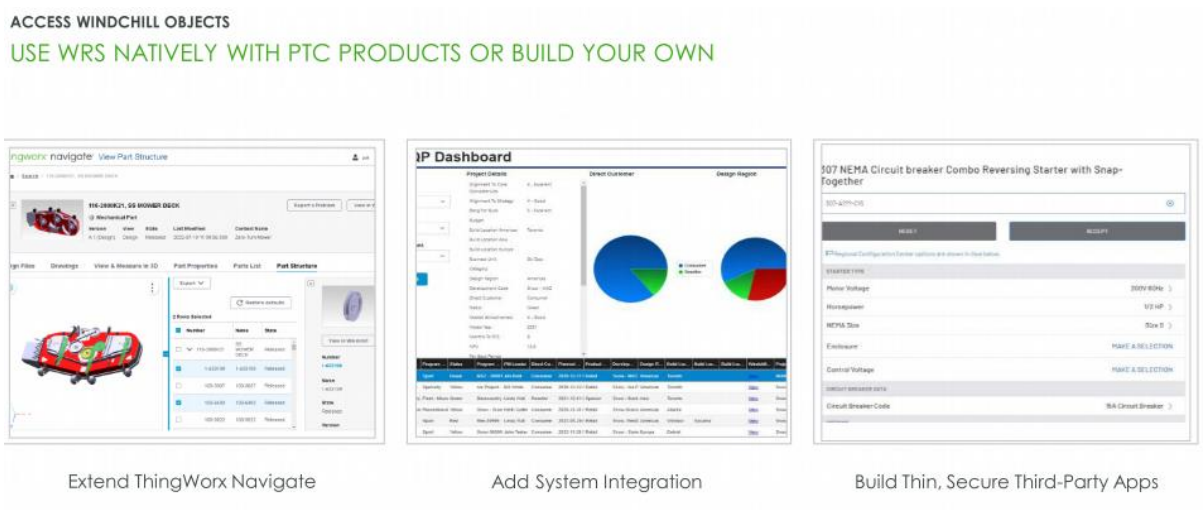




Task-4 Result:



Use WRS Natively with PTC Products or Build Your Own



What do you do with the URLs and responses achieved in different clients?

You can use the data natively with PTC products, such as integrating Windchill representations into Vuforia experiences, ThingWorx applications to create data in Windchill, and more. Windchill REST Services can link identified data across systems to keep systems in sync. You can build digital infrastructure to fetch and consume traced remote data (metadata and associated content) in Windchill. This can eliminate duplicated efforts, provide multiple sources of truth, and reduce data inaccessibility.

Yes, you can extend use of ThingWorx Navigate custom apps with WRS. These apps are built using the ThingWorx platform and integration connectors. ThingWorx reusable components contain the REST hooks needed to create and use Navigate apps. Other integration connectors can aggregate data from Windchill and other systems like SAP, SalesForce, or PowerBI.

WRS offers a lot of flexibility in its architecture. And because there is support for security solutions like OAuth2, OpenID connect, SSO, and so on, you can develop thin and platform-independent applications.

Design Challenge

Overview

WINDCHILL: REST SERVICES

SECTION OVERVIEW

1 ANATOMY OF WRS

2 ACCESS WINDCHILL OBJECTS

3 DESIGN CHALLENGE

4 MULTI-OBJECT REQUESTS

5 CUSTOMIZATIONS

6 AUTHENTICATE END USERS

Highlights

DESIGN CHALLENGE

HIGHLIGHTS

Windchill REST Services can orchestrate data to and from Windchill. You will review your use case in this section and the clients you will use to prototype your REST calls.

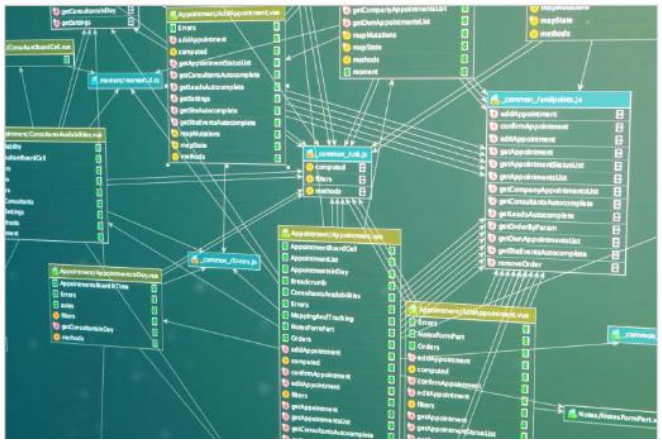
Planning the Windchill Use Case

DESIGN CHALLENGE

PLANNING THE WINDCHILL USE CASE

What should the application support?

1. Manufacturing engineers need a bill of materials and part quantities.
2. Manufacturing engineers need process plans with operations and allocated parts.
3. Production personnel need to communicate order completion.



Exercise Use Case – Pizza Manufacturing Solution

DESIGN CHALLENGE

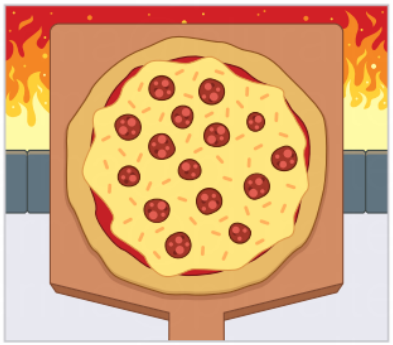
EXERCISE USE CASE – PIZZA MANUFACTURING SOLUTION

1. Review Order and Ingredients

NONCE token

Product Management domain

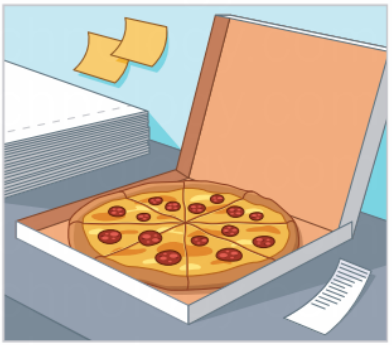
POST GetProductStructure



2. Build and Bake

Manufacturing Process Management domain

GET Process Plans



3. Set for Delivery

Product Management domain

Execute SetState to Released

We will test three WRS requests to help pizza engineers build a pizza from start to finish.

Instructor Demo

DESIGN CHALLENGE

INSTRUCTOR DEMO

Review Pizza Product Details in Windchill

Task 1: Access the Pizza Product

1. Sign in to Windchill as **isaha/ptc**.
2. Go to the **Pizza** product folders.
3. Open the **Engineering** folder and browse the three end items and their part structures.
4. Go to the **Manufacturing** folder and notice the parts in a Manufacturing view.
5. Go to the **Mfg Resources** folder and notice the resources.

Knowledge Check

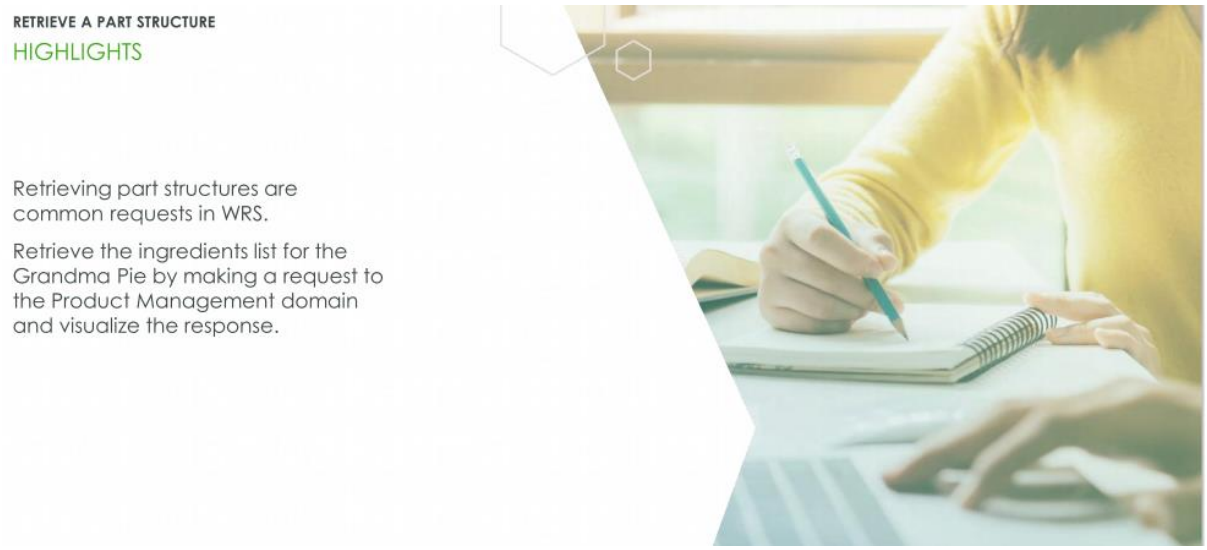
DESIGN CHALLENGE

KNOWLEDGE CHECK

1

What are the four methods used in combination with OData URLs to access Windchill objects?

Retrieve a Part Structure: Highlights



Retrieve a Part Structure: Ex Use Case – Pizza Manufacturing Solution

Retrieve a Part Structure: Secure a WRS Requests

RETRIEVE A PART STRUCTURE

SECURE WRS REQUESTS

Requests that create, update, or delete entities require a nonce (or nonce token) as a header.

Nonces prevent replay attacks that rely on impersonating prior communications to gain access.

The returned value verifies the user credentials attached to it.

```
GET http://ptc-training.ptc.com:8181/Windchill/servlet/odata/PTC/GETCSRFToken()
```

200 OK

```
{
    "@odata.context": "…/Windchill/servlet/odata/v3/PTC/$metadata#PTC.CSRFToken",
    "@PTC.AppliedContainerContext.LocalTimeZone": "UTC",
    "NonceKey": "CSRF_NONCE",
    "NonceValue": "zdlWBcTCGrSYOY8CjJoQQbK0K/n5FsNt5roadKz0Uf/VYdds+rhhSp2bSuLPY95Xv6BjMYWsbYmiCLk1+OhnNPTwKYaoDbUwn7BkSaCVKPjWXOVAipEydo2xbYD1S+xuleAvY5Twdfb+e8V4rp0lPav2Jw==“

}
```

To get the Pizza product's part structure, you need to make a POST request in Postman. POST methods, along with any other method that modifies, updates, or deletes entities, requires a random authentication protocol called a nonce. These randomly-generated strings prevent a malicious client from storing signed-in requests and resubmitting them again later.

Nonce strings expire after 24 hours. You can set up an environment variable or global variable in your prototyping tool or client to act as a placeholder to repeat a nonce across other requests.

Nonce is similar to an appkey in ThingWorx: the token is bound to the client to temporarily execute APIs that a user has permission to run.

For prototyping, you will use it in combination with a basic authentication (username and password) method.

Retrieve a Part Structure: Ex-5 Fetch a Nonce Token

RETRIEVE A PART STRUCTURE

EXERCISE 5 FETCH A NONCE TOKEN

Exercise introduction

Make a request to the PTC Common Domain to fetch a nonce value for use in action-related requests.

ptc

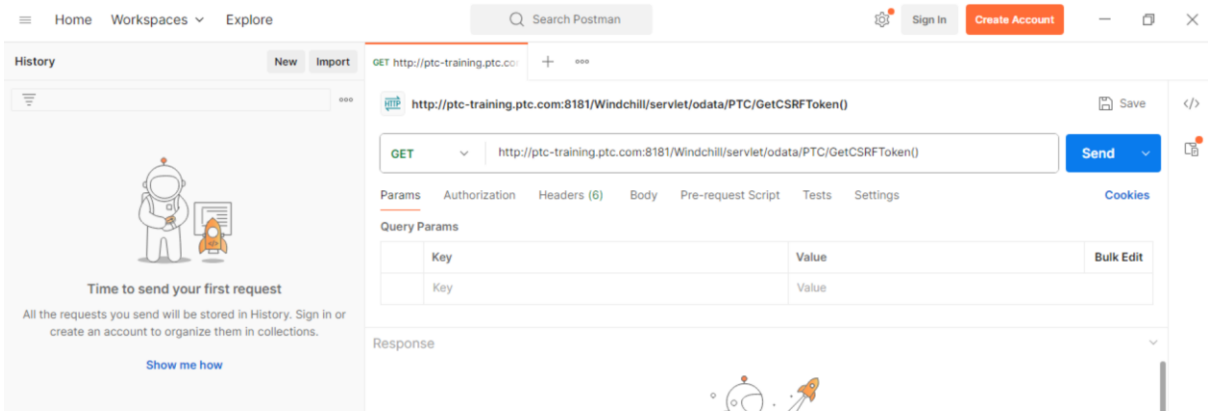
Task 1: Set the URL and Authorization in Postman

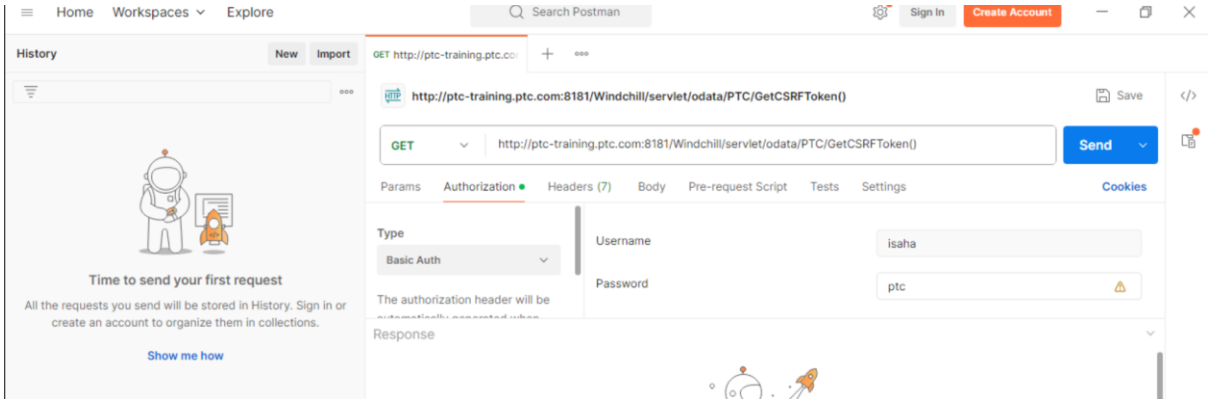
1. In Postman, click the **+** icon to create an Untitled Request.
2. Set the HTTP method to **GET**.
3. Set the URL to **http://ptc-training.ptc.com:8181/Windchill/servlet/odata/PTC/GetCSRFToken()**.
   Note: You can paste this URL from W:\WCEC-REST-Lab-Files\WCEC-REST-Copy-Text.txt.
4. In the Authorization tab, set the Type to **Basic Auth**.
5. Authorize the following user:
   - Username: **isaha**
   - Password: **ptc**

Task 2: Execute the Request and Save the Value

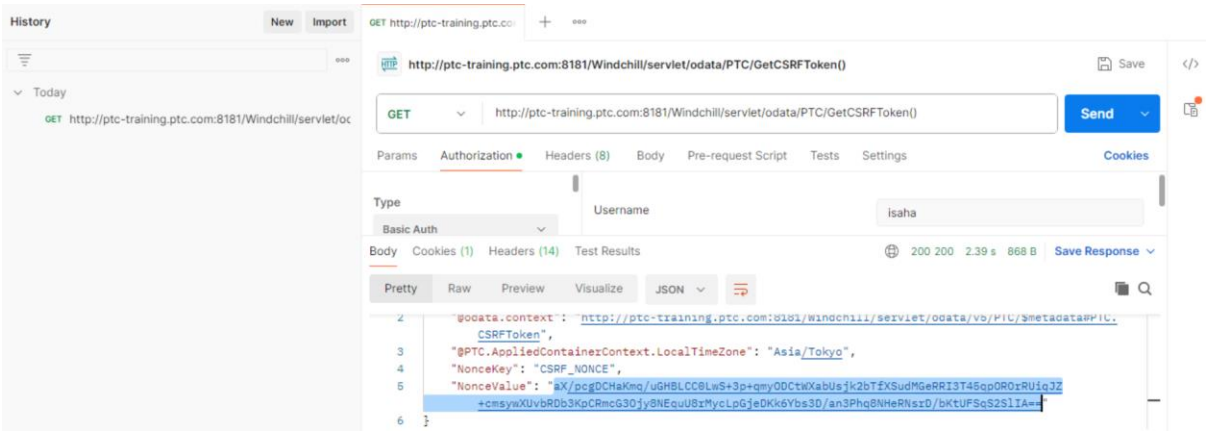
1. Send the request.
2. Verify the return status is 200.
3. In the response body, copy the NonceValue that appears between the quotation marks ("x") into a Notepad file. You will need this value for later exercises.
4. Do not close the Postman tab.

Task 1 Result:

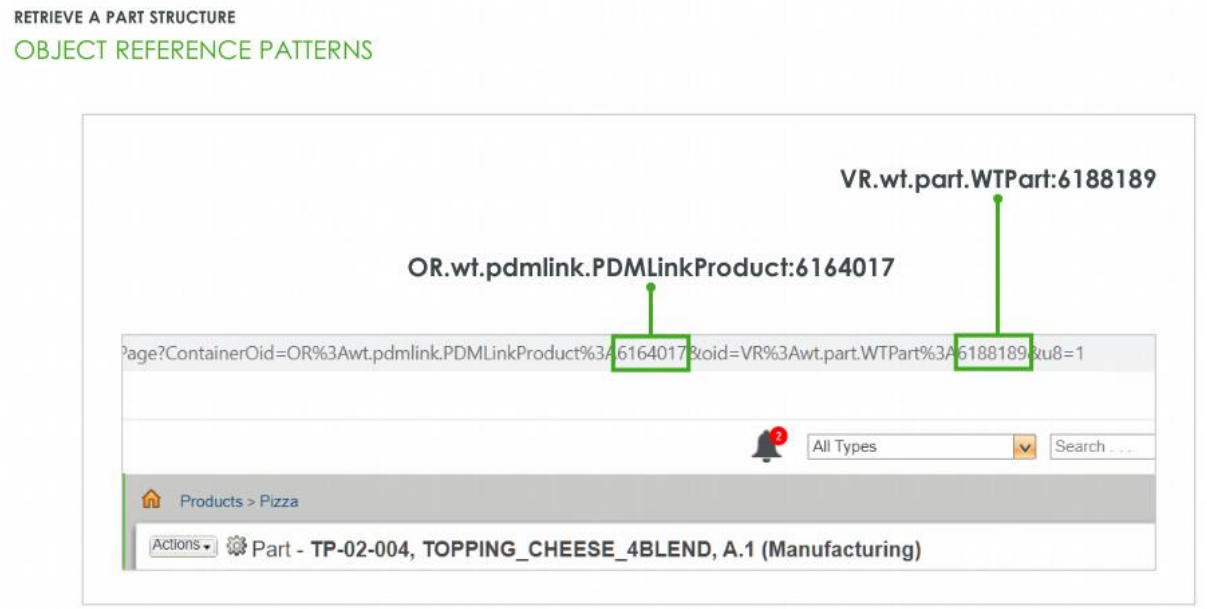




Task 2 Results:



Retrieve a Part Structure: Object Reference Patterns



Parts and Documents are persistable objects in Windchill. A reference is a lightweight handle to these persistable objects in the Windchill database. Each object reference has a string representation in the database of `<class>:<key:>` syntax. These strings are

Object Identifiers (or OID) are strings used to construct a URL for the Windchill object's Information page, but they are also used in OData URLs. Depending on the type of request you want to make, there are two reference patterns that may be applicable, depending on the operation needed.

1. Object Reference (OR:<class>:<key>) – This is an object's identifier used in the construction of a URL for the Windchill object's Information page, such as `OR:wt.part.WTPart:54321`.
2. Version Reference (VR:<class>:<key>) – This is an object iteration for iterated objects (that can be checked out and checked in), such as `VR:wt.part.WTPart:12345`. The VR always leads to the latest iteration in a branch.

These object references can be retrieved from the Details tab of an object's Information page. For example, if the URL contains `oid=OR%3Awt.part.WTPart%3A514559&u8=1`, you can convert the `%3A` encoding for a :, then the object identifier is `OR:wt.part.WTPart:514559`.

You can use an Object Identifier to specify how much information is called and returned to the client, and also expand multiple levels of containers and folders by requesting /<domain>/Containers('<oid>')?$expand=Folders($levels=max).

Retrieve a Part Structure: Ex-6 Get Object ID For a Part

RETRIEVE A PART STRUCTURE

EXERCISE 6 GET OBJECT ID FOR A PART

Exercise introduction

The Pizza product has two Views: Design and Manufacturing. Each View has a different object ID (OID) value.

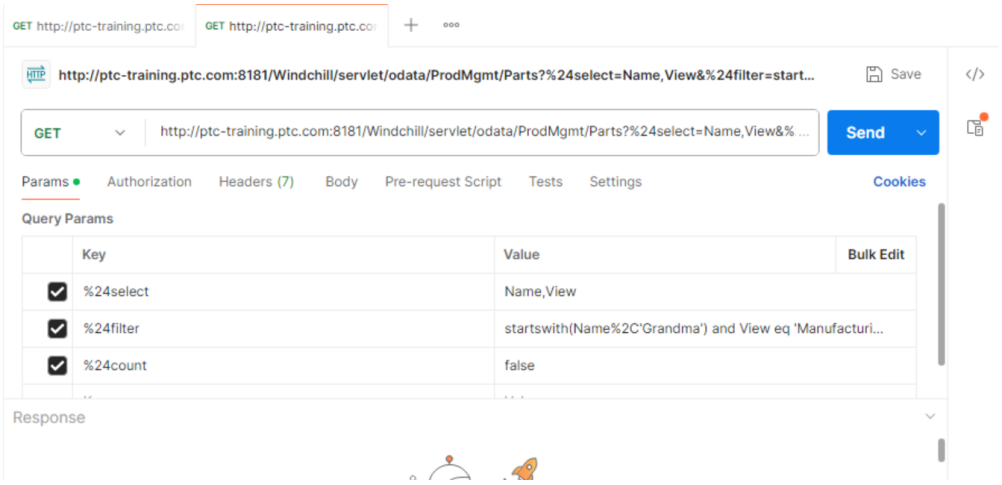
Filter the Name and View of the Grandma Pie to get its ObjectReference identifier (OID).

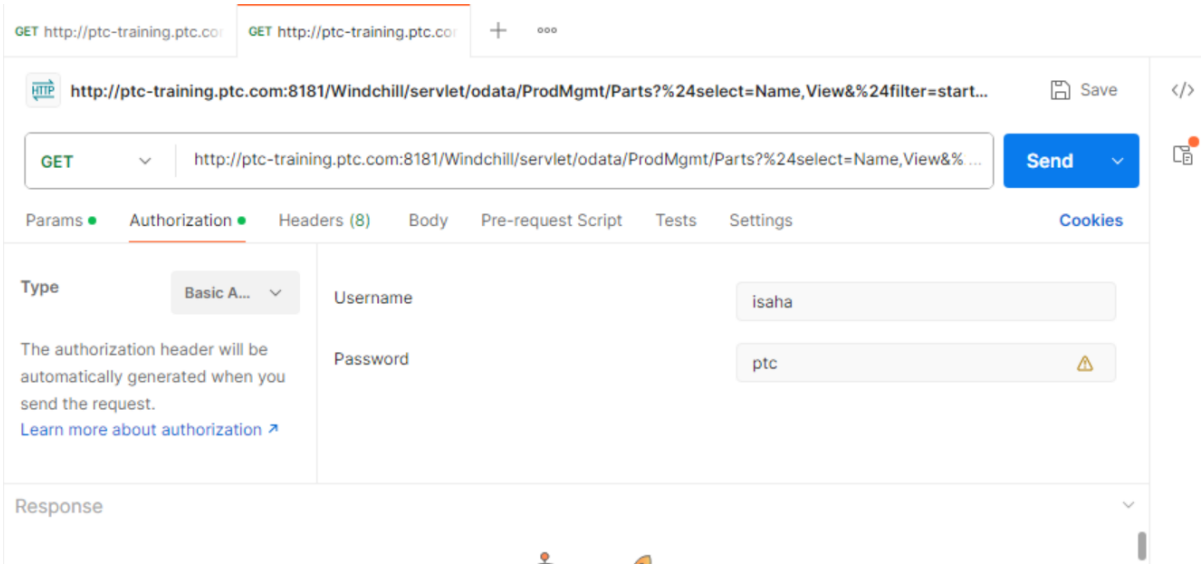
You will then use this OID in the next exercise.

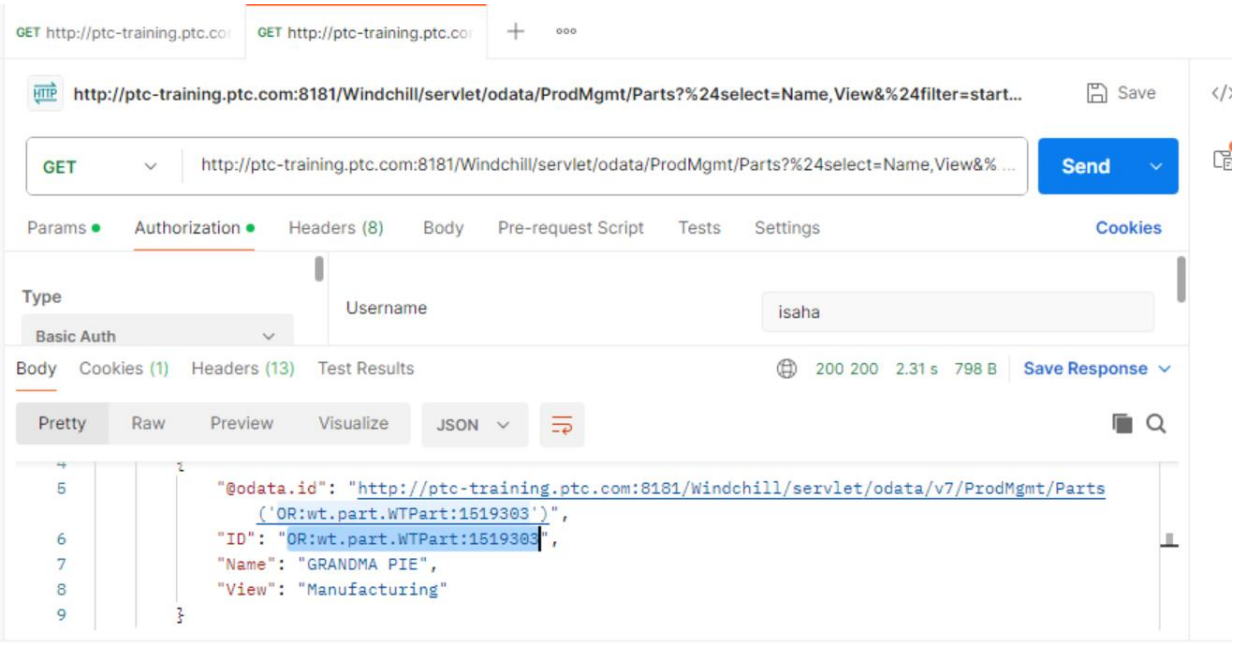
Task 1: Get Parts in Postman

1. In Postman, click the **+** icon to create a new Untitled Request.
2. Set the method to **GET**.
3. Set the URL to **http://ptc-training.ptc.com:8181/Windchill/servlet/odata/ProdMgmt/Parts?%24select=Name,View&%24filter=startswith(Name%2C'Grandma') and View eq 'Manufacturing'&%24count=false**.
   Note: You can paste this URL from W:\WCEC-REST-Lab-Files\WCEC-REST-Copy-Text.txt.
4. Select the **Params** tab and notice the query parameters for $select, $filter, $count, and their respective values.
5. In the Authorization tab, set the Type to **Basic Auth**.
6. Authorize the user.
   - Username: **isaha**
   - Password: **ptc**
7. Send the request.
8. Examine the request for a 200 status code.
9. In the response, locate the object ID value. You do not need to copy this value.

Task 1 Result:




GET http://ptc-training.ptc.co
GET http://ptc-training.ptc.co
http://ptc-training.ptc.com:8181/Windchill/servlet/odata/ProdMgmt/Parts?%24select=Name,View&%24filter=start...
Save
GET
http://ptc-training.ptc.com:8181/Windchill/servlet/odata/ProdMgmt/Parts?%24select=Name,View&%
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
Type
Basic A...
The authorization header will be automatically generated when you send the request.
Learn more about authorization
Username
isaha
Password
ptc
Response


GET http://ptc-training.ptc.co
GET http://ptc-training.ptc.co
http://ptc-training.ptc.com:8181/Windchill/servlet/odata/ProdMgmt/Parts?%24select=Name,View&%24filter=start...
Save
GET
http://ptc-training.ptc.com:8181/Windchill/servlet/odata/ProdMgmt/Parts?%24select=Name,View&%
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
Type
Basic Auth
Username
isaha
Body
Cookies (1)
Headers (13)
Test Results
200 200
2.31 s
798 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
5 "@odata.id": "http://ptc-training.ptc.com:8181/Windchill/servlet/odata/v7/ProdMgmt/Parts('OR:wt.part.WTPart:1519303')",
6 "ID": "OR:wt.part.WTPart:1519303",
7 "Name": "GRANDMA PIE",
8 "View": "Manufacturing"
9 }

Retrieve a Part Structure: Ex-7 Retrieve the Ingredients List

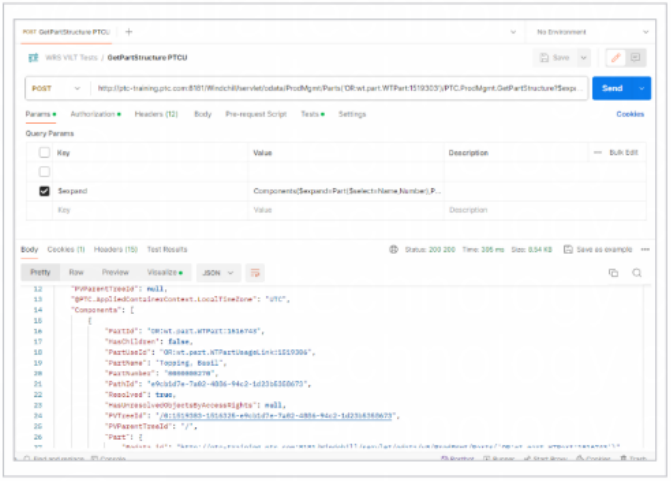
RETRIEVE A PART STRUCTURE

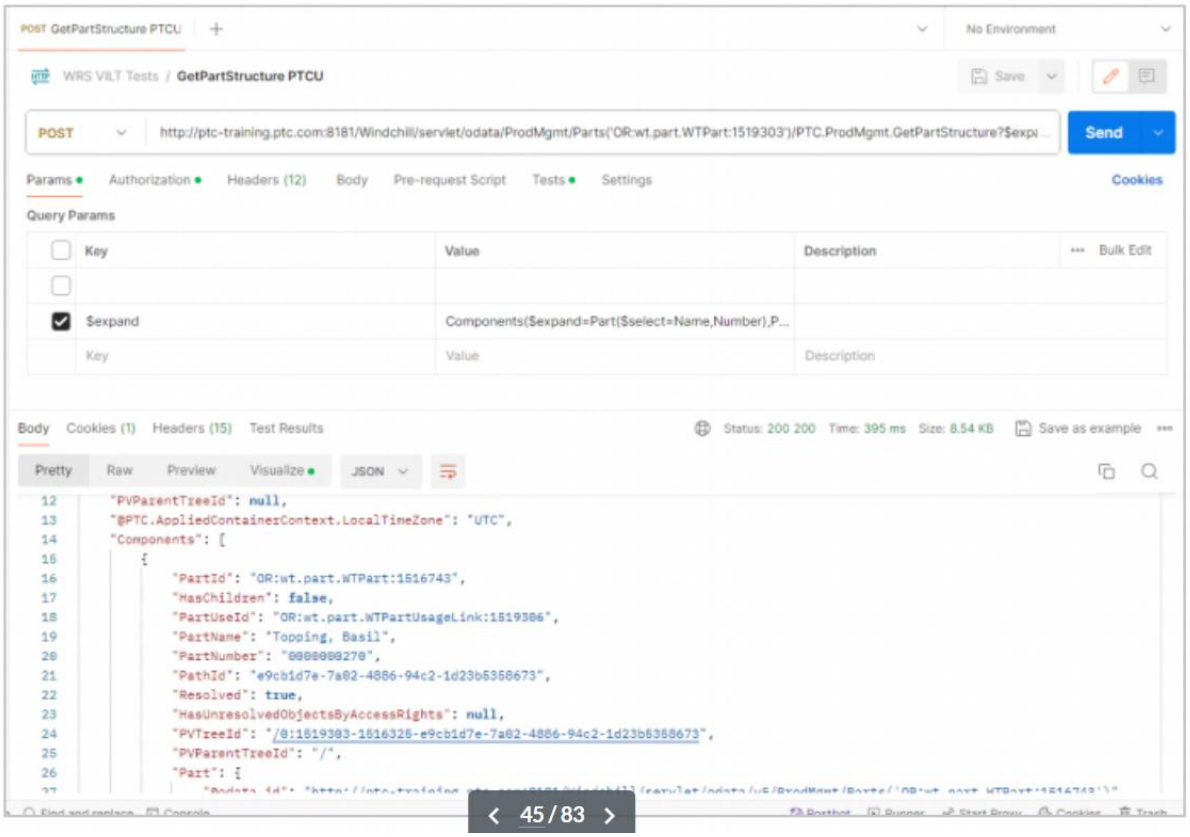
EXERCISE 7 RETRIEVE THE INGREDIENTS LIST

Exercise introduction

Before the pizza engineers prepare the pizza order, they need to get a list of the ingredients: a bill of materials.

Retrieve the part structure using a hierarchy of entities called Components, with path details for Occurrences.





Task 1: Set the URL in Postman

1. In Postman, create a new request tab.
2. Set the method to **POST**.
3. Set the URL as **http://ptc-training.ptc.com:8181/Windchill/servlet/odata/ProdMgmt/Parts('OR:wt.part.WTPart:1519303')/PTC.ProdMgmt.GetPartStructure?$expand=Components($expand=Part($select=Name,Number),PartUse;$levels=max)**.
   Note: You can paste this URL from W:\WCEC-REST-Lab-Files\WCEC-REST-Copy-Text.txt.

Task 2: Set the Headers

1. In the Authorization tab, set the Type to **Basic Auth**.
2. Authorize the following user:
   - Username: **isaha**
   - Password: **ptc**
3. Select the **Headers** tab.
4. Set the following keys and values in their respective columns:
   - CSRF_NONCE = [the nonce value you copied and stored earlier]
   - Content-Type = application/json

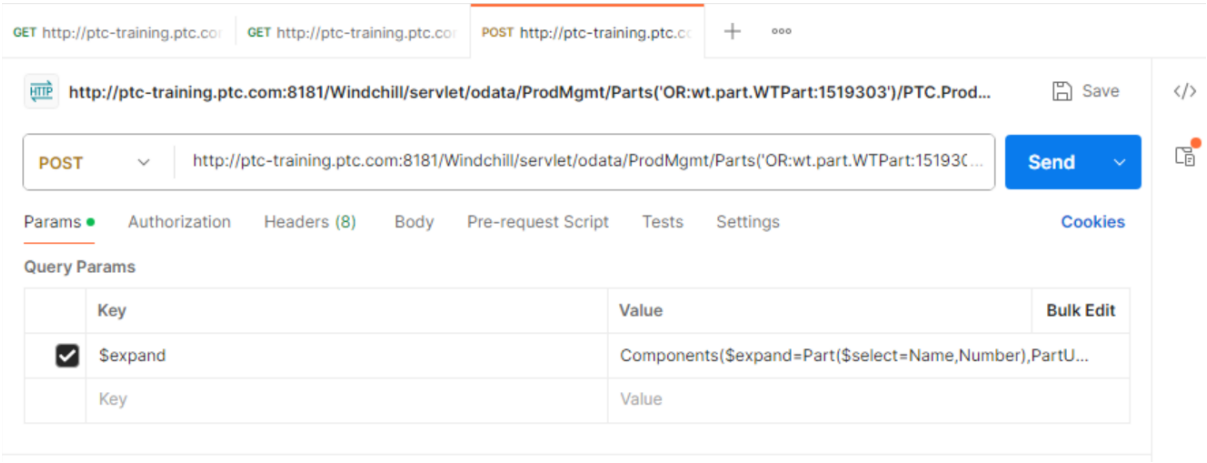
Task 3: Execute the Request

1. Send the request.
2. Verify the return status is successful.
3. Examine the response body. Locate the Components hierarchy and notice the child parts and properties.
4. Do not close the request tab in Postman.

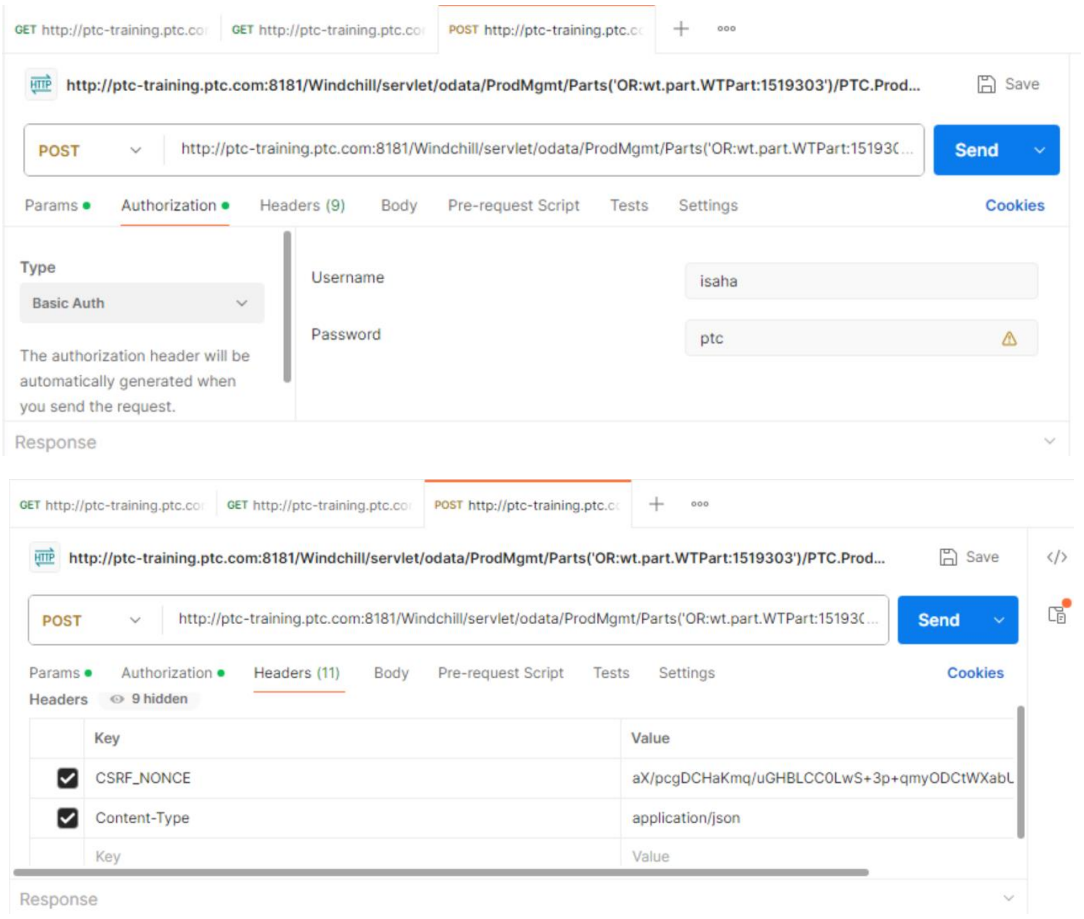
Additional information

- URLs must be typed precisely to avoid errors.
- For Accept and Content-Type headers, if you type the first few characters of the value, a list of responses will appear in a list, and you can select application/json and text/html.
- This is just one example of how you can get a part structure. You could also use a `GetMultiLevelComponentsReport` to return a consolidated flat list of leaf node components and their quantity and units of measure, but this does not list the levels of parts and the part-component relationships within an entire part structure. Another option is `GetBOMWithInlineNavCriteria` to return a BOM that includes working copies of parts and to specify a Plant Filter or Plant Config specification.

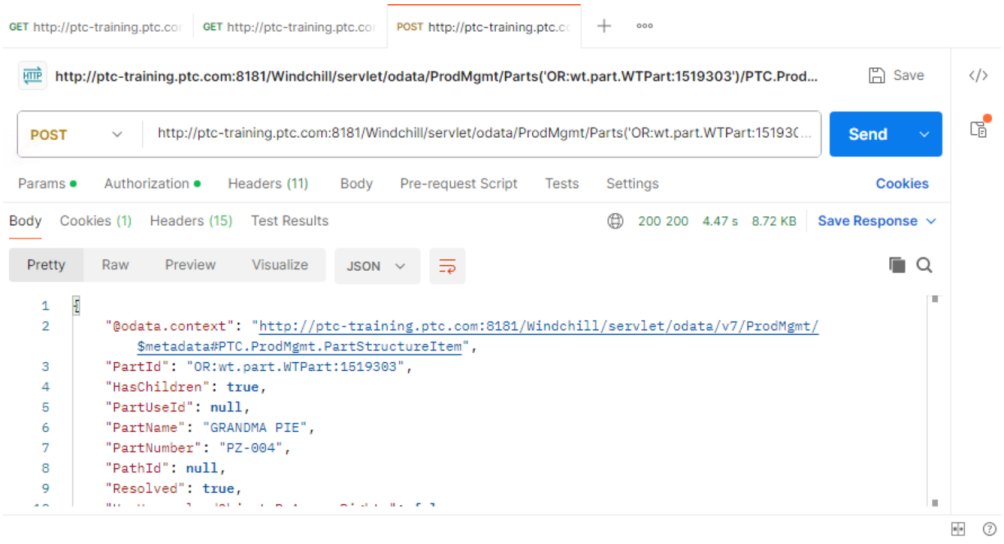
Task 1 Results:



Task 2 Results:



Task 3 Results:



Retrieve a Part Structure: Parse JSON in WRS

RETRIEVE A PART STRUCTURE

PARSE JSON IN WRS

JSON is the preferred format for WRS.

Parsing JSON enables clients to use data strings and perform operations.

You can write JavaScript tests for your Postman API requests to extract values from the JSON.


```
"PartUse": {
                "CreatedOn": "2024-08-21T23:24:17Z",
                "ID": "OR:wt.part.WTPartUsageLink:1519311",
                "LastModified": "2024-12-07T23:24:17Z",
                "FindNumber": null,
                "LineNumber": null,
                "ObjectType": "Part Usage",
                "PartOccurrenceAssignedExpression": "",
                "Quantity": 4.0,
                "ReferenceDesignatorRange": null,
                "TraceCode": {
                    "Value": "0",
                    "Display": "Untraced"
                },
                "Unit": {
                    "Value": "ea",
                    "Display": "each"
                }
```


JSON is the preferred format when using Windchill REST Services. These key/value pairs are outlined in double quotes and separated by colons and contained in curly brackets. The data parsed from a JSON API is in the form of objects that need to be converted into their respective data formats as acceptable by the system. When you parse JSON, you convert a string containing a JSON document into a structured data object on which you can perform operations: sorting, counting, or mapping different attributes like PartUse, Units, and Source. You can parse JSON to perform the navigation between persistables in Windchill. You can write test scripts for your Postman API requests in JavaScript to parse and extract some values from the JSON.

Retrieve a Part Structure: Visualization Responses

RETRIEVE A PART STRUCTURE

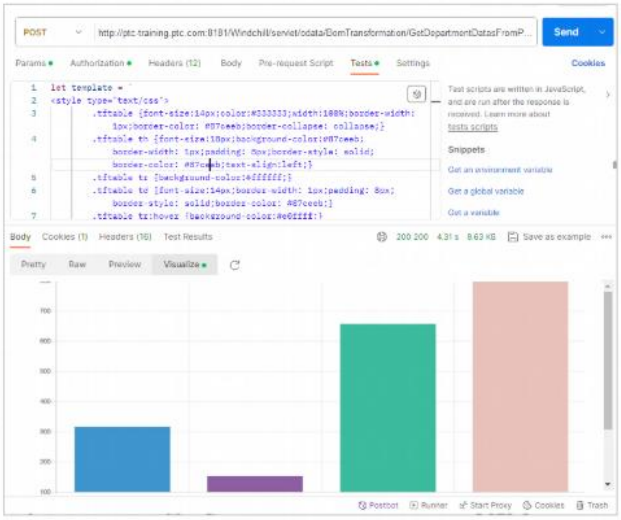
VISUALIZE RESPONSES

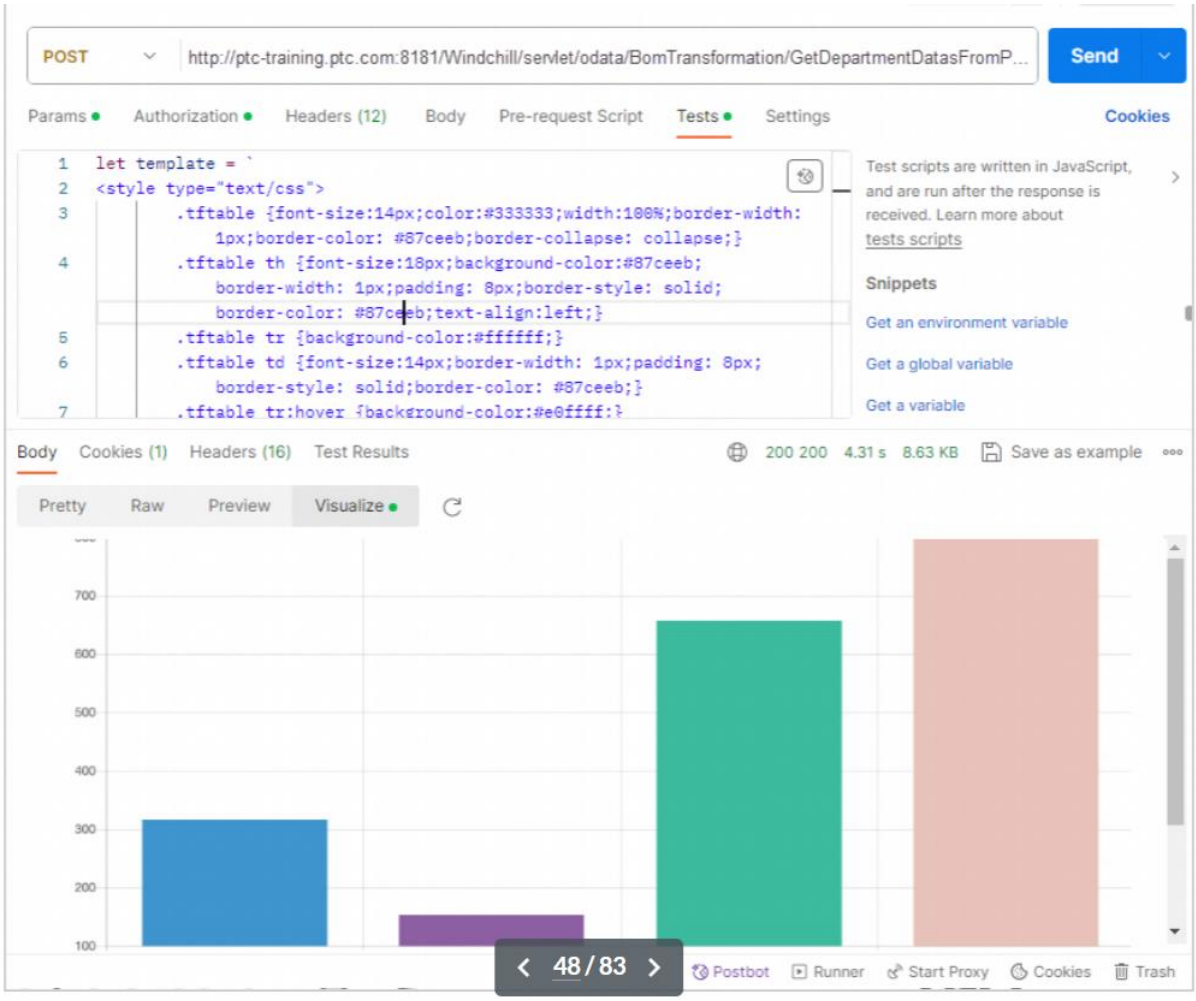
Extract specific response data

Postman Visualizer renders responses in HTML.

It uses the Handlebar template to filter large JSON and XML responses in a human readable format:

```
{{PartUse.Quantity}}
{{PartUse.Unit.Display}}
```





Extract specific data from request responses in a programmable way using the Postman Visualizer. The Postman API uses the information you pass to `pm.visualizer.set()` to render an HTML page in the Visualize tab. You can visualize responses as tables, charts, graphs, and plain text. You can add CSS and interactions like links, hover elements, and more.

The Visualizer code uses a Handlebar template to filter large JSON and XML responses into a more human-readable format.

The method uses three parameters:
- layout (required) – The first parameter of the Handlebars HTML template string
- data (optional) – Data which you can bind to the template
- options (optional) – Used to control how Handlebars compiles the template

In the next exercise, you will use a template to write different navigation properties to a visual table.

Retrieve a Part Structure: Ex-8 Test and Visualize Responses

RETRIEVE A PART STRUCTURE

EXERCISE 8 TEST AND VISUALIZE RESPONSES

Exercise introduction

Refine the data from the Grandma Pie part structure in a table by adding a Visualizer test in Postman.

Task 1: Add a Test in Postman

1. In the GetPartStructure request, select the **Tests** tab.
2. Paste the following code in line 1 of the script field:
   Note: You can paste this code from W:\WCEC-REST-Lab-Files\WCEC-REST-Copy-Text.txt.

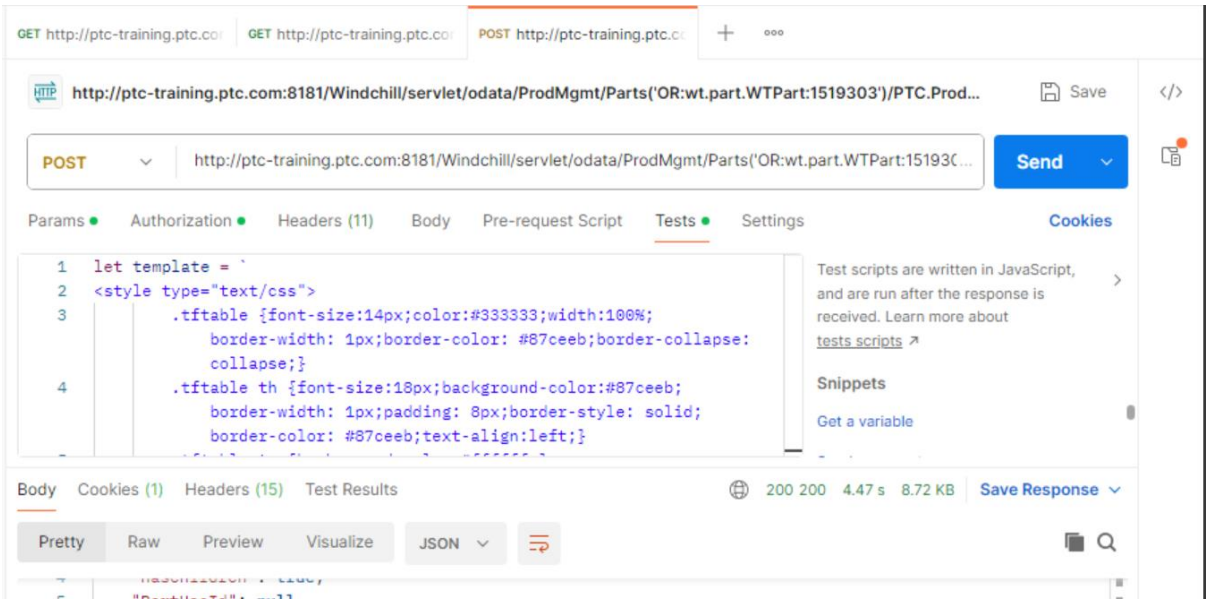
```
let template = `
<style type="text/css">
        .tftable {font-size:14px;color:#333333;width:100%;border-width: 1px;border-
color: #87ceeb;border-collapse: collapse;}
        .tftable th {font-size:18px;background-color:#87ceeb;border-width: 1px;padding:
8px;border-style: solid;border-color: #87ceeb;text-align:left;}
        .tftable tr {background-color:#ffffff;}
        .tftable td {font-size:14px;border-width: 1px;padding: 8px;border-style:
solid;border-color: #87ceeb;}
        .tftable tr:hover {background-color:#e0ffff;}
    </style>

    <h2>Pizza Order 004: Grandma Pie - {{response.PartId}}</h2>

    <table class="tftable" border="1">
        <tr>
            <th>Ingredients Needed</th>
            <th>Item Number</th>
            <th>Quantity</th>
```

```
                <th>Units</th>
            </tr>

            {{#each response.Components}}
                <tr>
                    <td>{{PartName}}</td>
                    <td>{{PartNumber}}</td>
                    <td>{{PartUse.Quantity}}</td>
                    <td>{{PartUse.Unit.Display}}</td>
                </tr>
            {{/each}}
        </table>
    `;

    // Set visualizer
    pm.visualizer.set(template, {
        // Pass the response body parsed as JSON as a simple table
        response: pm.response.json()
    });
```

Task 2: Validate Response

1. Send the request.
   Note: Post-response tests must be executed again.
2. In the Response pane, click **Visualize**.
3. Notice the table layout of the Grandma Pie product, including the Name, Number, Quantity in ounces, and the Unit Value.

Task 1 Result:



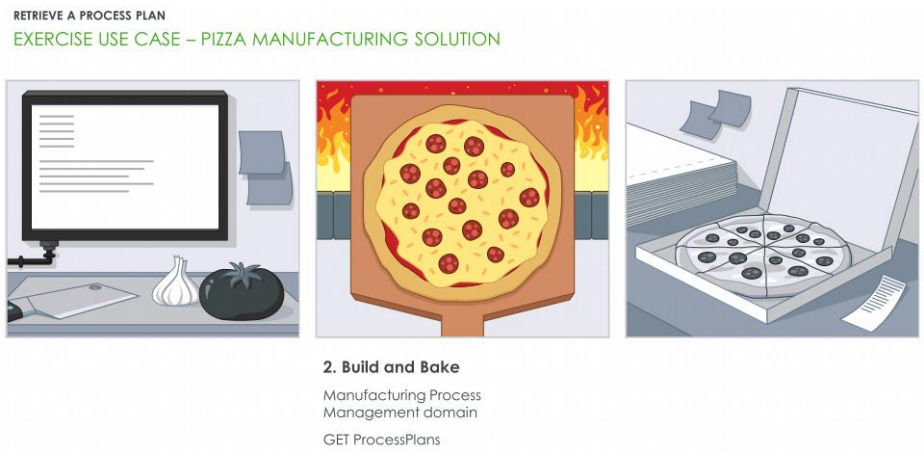
Task 2 Results:

Pizza Order 004: Grandma Pie - OR:wt.part.WTPart:1519303

Ingredients Needed	Item Number	Quantity	Units
Topping, Basil	0000000270	4	as needed
Topping, Hand Cut Pepperoni	0000000281	1	each
Dough Ball, Artisan, 18 oz	0000000263	1	each
Dough Ball, Artisan, 18 oz	0000000263	1	each
Topping, Mozzarella Cheese	0000000267	16	as needed
Topping, Garlic	0000000276	4	as needed
Sauce, Organic San Marzano	0000000283	32	as needed
Sesame Seeds	0000000279	7	as needed
Butter, Sea Salt	0000000274	1	as needed

Retrieve a Process Plan: Highlights

Retrieve a Process Plan: Ex Use Case – Pizza Manufacturing Solution



Retrieve a Process Plan: Manufacturing Operation Usage Link

RETRIEVE A PROCESS PLAN

MANUFACTURING OPERATION USAGE LINK

Windchill Entity

- Process Plan
- Standard Operation

OData Entity

- Process Plan

Navigation Properties

- Name
- Number
- MPMOperationUsageLinks
- OperationHolder

Manufacturing Process Management is the process of defining and managing the manufacturing processes, which are used to make parts, assemble final products, and perform inspections. The domain provides OData entities that represent business objects like a process plan, operation, sequence, bill of process (BOP), resource, and control characteristic (CC). The domain is available only if you install Windchill REST Services and Windchill MPMLink.

The GET ProcessPlans function returns the operations from the process plan structure. An OperationHolder represents the process plan, sequence, operation, or standard procedure. In the next exercise, you will fetch the OperationHolders with multiple inline criteria and order them by their number. This aligns with the View filters in the Structure tab of a process plan.

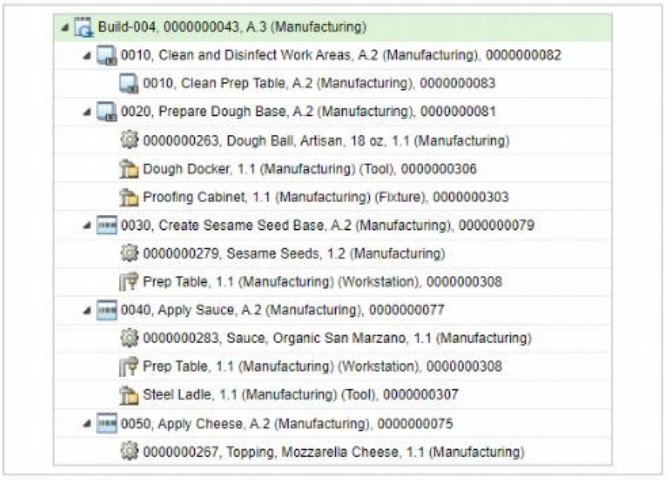
Retrieve a Process Plan: Ex-9 Retrieve the Process Plan for the Grandma Pie

RETRIEVE A PROCESS PLAN

EXERCISE 9 RETRIEVE THE PROCESS PLAN FOR THE GRANDMA PIE

Exercise introduction

As Irene Saha, make a request to fetch the process plan to prepare the Grandma Pie part.



Task 1: Set the URL for the Process Plan in Postman

1. In Postman, create a new Untitled Request.
2. Set the HTTP method to **GET**.
3. Set the URL to **http://ptc-training.ptc.com:8181/Windchill/servlet/odata/MfgProcMgmt/ProcessPlans('OR:com.ptc.windchill.mpml.processplan.MPMProcessPlan:1519760')/OperationUsageLinks?$expand=Operation&$orderby=OperationLabel**.
   Note: You can paste this URL from W:\WCEC-REST-Lab-Files\WCEC-REST-Copy-Text.txt.

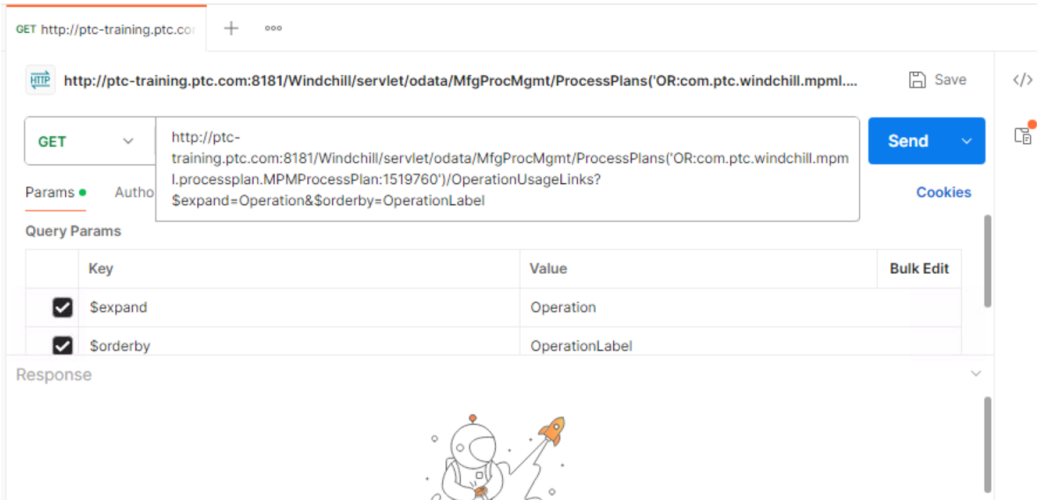
Task 2: Set the Headers

1. In the Authorization tab, set the Type to **Basic Auth**.
2. Authorize the following user:
   - Username: **isaha**
   - Password: **ptc**
3. Send the request.
4. Verify the return status code.
5. Examine the message in the Body pane.
6. Do not close the request tab.

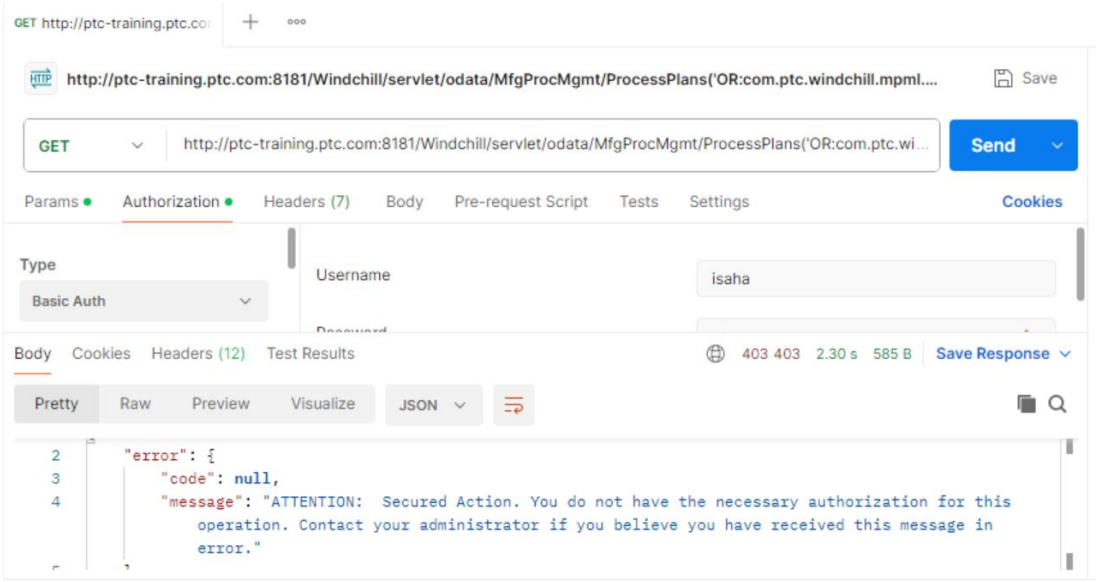
Additional information

The response should have a secured action error. If you received any status code other than 403, review the method, URL, headers, and authorization.

Task 1 Results:



Task 2 Results:



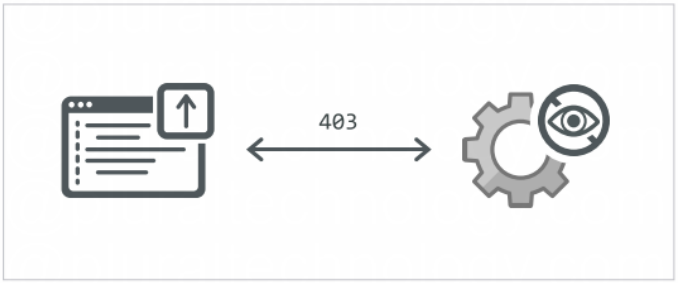
Retrieve a Process Plan: Access Control in WRS

RETRIEVE A PROCESS PLAN

ACCESS CONTROL IN WRS

Windchill REST Services follows the same access permissions governed by the Policy Administrator utility and attribute-based rules.

If an API request is made and returns a generic authorization error instead of the expected response, an administrator should check the access control rules for the user(s).



In the previous exercise, the request returned the error message "ATTENTION: Secured Action. You do not have the necessary authorization for this operation. Contact your administrator if you believe you have received this message in error."

Just as with working in the Windchill user interface, permission to read or modify objects applies in Windchill REST Services. The user performing the actions from a client must have the right permissions on the versioned object. For example, not all users can fetch Groups, patch a CAD Document's common attributes, or create a new Supplier organization. For some combined actions, users may need multiple permissions.

One exception is saved searches. Saved searches are not limited by access control. When you make a SavedQueries request from the Saved Searches domain, the response will return a list of all users' saved searches in Windchill.

While you cannot check or modify permissions using Windchill REST Services, you can troubleshoot permissions with the following actions:

- Check the nonce and/or authorization headers on the client.
- Try a different client.
- Switch the domain version as they may have different configurations.
- Consider any REST endpoint customizations which may interfere with access.
- Turn on logging of classes of com.ptc.odata package > Generic Loggers.

Instead of troubleshooting the permissions in Windchill, you will sign in as a different user to access your process plan details.

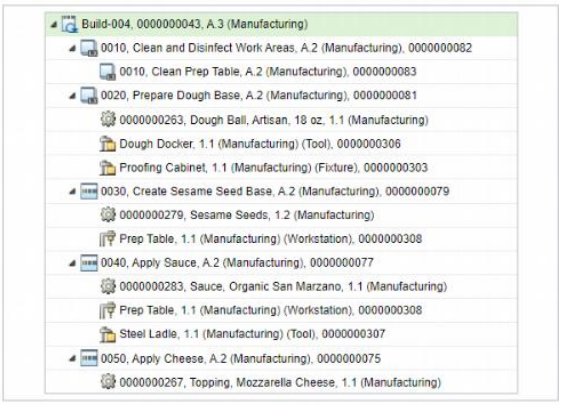
Retrieve a Process Plan: Ex-10 Retrieve the Process Plan By a Different User

RETRIEVE A PROCESS PLAN

EXERCISE 10 RETRIEVE THE PROCESS PLAN BY A DIFFERENT USER

Exercise introduction

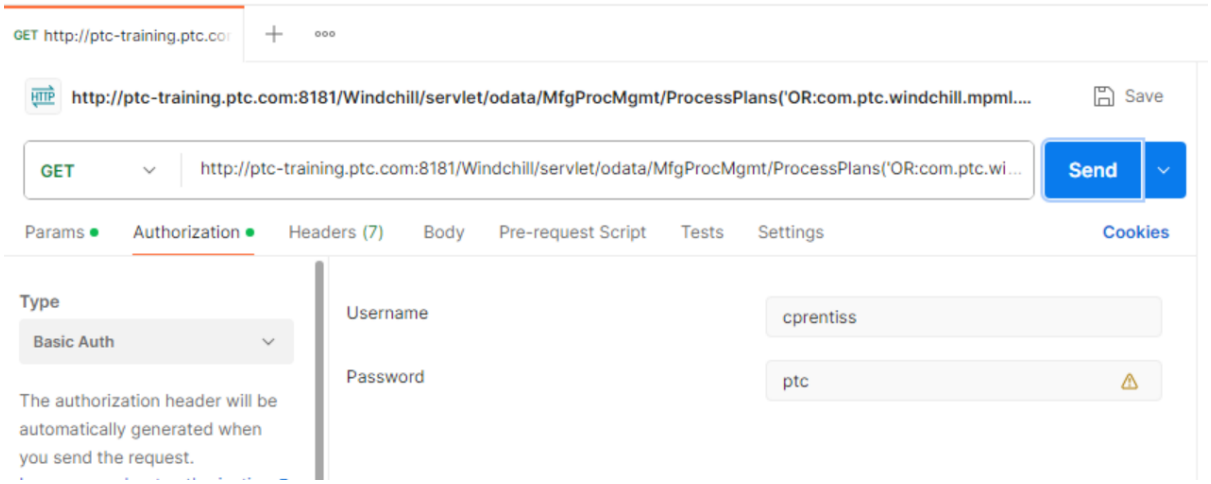
Initiate the request for the Grandma Pie process plan, then examine the response.

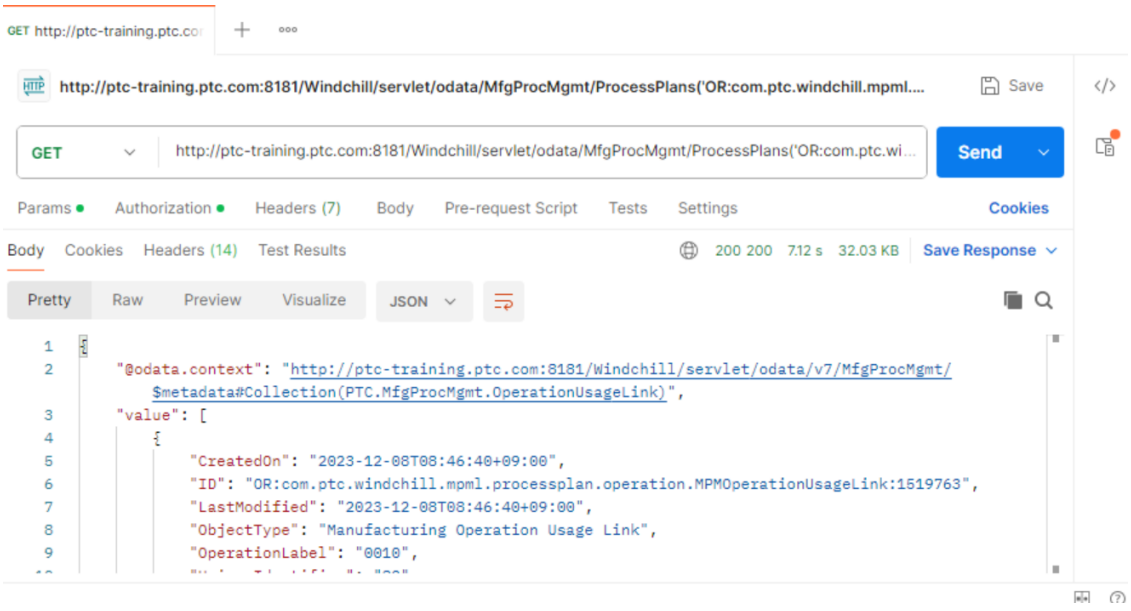


Task 1: Request the Process Plan as an Authorized User

1. Return to the request tab with the secured action error.
   Note: If you closed the tab, create a new untitled request and set the method to GET. You can paste the URL from W:\WCEC-REST-Lab-Files\WCEC-REST-Copy-Text.txt.
2. In the **Authorization** tab, change the username to **cprentiss**.
3. Send the request.
4. Verify the return status code.
5. Examine the response. Note that the operations are listed in order.

Task 1 Results:

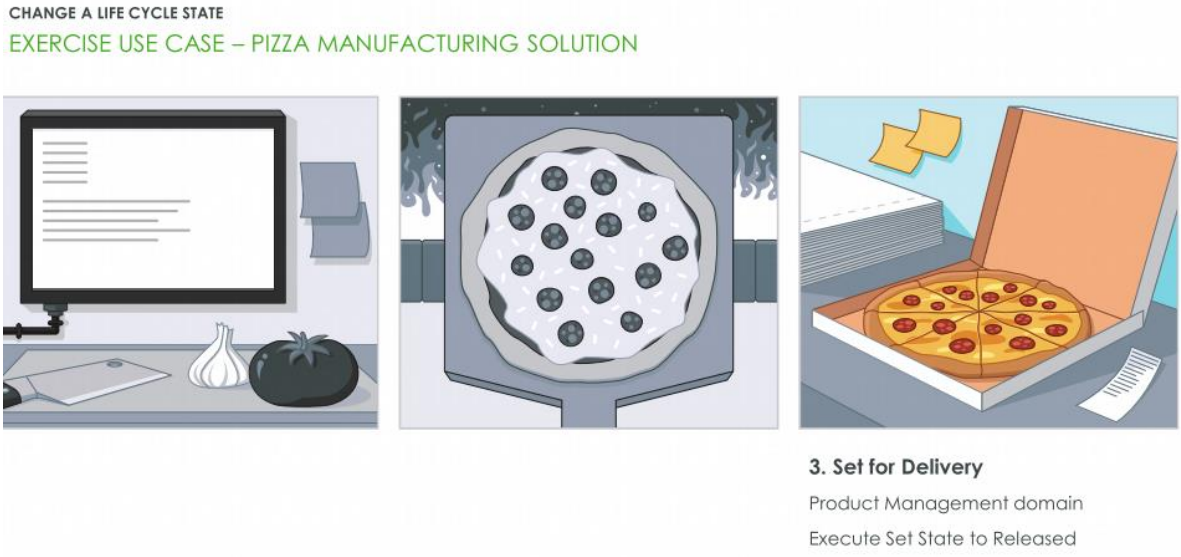




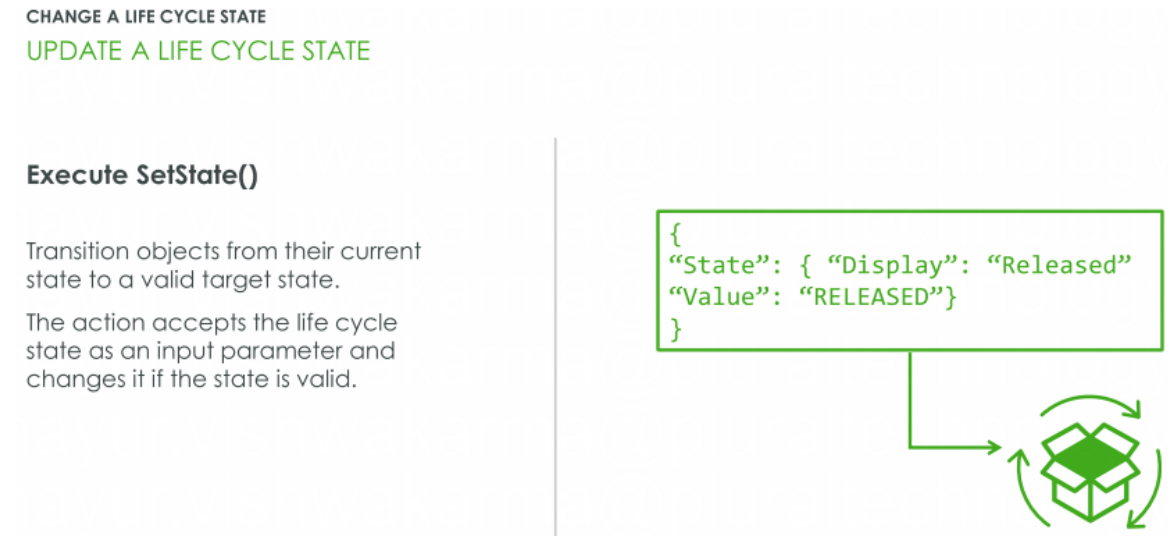
Change a Lifecycle State:

Highlights

Ex Use Case – Pizza Manufacturing Solution



Update a Lifecycle State



An entity can transition through different life cycle states. If an entity is associated with a life cycle template, then the transition states depend on the unique set of life cycle states defined in the template.

To inform the pizza shop manager and delivery personnel that the pizza is ready to go to the customer, you will execute the SetState() action to set a valid life cycle state for an entity. The action accepts the life cycle state as input parameter in the display-value pair format. The valid value for the life cycle state is of type PTC.EnumType.

This request requires a body, and the lifecycle state must already exist in Windchill.

In this next exercise, you will execute a state change on an object.

Ex-11 Set for Delivery

CHANGE A LIFE CYCLE STATE

EXERCISE 11 SET FOR DELIVERY

Exercise introduction

Make a request in the Product Management domain to update the life cycle state of an object.

Task 1: Observe the Life Cycle States of the Grandma Pie

1. Launch a browser and click the **Windchill** bookmark.
2. Click **Sign In**.
3. Sign in to Windchill.
   - Username: **cprentiss**
   - Password: **ptc**
4. Navigate to the Information page for the PZ-004, GRANDMA PIE part.
5. Observe the current and available states in the life cycle.
6. Do not close the browser.

Task 2: Fetch a Nonce Value for the User (Optional)

1. In Postman, click the **+** icon to create an untitled request or go to the tab with the GET request from a previous exercise.
2. Set the method to **GET**.
3. Set the URL to **http://ptc-training.ptc.com:8181/Windchill/servlet/odata/PTC/GetCSRFToken()**.
   Note: You can also paste this URL from W:\WCEC-REST-Lab-Files\WCEC-REST-Copy-Text.txt.
4. In the Authorization tab, set the Type to **Basic Auth**.
5. Type **cprentiss/ptc** to authorize the user.
6. Send the request.
7. Verify the return status is 200.
8. Examine the response body.
9. Copy the Nonce Value that appears between the quotation marks.

Task 3: Set the URL in Postman

1. Create a new request.

2. Set the method to **POST**.
3. Set the URL to **http://ptc-training.ptc.com:8181//Windchill/servlet/odata/ProdMgmt/Parts('OR:wt.part.WTPart:1519303')/PTC.ProdMgmt.SetState**.
   Note: You can also copy and paste this URL from W:\WCEC-REST-Lab-Files\WCEC-REST-Copy-Text.txt.

Task 4: Set the Headers

1. In the Authorization tab, set the Type to **Basic Auth**.
2. Type **cprentiss/ptc** to authorize the user.
3. Select the **Headers** tab.
4. Set the following keys and values:
   - CSRF_NONCE = [the nonce value you copied and stored earlier]
   - Content-Type = application/json

Task 5: Set the Request Body

1. Select the **Body** tab.
2. Select the **raw** radio button.
3. Type in the code:

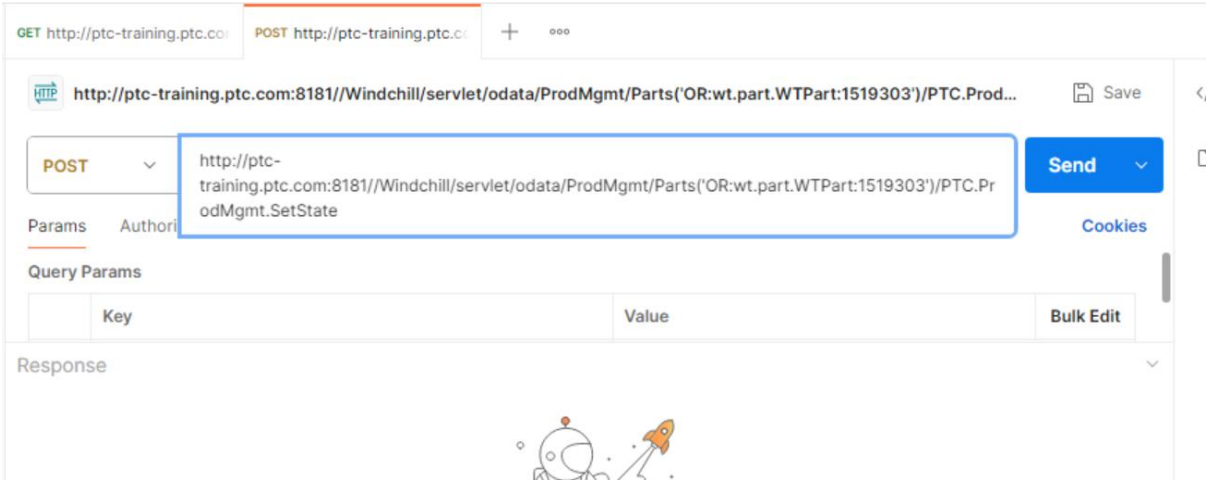
```
{
"State": { "Display": "Released" ,"Value": "RELEASED" }
}
```

   Note: You can also paste this body code from W:\WCEC-REST-Lab-Files\WCEC-REST-Copy-Text.txt.
4. Send the request.
5. Verify the return status is 200 OK.

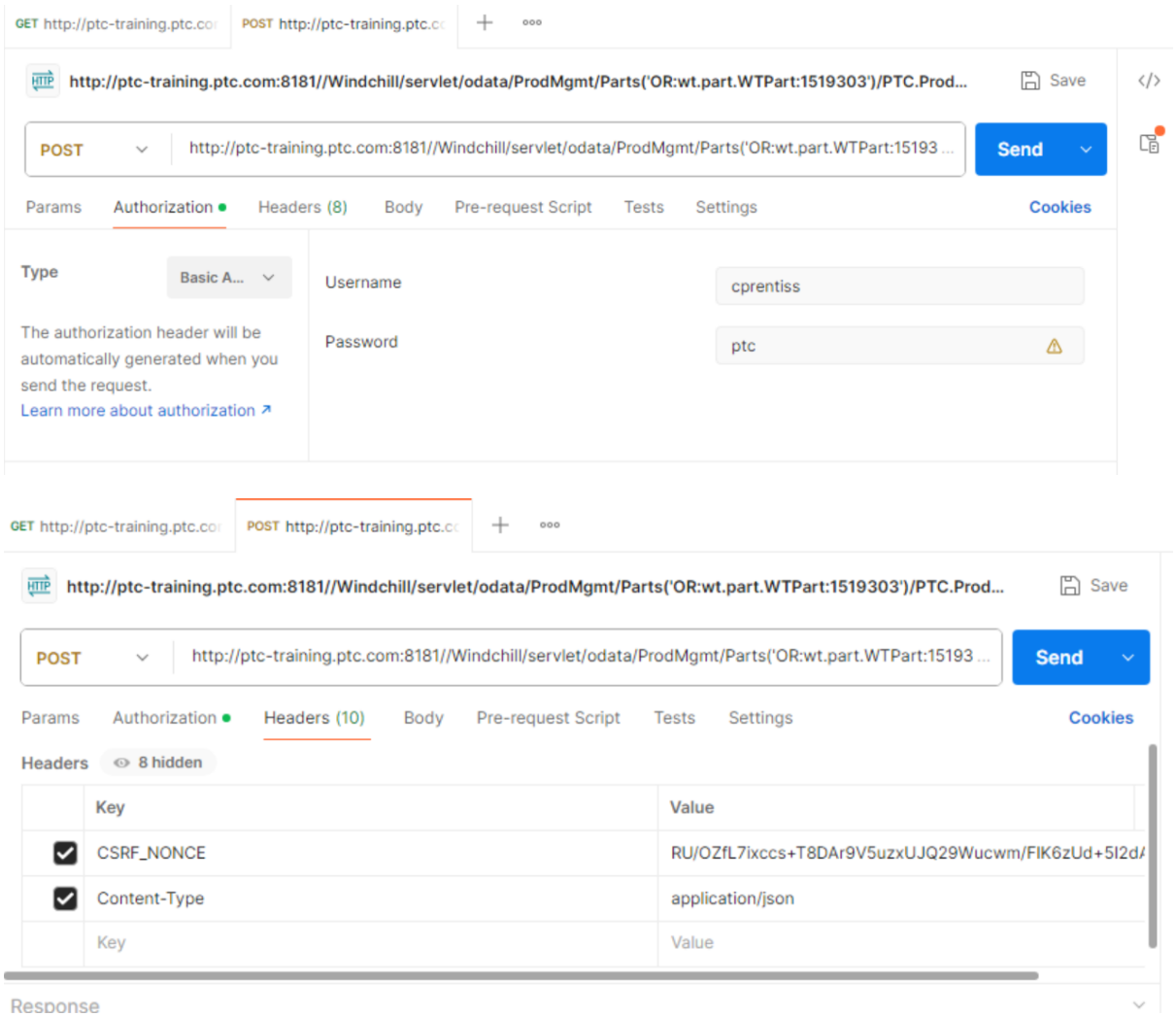
Task 6: Observe the Data Change

1. In the response body, locate the State property and observe the value.
2. Return to the browser.
3. Refresh the Information page for the PZ-004, GRANDMA PIE part and notice the life cycle change.

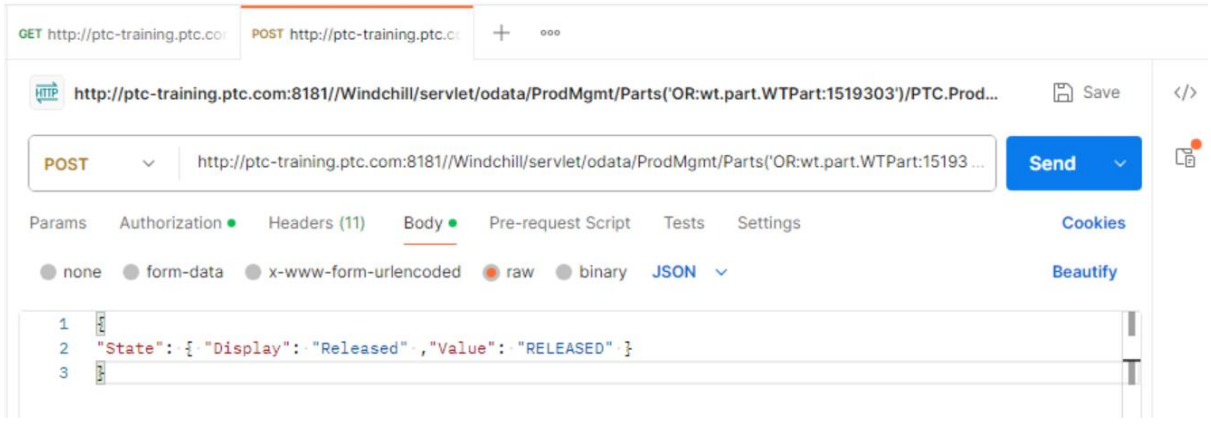
Task 3 Results:



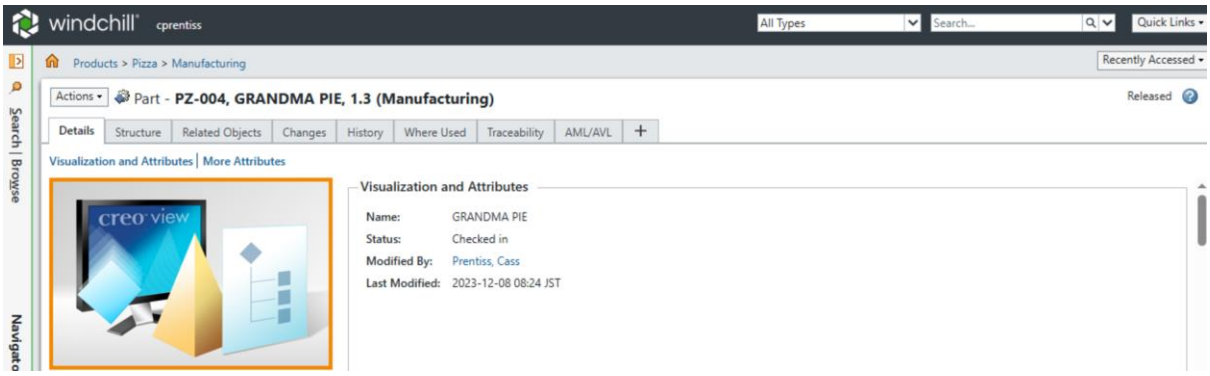
Task 4 Results:



Task 5 Results:



Task 6 Results:



Multi-Object Requests

Overview

Highlights

MULTI-OBJECT REQUESTS

HIGHLIGHTS

Group multiple operations in a single HTTP request. In this section, you will create two new ingredients that your pizza shop has added to the menu.

Batch Processing and Multi Object Processing

MULTI-OBJECT REQUESTS

BATCH PROCESSING AND MULTI-OBJECT PROCESSING

Reduce Network Overhead

Aggregate related operations to consume resources.

When creating or updating multiple objects at the same time, use the multi-object endpoints rather than $batch processing.

While each WRS URL uniquely identifies a resource, you can aggregate operations and dramatically reduce the number of round trips a client needs to perform to consume resources through batch processing or multi-object processing.

Use batch requests:

- In high latency environments for reducing network overhead while sending multiple requests.
- When multiple different operations are related and should be accomplished in a single transaction (OData changeset) and while using the `multipart/mixed,boundary=batch` Content Type. For example:
  1. Create a new document.
  2. Check out the document.
  3. Upload content to the checked-out document.
  4. Check in the working copy of the document.

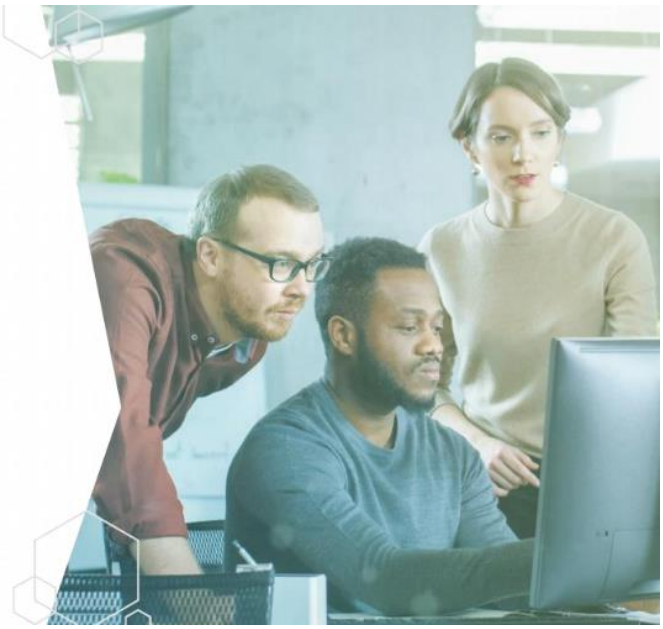
While creating or updating multiple objects simultaneously, use the multi-object endpoints (for example, /CreateParts). Multi-object endpoints are more efficient than a batch request. The operation is similar to creating multiple objects in Windchill.

In the next exercise, you will create two new parts in Windchill using one request. All necessary attribute values are included in the body of the request.

Ex-12: Create Multiple Parts

MULTI-OBJECT REQUESTS

EXERCISE 12 CREATE MULTIPLE PARTS



Exercise introduction

The PTC pizza shop has added two new ingredients to its menu.

Create a request to include these ingredients in the Engineering folder of the Pizza product.

Task 1: Set the URL in Postman

1. In Postman, create a new request.
2. Set the method to **POST**.
3. Set the URL to **http://ptc-training.ptc.com:8181/Windchill/servlet/odata/ProdMgmt/CreateParts**.
   Note: You can also paste this URL from W:\WCEC-REST-Lab-Files\WCEC-REST-Copy-Text.txt.

Task 2: Set the Headers

1. In the Authorization tab, set the Type to **Basic Auth**.
2. Type **cprentiss/ptc** to authorize the user.
3. Select the **Headers** tab.
4. Set the following keys and values:
   - CSRF_NONCE = [the nonce value you copied and stored earlier, or fetch a new nonce]
   - Content-Type = application/json

Task 3: Set the Request Body

1. Click the **Body** tab.
2. Select the **raw** radio button.
3. Paste the request body for this exercise and task from W:\WCEC-REST-Lab-Files\WCEC-REST-Copy-Text.txt into Line 1 of the Body field in Postman.

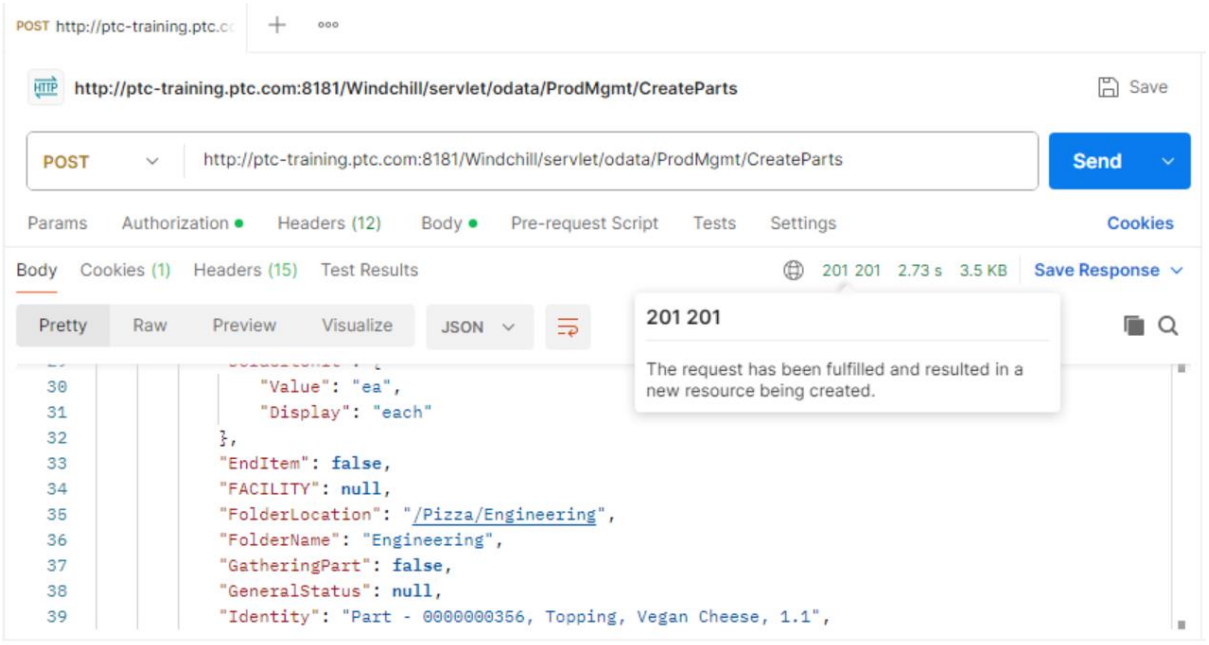
Task 4: Execute the Request

1. Send the request and confirm the status code.
2. Parse the JSON for `FolderLocation` and notice the product/subfolder name.

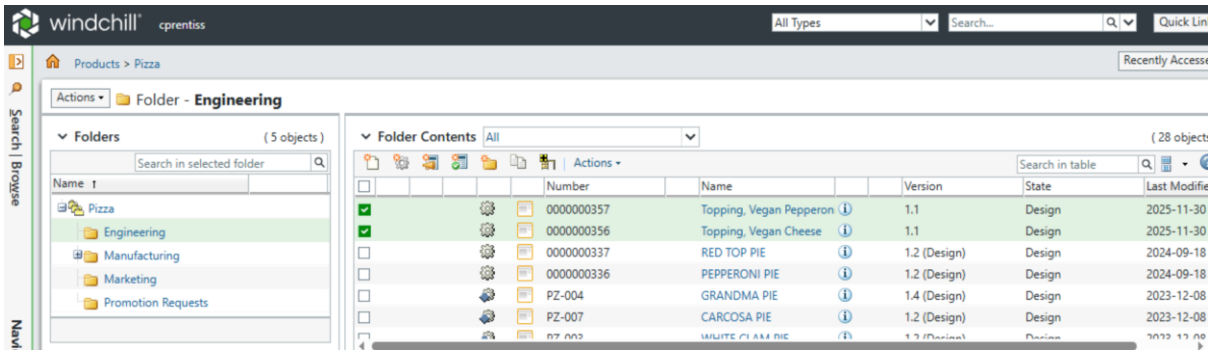
Task 5: View Results in Windchill

1. In a browser, navigate to the Pizza product.
   Note: You should still be signed in to Windchill as cprentiss/ptc.
2. Locate the two new Windchill parts in the Engineering folder.
3. Close the web browser.

Task 4 Results:



Task 5 Results:



Customizations

Overview

WINDCHILL: REST SERVICES

SECTION OVERVIEW

1 ANATOMY OF WRS

2 ACCESS WINDCHILL OBJECTS

3 DESIGN CHALLENGE

4 MULTI-OBJECT REQUESTS

5 CUSTOMIZATIONS

6 AUTHENTICATE END USERS

Highlights

CUSTOMIZATIONS

HIGHLIGHTS

Windchill REST Services lets you extend client functionality and create custom domains.

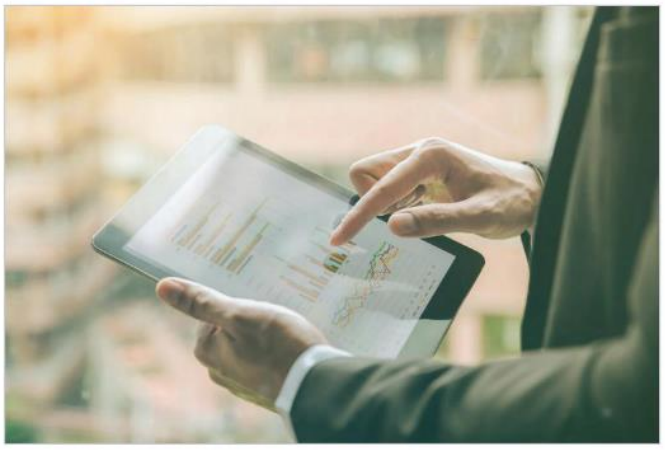
Extend WRS Domains

CUSTOMIZATIONS

EXTEND WRS DOMAINS

Windchill REST Services supports these extensions:

- Add type extensions of Windchill types to PTC Domains
- Add custom properties to entities in PTC Domains
- Add custom navigation between entities in PTC Domains
- Add new functions to PTC Domains
- Add new actions to PTC Domains



A PTC domain can be extended to add an OData entity that corresponds to a custom soft type created in Windchill. Entity types, structural properties, and navigation properties present in PTC domains cannot be modified. Customizers often create a custom soft type extension in Windchill to add a new behavior. To extend any PTC domain, mirror the domain folder structure from the PTC configuration path in the custom configuration path. Only the folder structure must be mirrored. Copies of JSON and JS files are not created, so you must create the required files. All domain configurations provided by PTC are maintained in $WT_HOME/codebase/rest/ptc/domain. You can extend PTC domains or create new domains by putting their configuration in $WT_HOME/codebase/rest/custom/domain/<domain name>/<version>/entity/.

For example, you can create a subclass of WTPart: a soft type `com.custom.PurchasePart`. You can also add additional string attribute called `SupplierName` on `PurchasePart`. To manage this soft type in the Product Management domain, you must first mirror the ProdMgmt domain folder structure in the custom configuration path. Follow the <domain name> > <version> > entity folder structure. Then, create the PurchasePart.json file:

```
{
"name": "PurchasePart",
"type": “wcType",
"wcType": “com.custom.PurchasePart",
"collectionName": “PurchaseParts",
"includeInServiceDocument": "false",
"parent": {
"name": "Part"
},
"attributes": [
{
"name": "SupplierName",
"internalName": "SupplierName",
"type": "String"
}
]
```

Lastly, save the JSON file to $WT_HOME/codebase/rest/custom/domain/ProdMgmt/<version>/entity.

Create Custom Domains

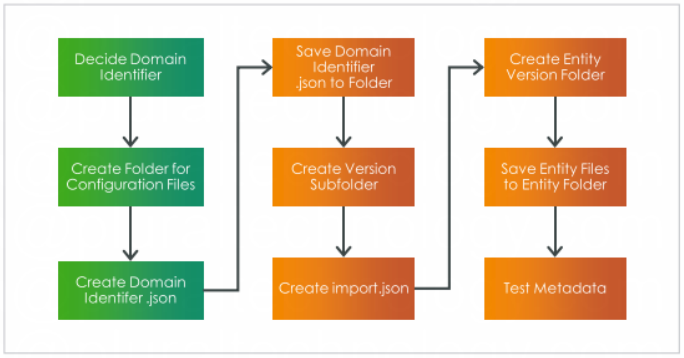
CUSTOMIZATIONS

CREATE CUSTOM DOMAINS

You can create new domains in any functional area.

Create a custom domain folder structure:

```
<Windchill>/codebase/rest/custom/domain
    <Domain_Identifier>
        <domain_version>
            <entity folder>
            complexType
            Entity
            import.json
            import.js
        <Domain JSON File>
```



In most cases, PTC Domains in Windchill REST Services can achieve the data requests needed for your application. However, you can create new WRS domains in any functional area, even in areas not available in the PTC Domains. To create a new domain, you must create and add new configuration files to a custom configuration path. Information can only be read from Windchill using the domain.

If a new domain is the objective, Windchill types must be exposed as OData entities. Any navigational relationship between entities must also be exposed. Each domain has its own folder and corresponding JSON configuration file describing the domain and its namespace. By convention, PTC domains' namespaces are PTC.<domainName>. From the codebase, open ProdMgmt.json in Notepad++, you'll notice that it has a name, ID, description, namespace, container name, and its default version.

The process looks like the following:

1. Identify a domain identifier.
2. In <Windchill>/codebase/rest/custom/domain, create a new folder for your configuration files. For example, a custom domain called CustomDomainName would include the path <Windchill>/codebase/rest/custom/domain/CustomDomainName/.
3. Create the domain identifier .json and save the CustomDomainName.json configuration file to the <Windchill>/codebase/rest/custom/domain/. This should have values for domain metadata attributes.
4. Create the version folder: <Windchill>/codebase/rest/custom/domain/CustomDomainName/v1.
5. Create the import.json file and save it to the version folder.
6. Create the entity version folder: <Windchill>/codebase/rest/custom/domain/CustomDomainName/v1/entity.
7. Create the entity files to expose read-only information on entities like ChangeableItems and ProblemReports. This file helps navigate from source to target entities, so specify a fully qualified name for the entity.
8. Save the entity files to the entity folder.
9. Test the EDM with the URL, such as https://<Windchill server>/Windchill/servlet/odata/CustomDomainName/$metadata.

Custom domains will appear in the list of available domains in the API Catalog.

Authenticated End Users

Overview

WINDCHILL: REST SERVICES

SECTION OVERVIEW

1 ANATOMY OF WRS

2 ACCESS WINDCHILL OBJECTS

3 DESIGN CHALLENGE

4 MULTI-OBJECT REQUESTS

5 CUSTOMIZATIONS

6 AUTHENTICATE END USERS

Highlights

AUTHENTICATE END USERS

HIGHLIGHTS

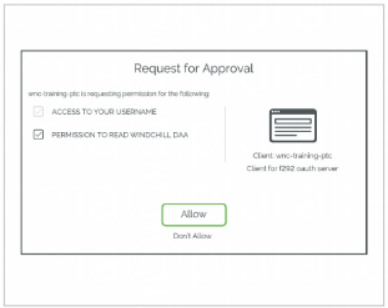
For browser- or mobile-based applications, getting authorization from the user is the first step to using OAuth authentication.

OAUTH2 Flow for User Authentication

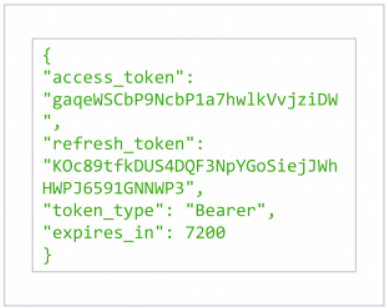
AUTHENTICATE END USERS

OAUTH2 FLOW FOR USER AUTHENTICATION

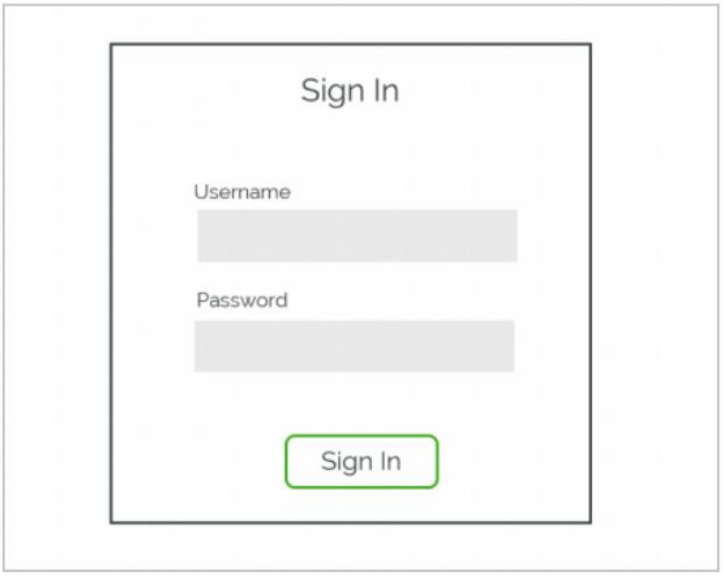
Create Authorization Prompt



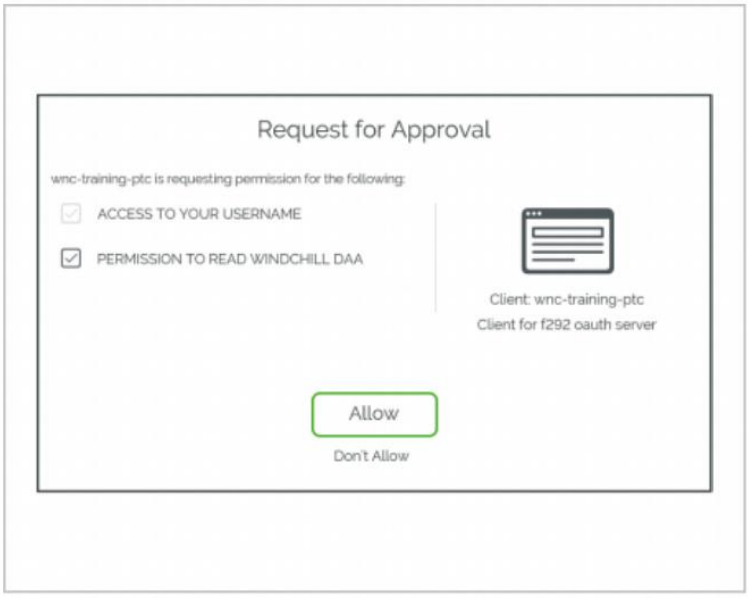
User Approves the Request



Use the Access Token



Create Authorization Prompt



User Approves the Request

```
{
"access_token":
"gaqeWSCbP9NcbP1a7hwlkVvjziDW
",
"refresh_token":
"KOc89tfkDUS4DQF3NpYGoSiejJWh
HWPJ6591GNNWP3",
"token_type": "Bearer",
"expires_in": 7200
}
```


Use the Access Token

Windchill REST Services does not contain any authorization code or compatibility with SSO. It is up to the client to provide authentication for the request through basic authentication or an authentication token. OAuth2 is a networking protocol and framework providing access to HTTP Services and can be used for system-to-system communication in standalone mode or on behalf of a user. It does not directly handle authentication and operates more as a general framework for authorization, so you will need to build a request to authorize services to a granted client.

Note: Basic authentication is not supported in Windchill+ Select or MedDev systems. A token-based AuthN or AuthZ as per OAuth2 specification is required.

There are five general steps in the authorization flow:

1. Create an authorization prompt as a sign-in link to the user.
2. When the user arrives at the OAuth prompt and clicks Sign In, the service redirects the user to an approval page.
3. When the user clicks Allow, the redirect URL is called with an authorization code.
4. The authorization code is exchanged in a POST request for an access token from the server's token endpoint. The resource server checks the token with the OAuth server to confirm that the client is authorized to consume the resource.
5. The user can then use the access token as an Authorization header and refresh the token as needed. The server can respond with the requested protected resources.

Create an Authorization Prompt

The server exchanges the authorization code for an access token by making a POST request to the authorization server's token endpoint:

POST
https://<authorization_server_api>/tokengrant_type=authorization_code&code=<AUTH_CODE>&redirect_uri=REDIRECT_URI&client_id=CLIENT_ID&client_secret=CLIENT_SECRET

For example:
https://authorization-server.com/auth?response_type=code&client_id=CLIENT_ID&redirect_uri=REDIRECT_URI&scope=photos&state=1234zyx

Where:

- response_type=code – Indicates that your server expects to receive an authorization code.
- client_id – The client ID you received when you first created the application.
- redirect_uri – Indicates the URI to return the user to after authorization is complete.
- scope – One or more scope values indicating which parts of the user's account you wish to access.
- state – A random string generated by your application, which you will verify later.

Approve the Request

You should first compare this state value to ensure it matches the one you started with. You can typically store the state value in a cookie or session and compare it when the user comes back. This helps ensure your redirection endpoint can't be tricked into attempting to exchange arbitrary authorization codes. Examples of a redirect_uri might be to redirect right back into a web application that initially requested the sign-in, or it might redirect into a native application on the client OS using URI schemes.

The server replies with an access token and expiration time:

```
{"access_token":"RsT5OjbzRn430zqMLgV3Ia",
"expires_in":3600 }
```

In this example, the authorization server is configured so that the client_id and client_secret must not be sent as part of the query string. In this case, it uses Basic Auth using the client_id as the user and client_secret as the password:

POST <https://pun-w13-utf038.ptcnet.ptc.com:9031/as/token.oauth2?grant_type=authorization_code&code=EXTtWU5ydKl1j1m4kLwdh_aGDjx-3DVmr444eu_k&redirect_uri=https://app.getpostman.com/oauth2/callback>

The server replies with:

```
{
"access_token": "gaqeWSCbP9NcbP1a7hwlkVvjziDW",
"refresh_token": "KOc89tfkDUS4DQF3NpYGoSiejJWhHWPJ6591GNNWP3",
"token_type": "Bearer", "expires_in": 7200
}
```

Use the Access Token

When you have the access token you can perform the WRS request and specify the access token as the Bearer Token in the Authorization header:

GET https://i7752.ptcnet.ptc.com:443/Windchill/oauth/servlet/odata/ProdMgmt/Parts Request Headers: Authorization: Bearer gaqeWSCbP9NcbP1a7hwlkVvjziDW

Request Header:
Authorization: Bearer gaqeWSCbP9NcbP1a7hwlkVvjziDW

Access tokens have a set lifespan. To save the user the effort of signing in again, use the refresh_token returned when fetching the original access token and use the Basic Auth using the client_id as the user and client_secret as the password:

POST https://<authorization_server_api>/token?grant_type=refresh_token&refresh_token=<REFRESH_TOKEN>&client_id=CLIENT_ID&client_secret=CLIENT_SECRET

The server replies with:

```
{ "access_token": "gaqeWSCbP9NcbP1a7hwlkVvjziDW", "refresh_token":
"KOc89tfkDUS4DQF3NpYGoSiejJWhHWPJ6591GNNWP3", "token_type": "Bearer", "expires_in": 7200
}
```

You can then use the new access token for subsequent WRS calls and the refresh token to get subsequent access tokens.

Conclusion

Discussion

CONCLUSION

DISCUSSION

What methods and resources would you use if you wanted to create an interface to display the pizza order status to a customer?

Consider what Windchill resources you may need. What domains would you use? Methods? What data would you need to integrate from a third-party system, such as a material number from SAP?

Course Summary

CONCLUSION

COURSE SUMMARY

REST APIs
HTTP methods
Query options
Windchill REST Services API catalog
Postman
Visualization
JSON key/value pairs
Access control
Multi-object requests
Customization
End-user authentication

You have reached the end of this course. During this course, you:

- Compared Windchill REST Services to other REST APIs.
- Listed the four HTTP methods used by Windchill REST Services.
- Analyzed query options for refining returned Windchill data.
- Exposed the API Catalog and documentation.
- Used two clients to prototype multiple types of REST requests on Windchill data, including reading a part structure with multiple levels, reading a process plan, and changing an object's life cycle state.
- Visualized data in Postman.
- Evaluated access control in REST calls.
- Created new data in Windchill through a REST call.
- Read and parsed JSON key/value pairs.
- Examined the end-user authorization flow.

Course References

REFERENCES AND CITATIONS

COURSE REFERENCES

Open Data Protocol Basic Tutorial

PTC eSupport Windchill REST Services Knowledge Hub

Windchill REST Services tutorials on PTC University Learning Connector