ptc university

Windchill 11 - Server Customization

Authored and published using servigistics arbortext

ptc university

PTC Inc., 140 Kendrick Street, Needham, MA 02494 USA

PRINTING HISTORY

Document No.	**Date**	**Description**
TRN_4626		Initial Printing of: **Windchill 11 - Server Customization**

Printed in the U.S.A

Table of Contents

Windchill 11 - Server Customization

ptc university

Course Description

In this highly technical hands-on course, you will learn to design and implement server customization in Windchill 11.

Course Objectives

- Demonstrate Annotations
- Demonstrate how to create Helpers
- Demonstrate the use of Annotations in Resource Bundles
- Demonstrate how to extend the Windchill Object Model using Annotations instead of Rational Rose

Module 1

Introduction

Module Overview

Welcome to the class! This module is a discussion about course logistics and expectations. Please review them with your instructor.

Objectives

After completing this module, you will be able to:

- Describe classroom logistics.
- Provide an overview of Tools and System Logistics.

Class Introduction

Introduction

Take some time to meet each other and find out more about everyone's backgrounds and history.

Introduce yourselves by stating your:

- Name
- Company or office location
- Title/role
- Experience with PTC products (roles, products used, types of projects)
- Role and experience on recent projects

Introductions

Classroom Behavior

These are not good classroom behaviors

- Sidebar conversations
- Cell phone ringing or taking a call during class
- Questions unrelated to the topic
- Arguments instead of discussions
- Interrupting
- Inappropriate language
- Distracted by a computer (E-mail or Instant Messages)
- Starting late
- Running long

Classroom Behavior

Because we are taking the time to come together as a group for this training, it is important to review some proper classroom etiquette and expectations before getting started.

Classroom Logistics

Classroom Logistics

Be sure that you are familiar with the building and its surroundings.

- Accessing the building

  If you have been assigned a badge, you are responsible for returning it on the last day of class.

- Toilets
- Emergency exits
- Printers and office supplies
- Kitchen, snacks/beverages
- Lunch

Logistics

Tools and System Logistics — VM

VM details

Internal Students VM Template	Partner Students VM Template
OOTB- WC 11.0 M020 - Windows Based with Thingworx 7.1	OOTB- WC 11.0 M020 - Windows Based with Thingworx 7.1

Accessing Cloud Portal(Using PTC LDAP Login)

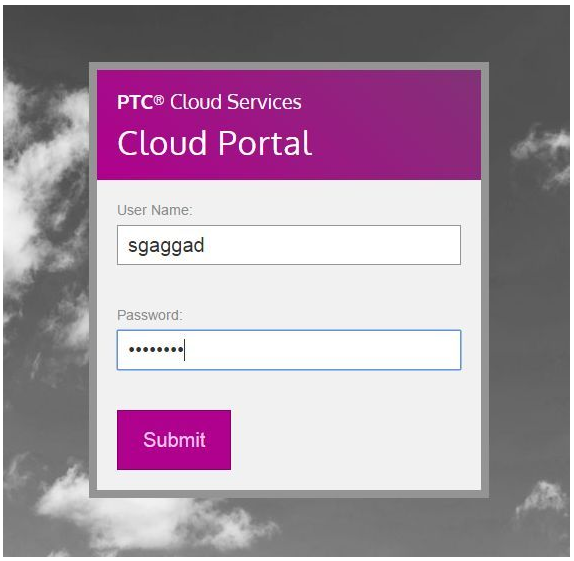
Internal Students	Partner Students
https://portal.ptc.io	https://portal.ptc.io

VM Credentials

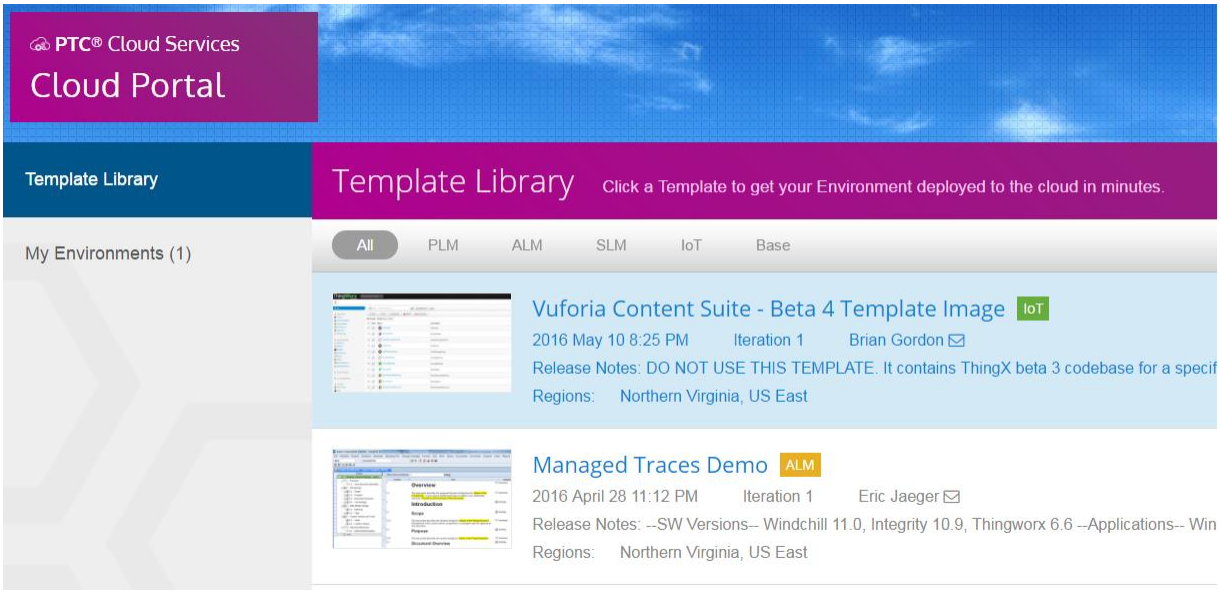
Internal Students Login	Partner Students Login
Administrator / (password should be uniquely generated on deployment of template)	Administrator / (password should be uniquely generated on deployment of template)

Accessing PTC Cloud Portal

- Log in to PTC Cloud Portal using the following link:
  https://portal.ptc.io

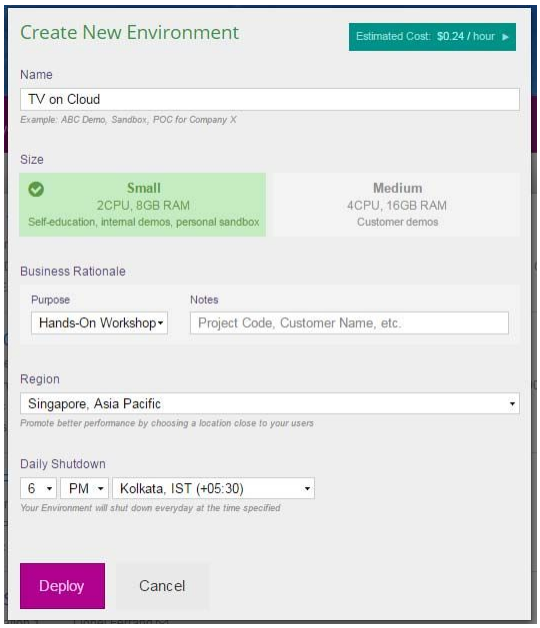


- Login to the portal with your PTC login credentials.
- After Logging in you will be directed to the following page:

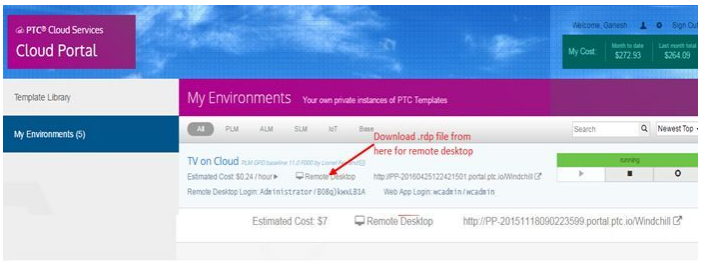


Creating a cloud VM Instance from template

- From the templates library, Select the template **"OOTB- WC 11.0 M020 - Windows Based with Thingworx 7.1"**.
- Click on the template to instantiate. The “Provision a New Instance” dialog appears. Enter a name of the instance and other details as below:



- Click “Deploy”. The instance will appear in “My Environments” list. You can see the SSH Login ID and password.



Exercise 1: Set Up Your Exercise Environment

Objectives

After successfully completing this exercise, you will be able to:

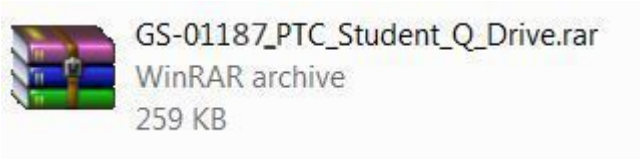
- Prepare all the required exercises in the classroom image.

Scenario

Set up classroom image to have all the required course exercise data.

Task 1: Set Up Your Exercise Environment.

1. If you access classroom image through the browser, you should check the step results mentioned below, which are already configured in your image.
2. Get the Q-Drive zip file from your instructor (or download from PDS course material student folder) and transfer to your classroom image. (Extract this file and include its file path to drive *D:**Q_Drive* on VM image)
3. Share *D:**Q_Drive*
4. Map *D:**Q_Drive* to a new Network Drive: *Q:*\
   Example:



Map a New Network Drive Q

Completion Criteria

1. The students must have prepared all the required exercises in the classroom image.

This completes the exercise.

Module 2

Introduction to Windchill Object Model

Module Overview

Objects in Windchill are derived from one of two basic types: documents and parts. These objects implement many interfaces and are related (linked) to each other. It is the purpose of this lecture to review the definition of these objects in the context of the UML model as a means of introducing the student to both the OOTB behavior of these objects as well as a precursor to the way in which this behavior is defined.

Objectives

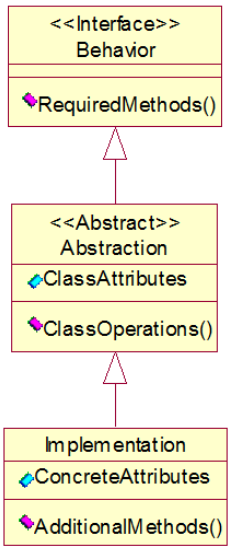
After completing this module, you will be able to:

- List the basic Windchill objects.
- Describe what each object is used for and the relationships they hold to other objects.

Object Oriented Development Principles

Inheritance/Generalization is crucial

- Interface – the highest level of abstraction
  - Represents a desired and common "behavior"
  - Requires a class to implement the inherited functionality
  - Ensures a class is compatible with a corresponding service
- Abstraction – the next level of abstraction
  - Represents desired and common characteristics and behavior
  - Minimizes inconsistency and duplication in implementations
- Concrete – the lowest level of abstraction
  - Represents desired and unique characteristics and behavior



Foundation Package (wt.fc) Interfaces

Foundation Package (wt.fc) Interfaces

ObjectMappable	Objects that implement the ObjectMappable interface can be written into and read from the database. The system generates readExternal and writeExternal methods for these objects.
PersistInfo	Contains information for each object which uniquely identifies each object that is stored in the database.
QueryKey	The QueryKey interface specifies a qualification for a persistable object in the database. It can be used as a primary key, secondary key, or a foreign key.
WTReference	Specifies an indirect addressing mechanism in which it persists one key that, when used in a query, results in finding an object. If none or more than one object is found, this results in an exception.
Link	Specifies the concept of a container of roles. It defines a general relationship between two or more business objects.
BinaryLink	A kind of link; defines the basic relationship between two other business objects.
Persistable	Defines the ability of an object to be written to and read from the database. Any business class that needs to be stored in the database needs to implement this interface.
ObjectReference	Holds a reference to an object by means of the object's identifier.
ObjectIdentifier	Represents a unique identifier assigned to an object.

Enterprise (wt.enterprise) Abstractions

Enterprise (wt.enterprise) Abstractions

Simple	Business objects subject to access control, domain administration, and notification. Not LifeCycleManaged. Not organized into folders.
FolderResident	Business objects that can reside in folders and are subject to administrative rules. Because these objects are accessible in folders, they are visible to users in the Windchill Explorer. Not LifeCycleManaged All FolderResident objects automatically record the principal creator.
IteratedFolderResident	Reside in folders. Visible in Windchill Explorer. Users create new iterations of these objects using checkout/checkin operations. Subject to access control. Automatically record the principal (user) who created them.
Managed	Represents business objects that are AccessControlled, DomainAdministered, Notifiable, Indexable, Ownable, Foldered, and LifeCycleManaged.
CabinetManaged	Represent business objects that are not iterated but are LifeCycleManaged. Do not reside in folders and are not visible in Windchill Explorer. They are associated with cabinets for reasons of access control and local search.
RevisionControlled	Represents objects that are Workable (can be checked in and out), Iterated, and subject to other administration policies. They are managed and changed via a checkin/checkout mechanism. Since they are Versioned, there can have multiple business versions of the Master object, such as revision A and revision B of a single document.

Foundation Package (wt.fc) Classes

Foundation Package (wt.fc) Classes

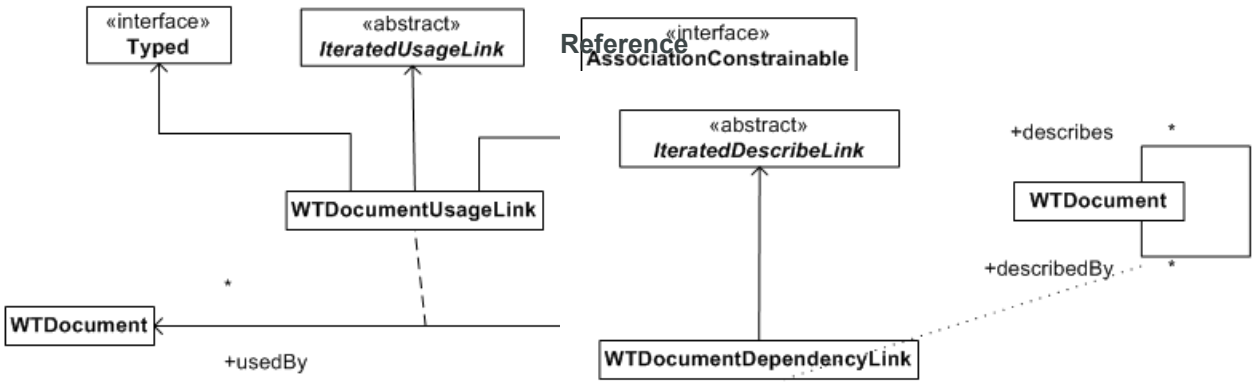
WTObject	Represents the base class for all Windchill business information classes.
ObjectToObjectLink	Represents a concrete binary association between two business objects allowing the developer to create additional attributes can be created and associated between the two business classes.
Item	It represents discrete items that can be associated with an administrative domain and are directly subject to access control.

WTDocument Model

Review WTDocument

- Shown here are just the Structure and Reference portions of the model

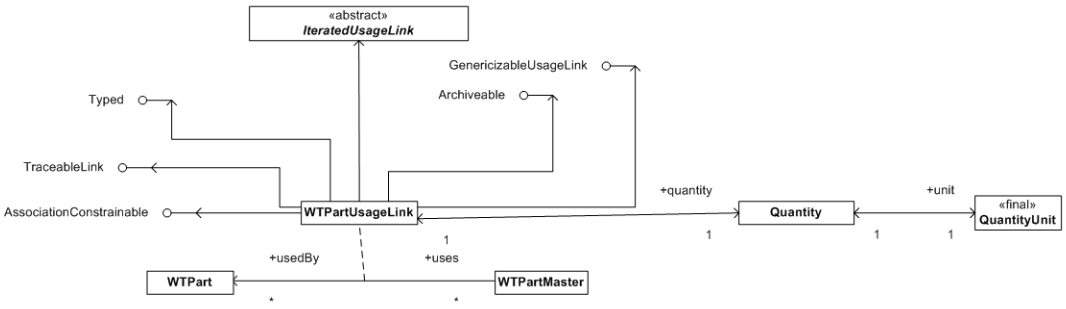
Structure



WTPart Model

Review WTPart

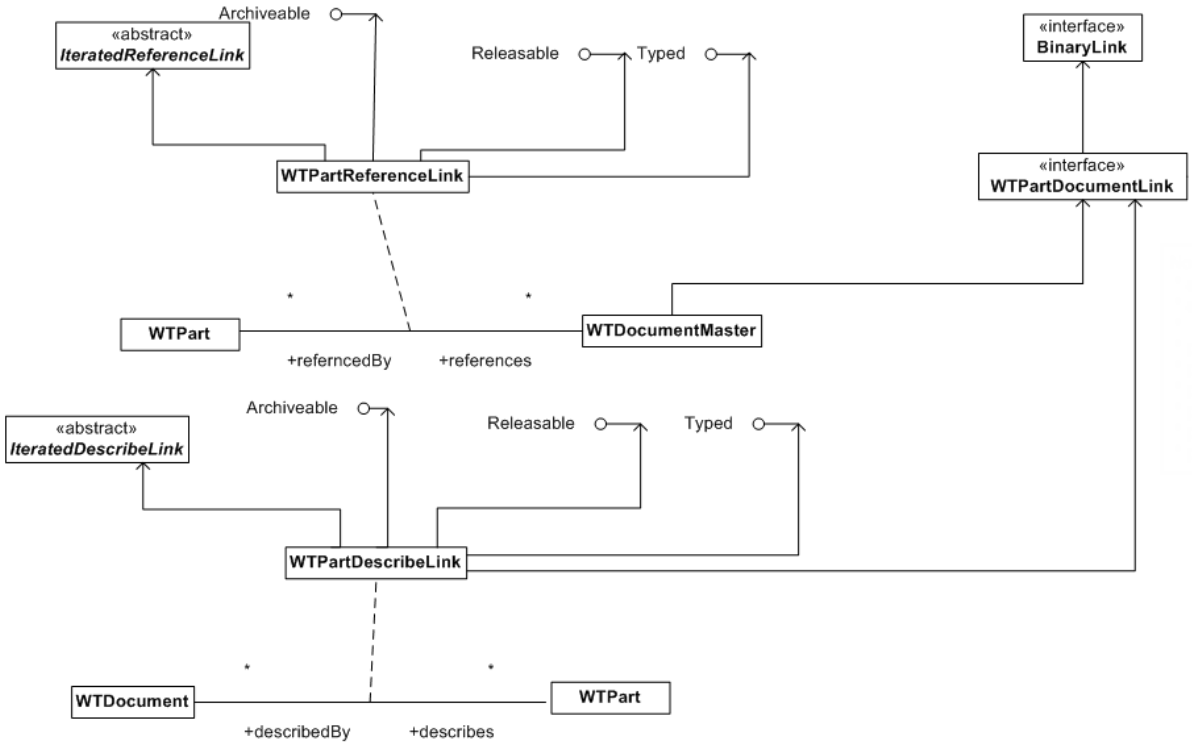
Part Structure



Windchill Part Associations

Review Part Associations of Windchill

WTPart Describe and WTPart Reference



Module 3

Windchill Object Model

Module Overview

The Windchill Object Model is now documented in a Static format.

Objectives

After completing this module, you will be able to:

- Describe how to access documentation on the Windchill Object Model
- Demonstrate sources of Documentation
- Review key components of the model.
- Describe how the Object Model is made available for customization.

Java Code

From Picture Driven to Template Driven Coding

Evolution:

- UML is still required for conceptual understanding
  - Diagram is viewed more as Documentation
  - “Notes” categorize reusable code templates
- Annotation Processing
  - Java code generates more java code.
  - Separation of Custom Code and Generated Code
  - Streamlined custom code
- IDE-friendly Java resource bundles
- RMI via annotation + dynamic proxies

Windchill JavaDoc

JavaDoc is the primary source for documentation

- An example is wt.vc.views.View

1. Hierarchy, superClass, and extends
2. “name” and “sortId” as properties.

wt.vc.views

Class View

java.lang.Object
  wt.fc._WTObject
    wt.fc.WTObject
      wt.vc.views._View
        wt.vc.views.View

1

All Implemented Interfaces:

Externalizable, Serializable, wt.fc._NetFactor, wt.fc._ObjectMappable, wt.fc._Persistable, NetFactor, ObjectMappable, Persistable, DisplayIdentification

2

```
@GenAsPersistable(superClass=WTObject.class,
                  versions=-4333025332245130 71L,
                  properties={
                  @GeneratedProperty(name="name",type=java.lang.String.class,supportedAPI=PUBLIC,
                     javaDoc="The name of the View.  Must be unique.",constraints=@PropertyConstraints(upperLimit=30,
                     required=true),columnProperties=@ColumnProperties(unique=true)),
                  @GeneratedProperty(name="sortId",type=int.class,accessors=@PropertyAccessors(setAccess=PACKAGE))),
                  tableProperties=@TableProperties(tableName="WTView"))
public final class View
extends _View
```

Files with a “_” prefix are Generated

Annotation Processors creates parent classes.

- Common practice to extend _[Classname] when creating a class.
- Compiler initially gives errors but then automatically recompiles.

JavaDoc Samples

Out-Of-The-Box code provides good examples

wt.part.WTPart

- derivedProperties & foreignKeys

wt.vc.baseline.BaselineMember

- Example association

wt.part.PartType

- Example enumerated type

wt.part.partResource

- Example resource bundle

wt.fc.PersistenceManager & wt.fc.StandardPersistenceManager

- Example service

com.ptc.windchill.annotations.metadata

- Complete JavaDoc for the annotations
- See, especially, the package JavaDoc

Samples

From **WTPart**

```
178    foreignKeys={
179    @GeneratedForeignKey( /* first defined by: Iterated */
180       foreignKeyRole=@ForeignKeyRole(name="master", type=wt.part.WTPartMaster.class, supportedAPI=SupportedAPI.PUBLIC, cascade=false,
181          constraints=@PropertyConstraints(required=true)),
182       myRole=@MyRole(name="iteration", supportedAPI=SupportedAPI.PUBLIC, cascade=false))
183    },
184    derivedProperties={
185    @DerivedProperty(name="number", derivedFrom="master>number", supportedAPI=SupportedAPI.PUBLIC),
186    @DerivedProperty(name="name", derivedFrom="master>name", supportedAPI=SupportedAPI.PUBLIC),
187    @DerivedProperty(name="defaultUnit", derivedFrom="master>defaultUnit", supportedAPI=SupportedAPI.PUBLIC),
188    @DerivedProperty(name="endItem", derivedFrom="master>endItem", supportedAPI=SupportedAPI.PUBLIC),
189    @DerivedProperty(name="defaultTraceCode", derivedFrom="master>defaultTraceCode", supportedAPI=SupportedAPI.PUBLIC),
190    @DerivedProperty(name="phantom", derivedFrom="master>phantom", supportedAPI=SupportedAPI.PUBLIC),
191    @DerivedProperty(name="hidePartInStructure", derivedFrom="master>hidePartInStructure", supportedAPI=SupportedAPI.PUBLIC),
192    @DerivedProperty(name="servicekit", derivedFrom="master>servicekit", supportedAPI=SupportedAPI.PUBLIC),
193    @DerivedProperty(name="serviceable", derivedFrom="master>serviceable", supportedAPI=SupportedAPI.PUBLIC)
194    },
```

Sample from **derivedProperties** & **foreignKeys**

From **BaselineMember**

```
47  @GenAsBinaryLink(superClass=ObjectToObjectLink.class,
48     versions={2538346186404157511L},
49     roleA=@GeneratedRole(name="theBaseline", type=Baseline.class, supportedAPI=SupportedAPI.PUBLIC),
50     roleB=@GeneratedRole(name="theBaselineable", type=Baselineable.class, supportedAPI=SupportedAPI.PUBLIC,
51        owner=false),
```

Sample Associations

From **PersistenceManager** and **StandardPersistenceManager**

```
50  @RemoteInterface
51  public interface PersistenceManager {
```

Sample of a Service

```
172 public class StandardPersistenceManager extends StandardManager implements PersistenceManager, PersistenceManagerSvr, Serializable {
173    private static final String RESOURCE = "wt.fc.fcResource";
174    private static final String CLASSNAME = StandardPersistenceManager.class.getName();
```

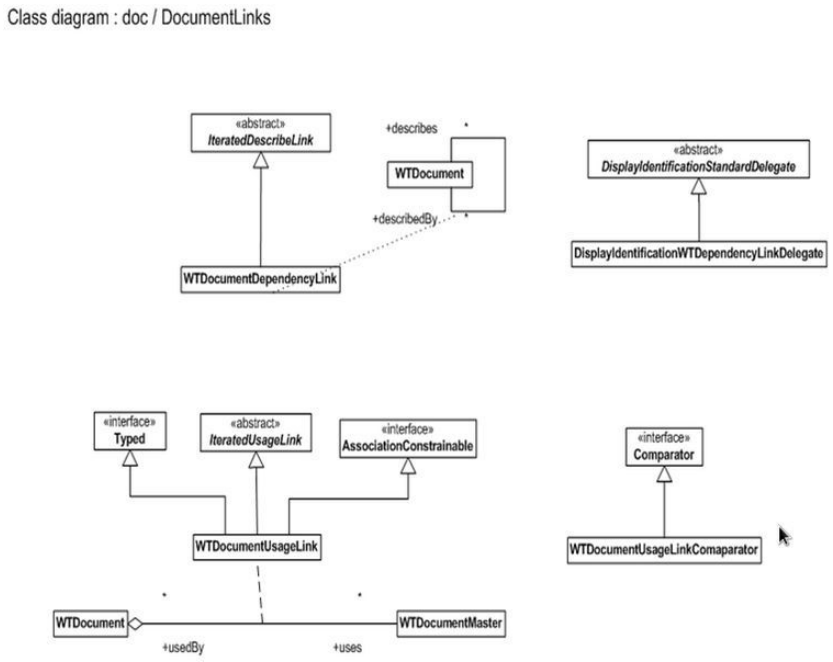
Sample of a Service (Standard)

Javadoc Pictures

Model is Documented in the Javadoc

wt.access&wt.doc

Example of (new) “standard” practice of including diagrams in Javadoc



Javadoc Summary

How about .... It's in there ...

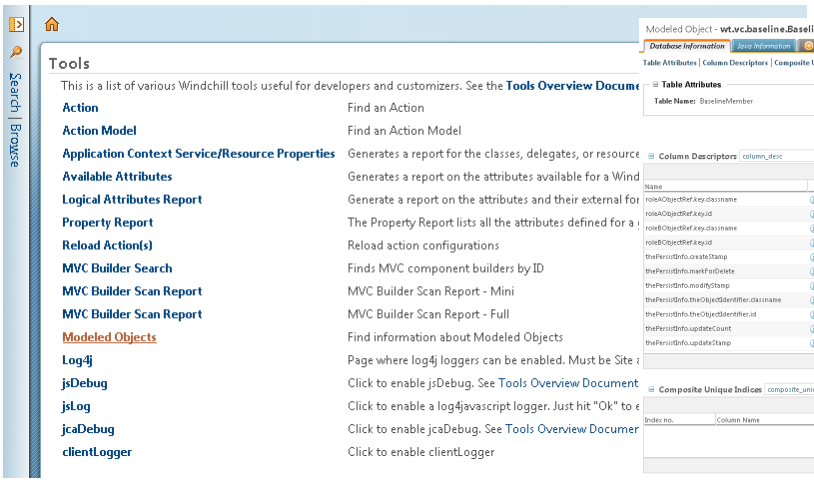
Everything is in JavaDoc

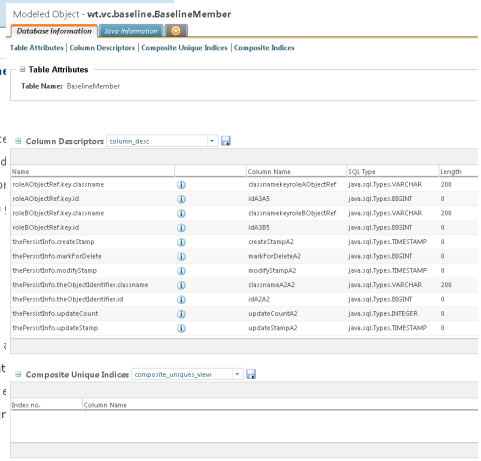
- Modeling artifacts and their documentation (**com.ptc.Windchill.annotations.metadata**)
- The model itself (@GenAs…)
- Bundles and bundle constants
- Even (static) pictures!

Modeled Objects tool in Customization Tab

Object Information is available in the product

Access the Tool from the Customization Tab under Tools





Accessing the Tool

BaselineMember as displayed in the Tool

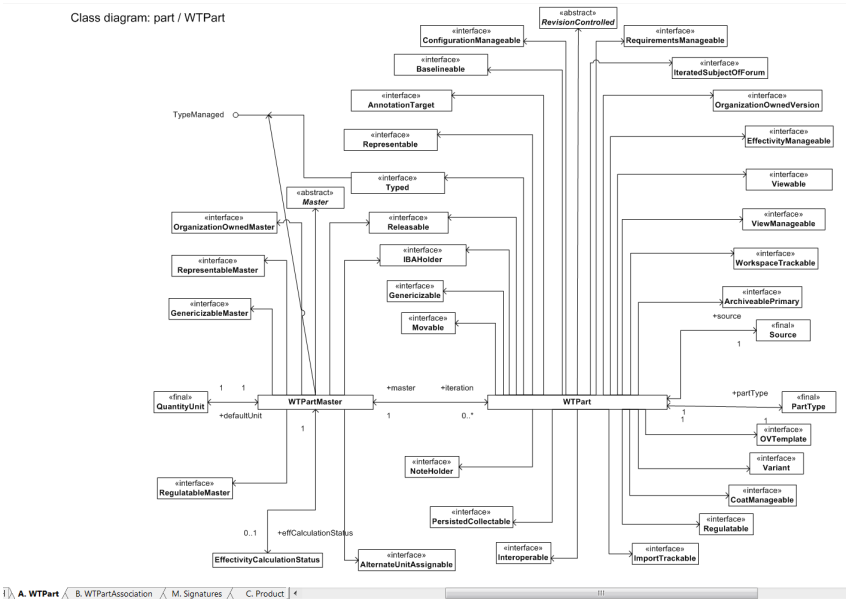
Visio Files

Model now Documented in Visio Only

OpenGrok is the only source for this information:

- http://ah-grok.ptcnet.ptc.com/search?q=&project=main&defs=&refs=&path=vsd+docfiles&hist

AH-GROK



Sample Model

Module 4

Annotations in General

Module Overview

In this module, we will describe annotations in general as they apply to Java coding as well as Windchill coding in version 11.

Objectives

After completing this module, you will be able to:

- Describe the use of common Java Annotations
- Describe how annotations are processed by the Java compiler in Java 1.7
- Identify Annotations in the Windchill Source
- Identify Annotation Processors in Windchill.

Strict Definition

What and Why?

Annotations are:

- A special form of syntactic metadata that can be added to Java source code.
- Reflective — can be embedded in class files generated by the compiler and may be retained by the Java VM to be made retrievable at run-time.
- Intended to provide:
    1. Additional semantics for various class elements which can help developers to understand the intent.
    2. Execution of additional compile time checks that ensure various constraints are met.
    3. Support of additional code analysis by annotation-aware tools.

Standard Built-in Annotations

Annotations Built-In to Java

Annotation Interface	Applicable To	Purpose
Deprecated	All	Marks item as deprecated
SuppressWarnings	All but packages and annotations	Suppresses warnings of the given type
Override	Methods	Checks that this method overrides a superclass method
PostConstruct PreDestroy	Methods	The marked method should be invoked immediately after construction or before removal
Resource	Classes, interfaces, methods, fields	On a class or interface: marks it as a resource to be used elsewhere. On a method or field: marks it for "injection"
Resources	Classes, interfaces	An array of resources
Generated	All	Marks item as source code that has been generated by a tool
Target	Annotations	Specifies the items to which this annotation can be applied
Retention	Annotations	Specifies how long this annotation is retained
Documented	Annotations	Specifies that this annotation should be included in the documentation of annotated items
Inherited	Annotations	Specifies that this annotation, when applied to a class, is automatically inherited by its subclasses

Example of Standard Annotation

@Override and @Depricated

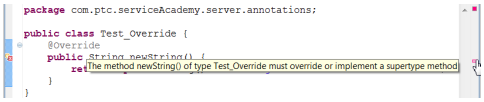
@Override forces the method to override a parent method

- Compiler complains if the method doesn't override

```
package com.ptc.serviceAcademy.server.annotations;

public class Test_Override {
    @Override
    public String toString() {
        return super.toString() + "Testing annotation name: 'Override'";
    }
}
```

Overriding



Not Overriding — Compiler error

Classes annotated @Depricated will display compiler warnings when used.

```
package com.ptc.serviceAcademy.server.annotations;

public class Test_Depricated {
    @Deprecated
    public void doSometing(){
        System.out.println("Testing annotation name: 'Depricated'");
    }
}
```

@Depricated

```
package com.ptc.serviceAcademy.server.annotations;

public class Test_Annotations {
    public static void main(String[] args) {
        new Test_Annotations();
    }

    public Test_Annotations() {
        Test_Depricated t2 = new Test_Depricated();
        t2.doSometing();
    }
}
```

The method doSometing() from the type Test_Depricated is deprecated

1 quick fix available:

@ Add @SuppressWarnings 'deprecation' to 'Test_Annotations()'

Press 'F2' for focus

Use of Deprecated Class

Additional Notes

Warning disappears when the following code is added.

@SuppressWarnings("deprecation")

Clicking the warning will add this line of code automatically.

Creating your own Annotation

Creating a New Annotation for the Javadoc

You can easily create your own annotations by defining a new annotation type.

- @interface key word

```
package com.ptc.serviceAcademy.server.annotations;

import java.lang.annotation.Documented;

@Documented
public @interface Test_Documented {
    String doTestDocument();
}
```

Definition of New Annotation

```
package com.ptc.serviceAcademy.server.annotations;

public class Test_Annotations {
    public static void main(String[] args) {
        new Test_Annotations();
    }

    public Test_Annotations() {
        Test_Depricated t2 = new Test_Depricated();
        t2.doSometing();
    }

    @Test_Documented(doTestDocument = "Hello document")
    public void doSomeTestDocumented(){
        System.out.println("Testing annotation 'Documented'");
    }
}
```

Using the New Annotation

Constructor Detail

Test_Annotations

```
public Test_Annotations()
```

Method Detail

main

```
public static void main(java.lang.String[] arg)
                throws java.lang.Exception
```

Throws:

java.lang.Exception

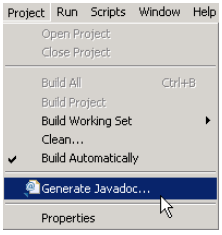
doSomeTestDocumented

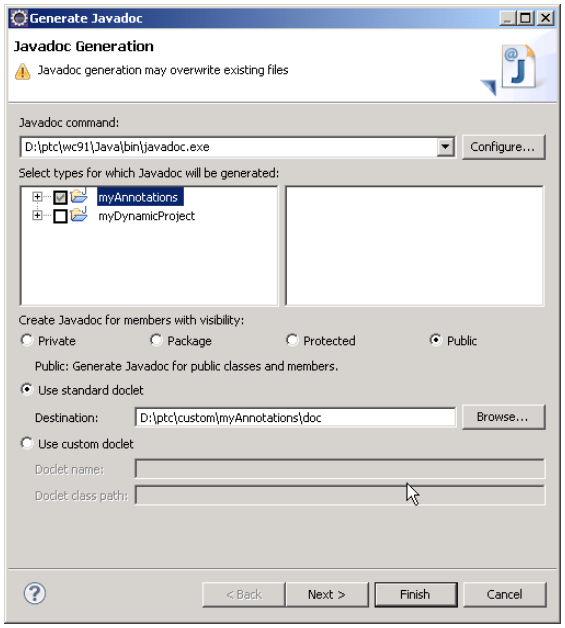
```
@Test_Documented(doTestDocument="Hello document")
public void doSomeTestDocumented()
```

Generated Javadoc contains Additional Comments

Additional Details

You can generate javadoc from Eclipse as follows:




Generate Javadoc
Javadoc Generation
Javadoc generation may overwrite existing files
Javadoc command:
D:\ptc\wc91\Java\bin\javadoc.exe
Configure...
Select types for which Javadoc will be generated:
myAnnotations
myDynamicProject
Create Javadoc for members with visibility:
Private
Package
Protected
Public
Public: Generate Javadoc for public classes and members.
Use standard doclet
Destination:
D:\ptc\custom\myAnnotations\doc
Browse...
Use custom doclet
Doclet name:
Doclet class path:
< Back
Next >
Finish
Cancel

Target Annotation

@Target indicates the elements in which the annotation will be applicable

In this case, if we define the annotation as follows, it can only be applied to a method:

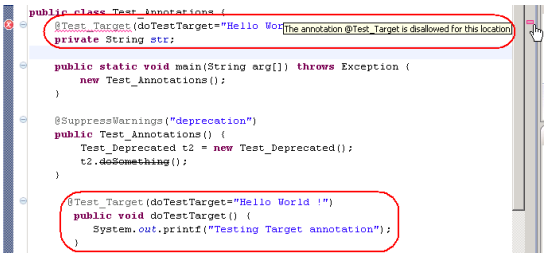
```
package com.ptc.serviceAcademy.server.annotations;

import java.lang.annotation.ElementType;
import java.lang.annotation.Target;

@Target(ElementType.METHOD)
public @interface Test_Target {
    public String doTestTarget();
}
```

ElementType.METHOD

When applied to a variable, the compiler complains:



Only applies to Methods

The table below lists the variations:

Annotation	Only applies to
@Target(Element-Type.TYPE)	any element of a class
@Target(Element-Type.FIELD)	a field or property
@Target(Element-Type.METHOD)	a method level annotation
@Target(Element-Type.PARAMETER)	the parameters of a method
@Target(Element-Type. CONSTRUCTOR	constructors
@Target(Element-Type.LOCAL_ VARIABLE)	local variables
@Target(Element-Type. ANNOTATION_ TYPE)	an annotation type

Retention Annotation

Retention annotation indicates where and how long annotations with this type are to be retained

There are three values

- **RetentionPolicy.SOURCE**
  - Retained only at the source level and will be ignored by the compiler
- **RetentionPolicy.CLASS**
  - Retained by the compiler at compile time, but will be ignored by the VM
- **RetentionPolicy.RUNTIME**
  - Retained by the VM so they can be read only at runtime.

```
package com.ptc.serviceAcademy.server.annotations;

import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;

@Retention(RetentionPolicy.RUNTIME)
public @interface Test_Retention {
    String doTestString();
}
```

Annotation that can only be read at runtime

Annotations in Windchill

Windchill uses these Annotations

GenAsPersistable will be discussed in more detail later.

- Custom Annotation
- Uses
  - Retention
  - Target
  - Documented

```
@Retention(RetentionPolicy.CLASS)
@Target(ElementType.TYPE)
@Documented
public @interface GenAsPersistable {
    /**
     * The parent class for this class. This class would normally
     * <code>extends</code> statement and is needed because the a
     * <i>"_"+classname"</i> (for example <code>public MyClass ex
     * <p>
     * Note: the parent should either be {@link Object} or some a
     * implements {@link wt.fc.Persistable Persistable}, like {@l
     * {@link wt.enterprise} abstractions. Also, if you want to e
     * {@link GenAsBinaryLink} instead.
     * <p>
     * Rose conversion note: Corresponds to Class Specification
     * (where the specialized class is the single (non-interface)
     * <p>
     * <b>Supported API: </b>true
     */
    @SuppressWarnings("unchecked")
    Class superClass() default Object.class;
```

Some Annotations Are not displayed by the Decompiler

GenAsPersistable.class

```
package com.ptc.windchill.annotations.metadata;

import java.lang.annotation.Annotation;
import java.lang.annotation.Documented;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Retention(RetentionPolicy.CLASS)
@Target({java.lang.annotation.ElementType.TYPE})
@Documented
public @interface GenAsPersistable
{
  public abstract Class superClass();

  public abstract Class[] interfaces();

  public abstract GeneratedProperty[] properties();

  public abstract GeneratedForeignKey[] foreignKeys();

  public abstract DerivedProperty[] derivedProperties();

  public abstract Serialization serializable();

  public abstract boolean extendable();

  public abstract long[] versions();

  public abstract int depthLock();

  public abstract TableProperties tableProperties();

  public abstract IconProperties iconProperties();
}
```

As seen by a Decompiler

Module 5

Calling the Method Server

Module Overview

In this module, we highlight how to invoke commands on a method server. In the past, helpers used a forwarder or service design pattern. Aannotations are used to model this pattern..

Objectives

After completing this module, you will be able to:

- Provide examples of using annotations for constructing Helpers
- Demonstrate the use of the “@Remote” annotation.

Fwders are Replaced by Annotations

Fwder->@RemoteInterface

- Background
  - Long-standing pattern of helper, remote interface, generated (RMI) forwarder, standard service
- Motivation
  - Boilerplate code can be handled by Java without need for code
- Mechanism
  - Annotation + dynamic proxies
  - See JavaDoc for RemoteInterface, ServiceFactory, various services

wt.method

Annotation Type RemoteInterface

```
@Documented
@Retention(value=RUNTIME)
@Target(value=TYPE)
public @interface RemoteInterface
```

Indicates that the methods of an interface can be invoked remotely by the RemoteMethodServer. RemoteInterface implementations are used in conjunction with a `RemoteAccessProxy`.

This annotation shares similar semantics with the <<RemoteInterface>> stereotype in Windchill code generation, and is typically used to denote remote Windchill services.

Supported API: true

Extendable: false

Javadoc provides Implementation pattern

wt.services

Class ServiceFactory

java.lang.Object
  wt.services.ServiceFactory

```
public final class ServiceFactory
extends Object
```

Creates implementations of Windchill service interfaces. Service interfaces must have implementations that also implement `Manager`. This is typically achieved by extending `StandardManager`. In addition to implementing Manager, the service interface/implementation mapping must be registered in wt.properties.

To expose a remote interface via the ServiceFactory, you must annotate the interface with `RemoteInterface`:

```
@RemoteInterface
public interface MyService {
  ...
}
```

Once you have registered your service, you can retrieve an implementation of it using the following code:

```
MyService service = ServiceFactory.getService(MyService.class);
```

Supported API: true

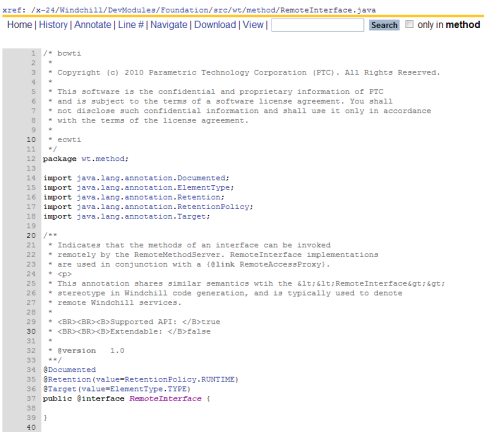
Extendable: false

Introducing @RemoteInterface

Annotation @RemoteInterface is used to create a Fwder

Fwder Replacement Via Simple Dynamic Proxy

- Dynamic proxies dynamically implement Fwders at runtime
- See wt.services.ServiceFactory (Foundation)


xref: /x-24/Windchill/DevModules/Foundation/src/wt/method/RemoteInterface.java

Home | History | Annotate | Line # | Navigate | Download | View | Search only in method

```
/* bowti
 *
 * Copyright (c) 2010 Parametric Technology Corporation (PTC). All Rights Reserved.
 *
 * This software is the confidential and proprietary information of PTC
 * and is subject to the terms of a software license agreement. You shall
 * not disclose such confidential information and shall use it only in accordance
 * with the terms of the license agreement.
 *
 * ecwti
 */
package wt.method;

import java.lang.annotation.Documented;
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
 * Indicates that the methods of an interface can be invoked
 * remotely by the RemoteMethodServer. RemoteInterface implementations
 * are used in conjunction with a {@link RemoteAccessProxy}.
 * <p>
 * This annotation shares similar semantics wtih the &lt;&lt;RemoteInterface&gt;&gt;
 * stereotype in Windchill code generation, and is typically used to denote
 * remote Windchill services.
 *
 * <BR><BR><B>Supported API: </B>true
 * <BR><BR><B>Extendable: </B>false
 *
 * @version   1.0
 **/
@Documented
@Retention(value=RetentionPolicy.RUNTIME)
@Target(value=ElementType.TYPE)
public @interface RemoteInterface {

}
```

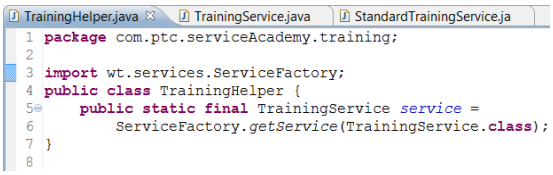
RemoteInterface.java

Implementing a Helper Service

Three objects need to be created

Helper that calls the Service

```
@RemoteInterface
public static final TrainingSer-
vice service = ServiceFactory.get-
Service(XXXService.class);
```



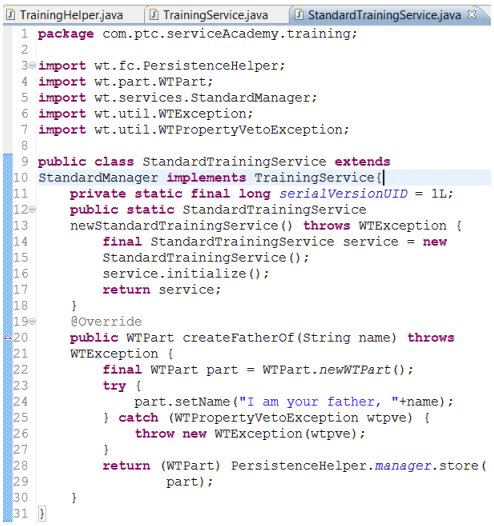
Helper — Call the Service

Service that calls the "Standard" Service

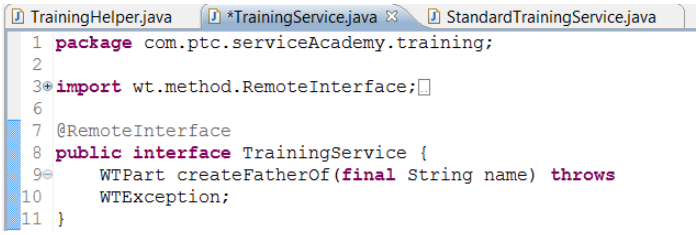
```
@RemoteInterface public inter-
face
XXXService {...}
```

"Standard" Service

```
public class StandardXXXSer-
vice extends StandardManager
implements XXXService {...
@Override
public ... { ... } ...}
```



StandardService — Runs on Method Server



Service — Runs Remotely

PersistenceHelperServiceConstants

PersistenceHelperServiceConstants demonstrates differences in the new architecture

Why The New Class?

- Fwders don't "cache" the actual Manager (getManager is no longer used)
- When starting your service, you can not use other services
- @RemoteInterface Returns the actual service/manager…

The Helpers Are No Longer Safe During MethodServerStartup

- Create a parent Helper only when needed for service-required constants, etc.
- Services needing constant use parent
- Others (continue to) use "real" Helper (LWC/src/com/ptc/core/lwc/server/cache/ConstraintFactory.java)

```
13 package wt.fc;
14
15 import wt.dataservice.DataServiceFactory;
16 import wt.dataservice.Datastore;
17 import wt.dataservice.Oracle;
18 import wt.util.WTProperties;
19
20 public abstract class PersistenceHelperServiceConstants {
21    private static final String CLASSNAME = PersistenceHelperServiceConstants.class.getName();
22
23    public final static int DB_MAX_BYTES_PER_CHAR;
24    public final static int DB_MAX_SQL_STRING_SIZE;
25
26    static {
27       try {
28          WTProperties properties = WTProperties.getLocalProperties();
29          Datastore datastore = DataServiceFactory.getDefault().getDatastore();
30
31          DB_MAX_BYTES_PER_CHAR = (datastore instanceof Oracle) ?
32                properties.getProperty("wt.db.maxBytesPerChar", 1) : 1;
33
34          DB_MAX_SQL_STRING_SIZE = properties.getProperty("wt.db.maxSqlStringSize", 4000);
35       }
36       catch (Throwable t) {
37          System.err.println("Error initializing " + CLASSNAME);
38          t.printStackTrace();
39          throw new ExceptionInInitializerError(t);
40       }
41    }
42 }
```

PersistenceHelperServiceConstants

```
56 public class PersistenceHelper extends PersistenceHelperServiceConstants implements Serializable {
57    private static final String RESOURCE = "wt.fc.fcResource";
58    private static final String CLASSNAME = PersistenceHelper.class.getName();
59
60    /**
61     * <BR><BR><B>Supported API: </B>true
62     **/
63    public static final PersistenceManager manager = wt.services.ServiceFactory.getService(PersistenceManager.class);
64
```

PeristanceHelper now uses the Manager (not the FWDer)

Exercise 1: Configure Eclipse for Windchill Development

Objectives

After successfully completing this exercise, you will be able to:

- Set up the developing environment.
- Successfully create Java project and debug.

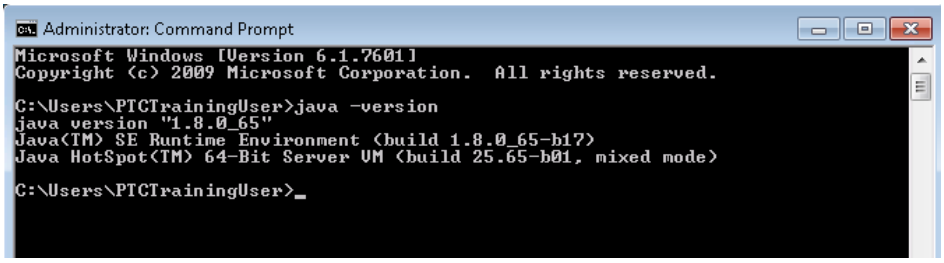
Scenario

In this exercise, you will install the Eclipse and set up some configuration for Windchill, and create a Java project and validate the configuration.

Task 1: Check for Java version.

PLEASE USE E: DRIVE FROM THE VM FOR ALL EXERCISES

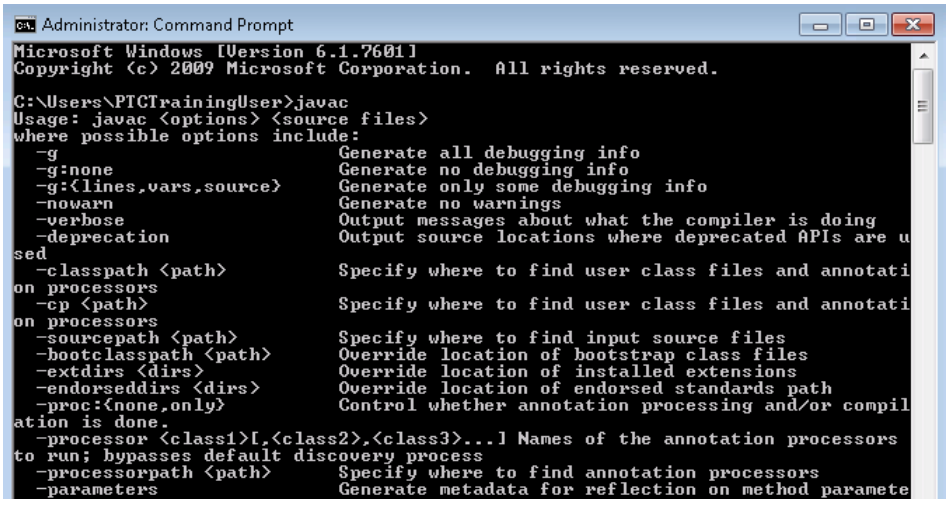
1. JDK should be installed on your local computer.
2. Open Command Prompt and Execute the command: ***java –version*** It should be 1.8 or above.



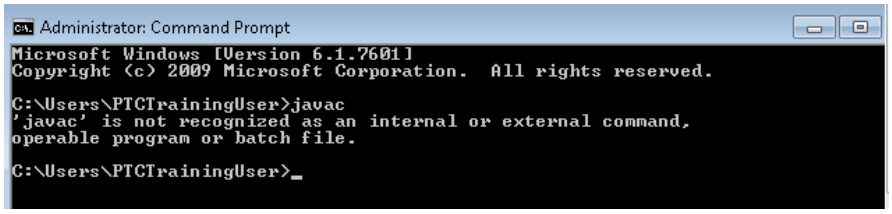
Task 2: Set the Environmental Variables.

1. Execute the command: ***javac***

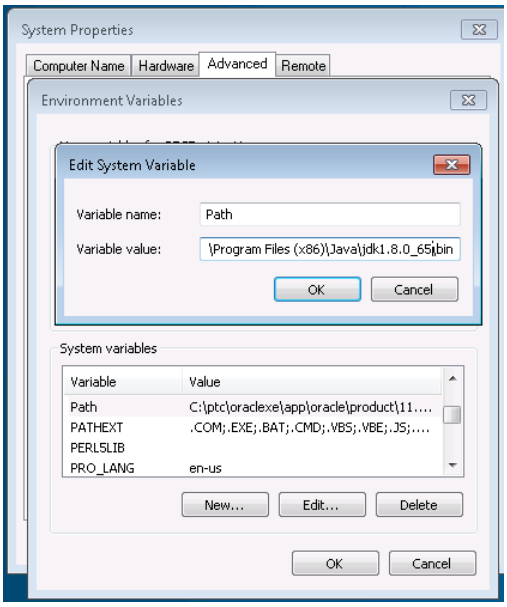
   The command should return “help” information for the command.



   If the command returns the following result, it means your ***environment variable*** is not set properly.



To set your ***environment variable***, navigate to Computer/Properties/Advanced System Settings/Environment Variables and set JAVA_HOME and Path as shown in Environment Variables figure.

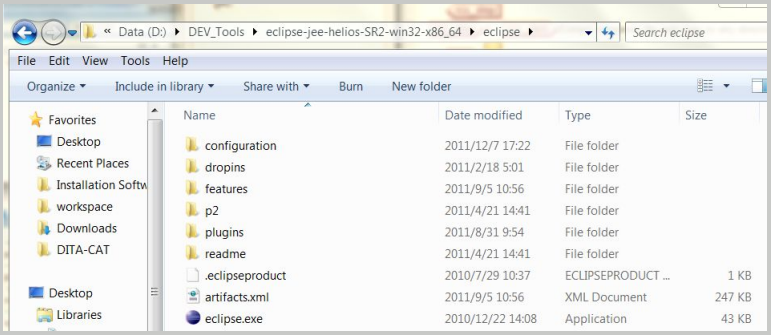


Any spaces in the Java directory path may cause problems.

Task 3: Install Eclipse.

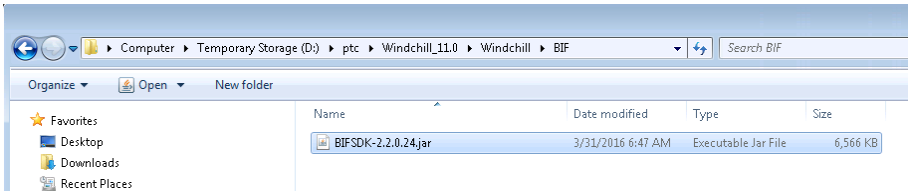
1. Download Eclipse from the following links. The eclipse version is: Helios 3.6 R2
   - Eclipse Helios 3.6 R2 for Windows 64Bit
   - Eclipse Helios 3.6 R2 for Windows 32Bit
2. Extract compressed file and copy to any directory on local machine.

It is widely accepted that all of the developing tools are installed in the directory: *D:\DEV_TOOLS*



Task 4: Configure BIF in Windchill.

1. To Download BIF SDK ,go to https://ssp.ptc.com/wiki/display/bif/BIF The BIF SDK (jar file) can be downloaded from the “Download” tab or go to https://sspsvn.ptc.com/Tools/BIF/11.0/releases/2.2.0/24/
2. Create a copy of BIF to your Windchill development environment. Download the Depricated_BIFSDK-2.2.0.24.jar file to this location as shown in below example and remove ’Depricated_” from its name.



- The jar file is required to execute the BIF installer.
- The contents of the jar do not need to be extracted for execution.
- Temporarily using BIFSDK 2.2.0.24 in the course since found issues in latest release.

3. Open a Windchill shell, and execute following command;

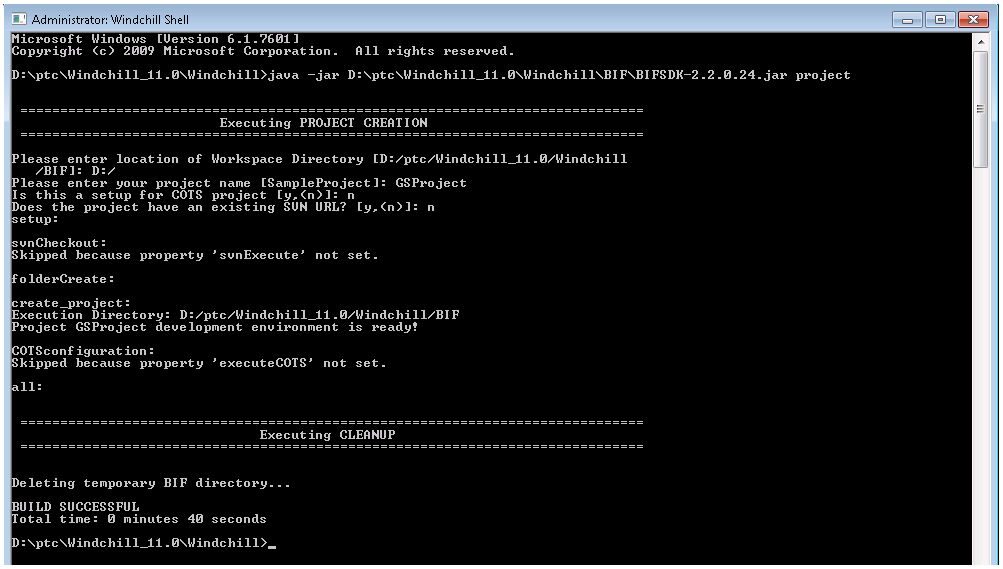
```
java -jar <Location>/BIFSDK-2.2.x.x.jar project
```

If there is exception of <WT_HOME> not defined, then you’ll need to define a user Variable ”WT_HOME” with the path of windchill home directory, for example “D:\ptc\Windchill_11.0\Windchill”.

Answer the following questions presented to create a project:

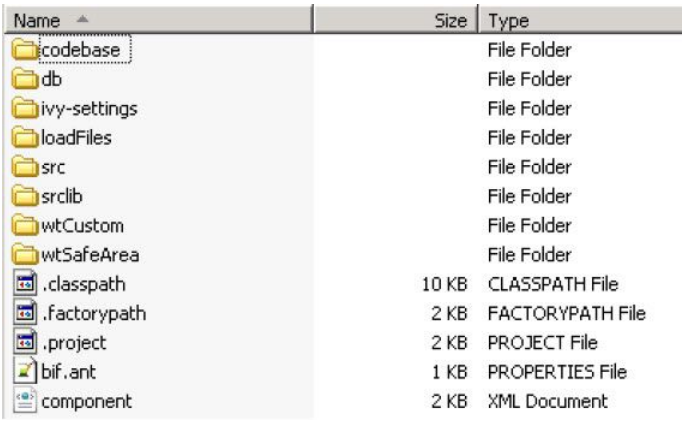
The following questions will be asked:

- The Location of the Workspace directory - Example: If the location of the workspace directory is desired to be D:\Workspace, then the location is D:\.
- The name of the project: **GSProject.**
- If third party libraries should be installed - This is only asked if Ivy is not already installed.
- Is the setup related to COTS: **n**.
- Is there an associated SVN repository: **n**.



The prompts may contain default values, values inside a set of brackets, that can be used by just pressing enter on the prompt. If the default contains a set of values, that are the only valid responses, the default value will be inside a set of parenthesis ( ).

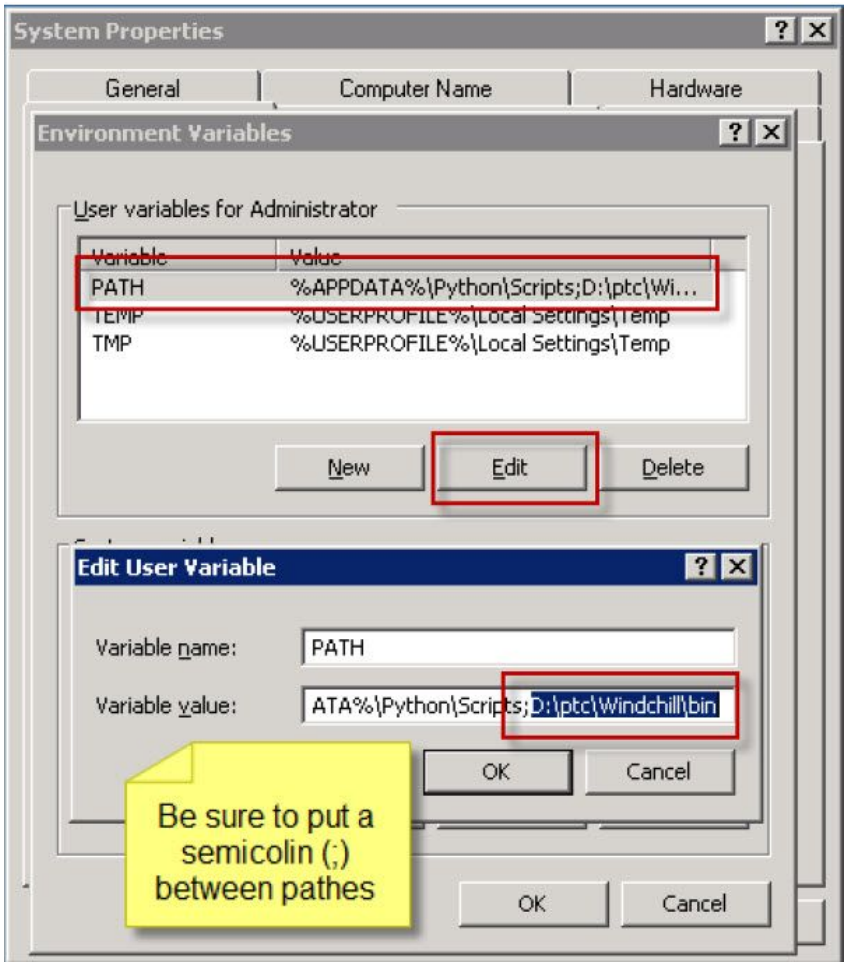
4. A blank project skeleton with the necessary BIF and Eclipse artifacts will be created at the location with the name specified.



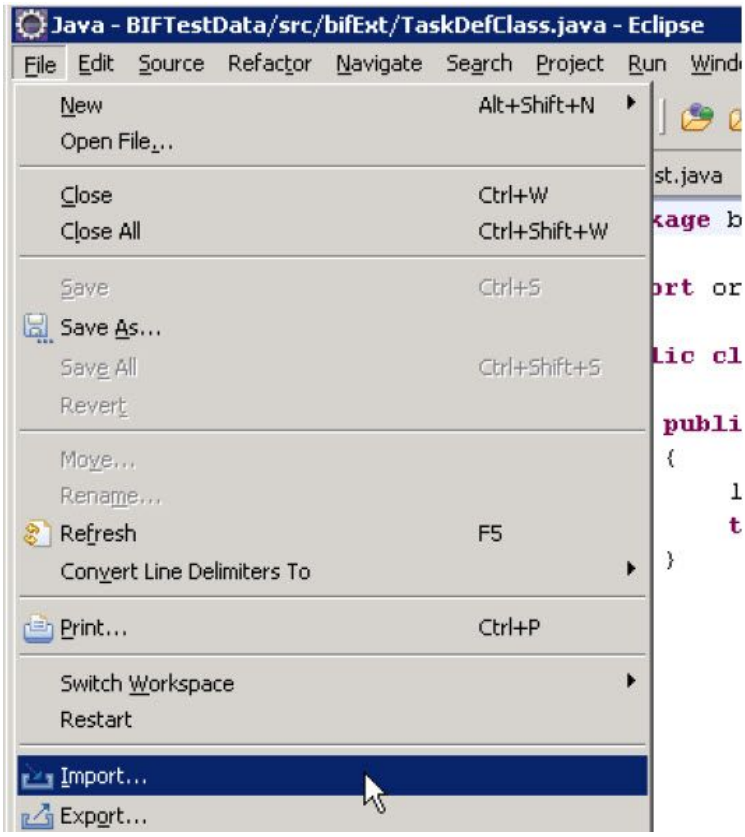
5. Copy the Workspace folder say at **D:\Wokspace**

Task 5: Configure eclipse and Import BIF Project.

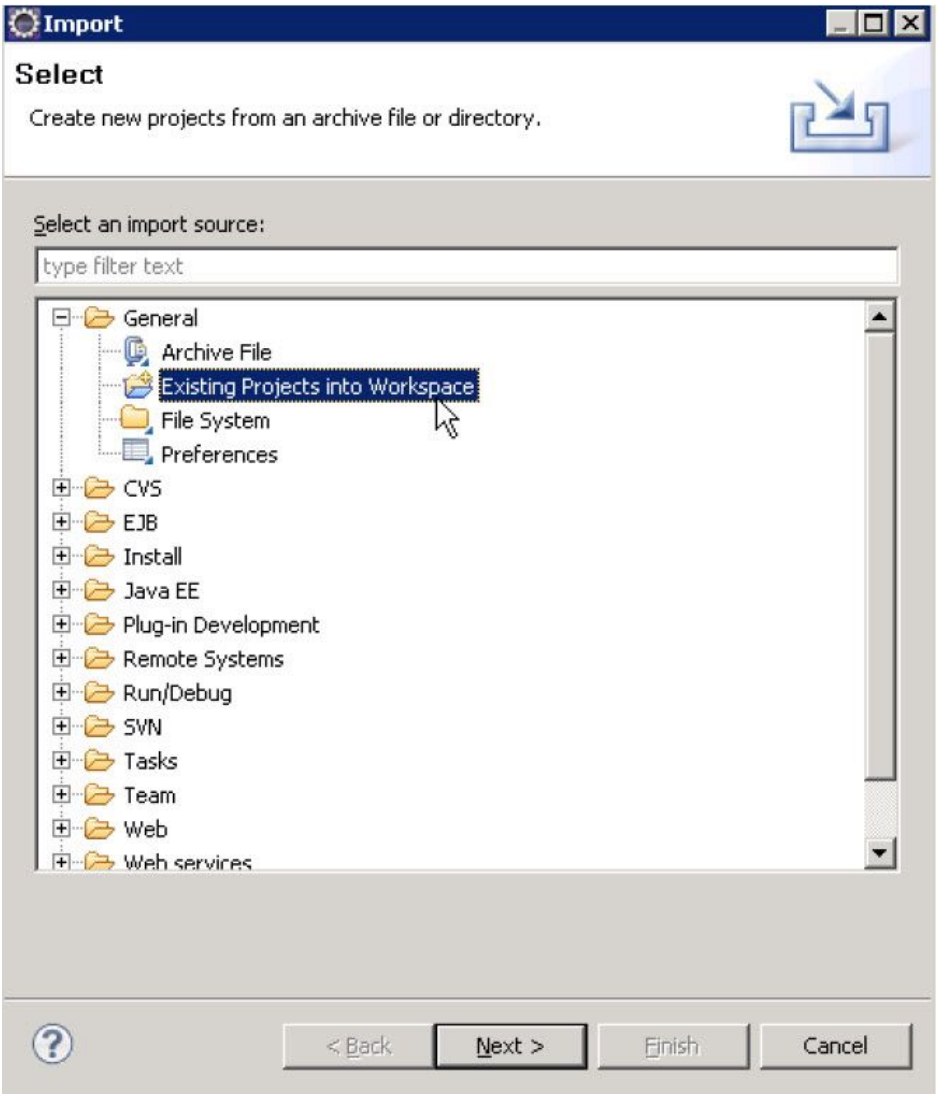
1. BEFORE OPENING ECLIPSE, Add <WT_HOME>/bin to the System Environment variable PATH.
   - Right click on **My Computer** and select **Properties**
   - Select the **Advanced** tab and click the **Environment Variables** button.
   - Highlight the **PATH** variable under *User variables for Administrator* and click the **Edit** button
   - In the *Edit User Variable* window, add <*WT_HOME*>/bin to the end of the *Variable value.*
   - Click **OK**.



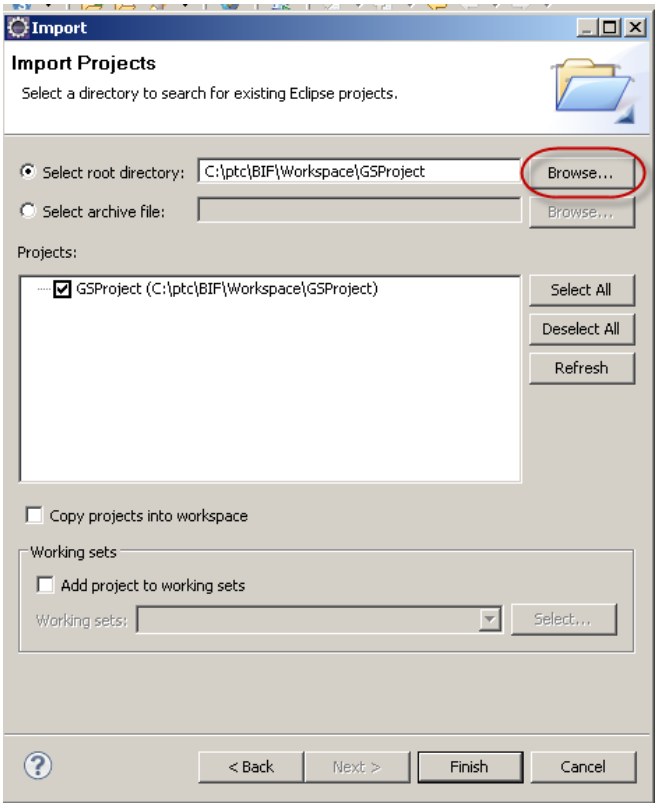
2. **Import BIF Project**
   - Open Eclipse and click **File** --> **Import**



- Select **Existing Projects into Workspace** under the **General** folder within the Import window.

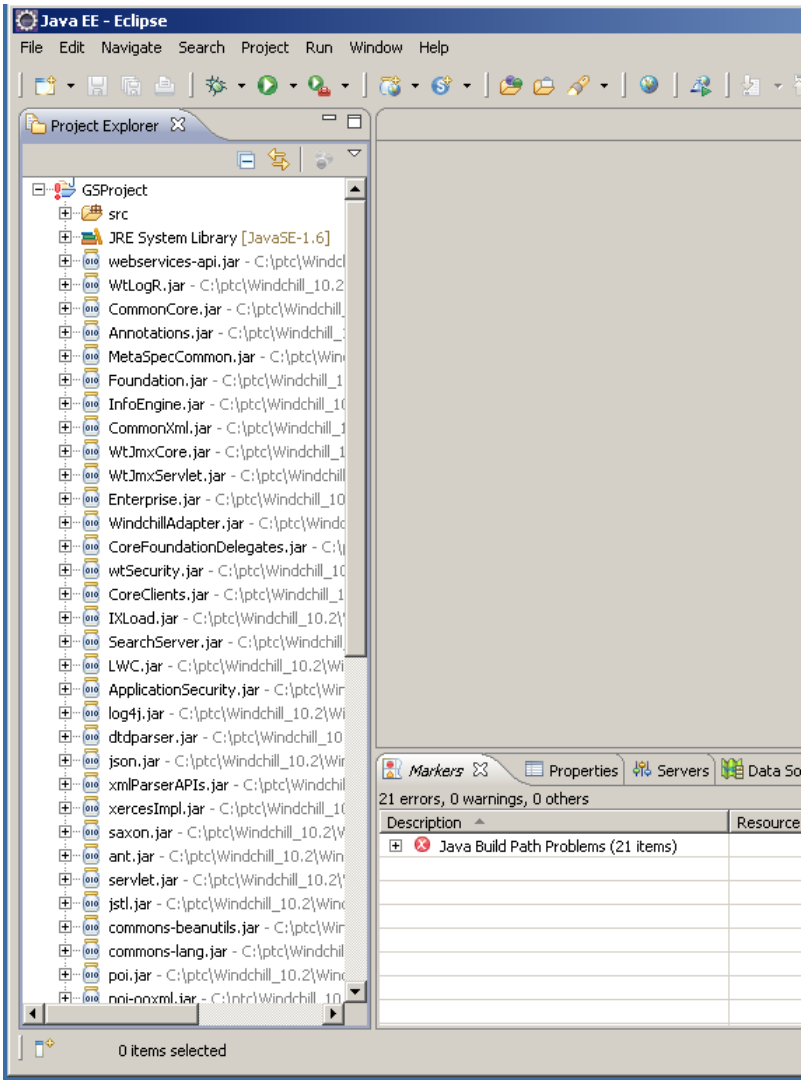


- Browse to the location of the newly created project “GSProject” and select the directory that contains the project.



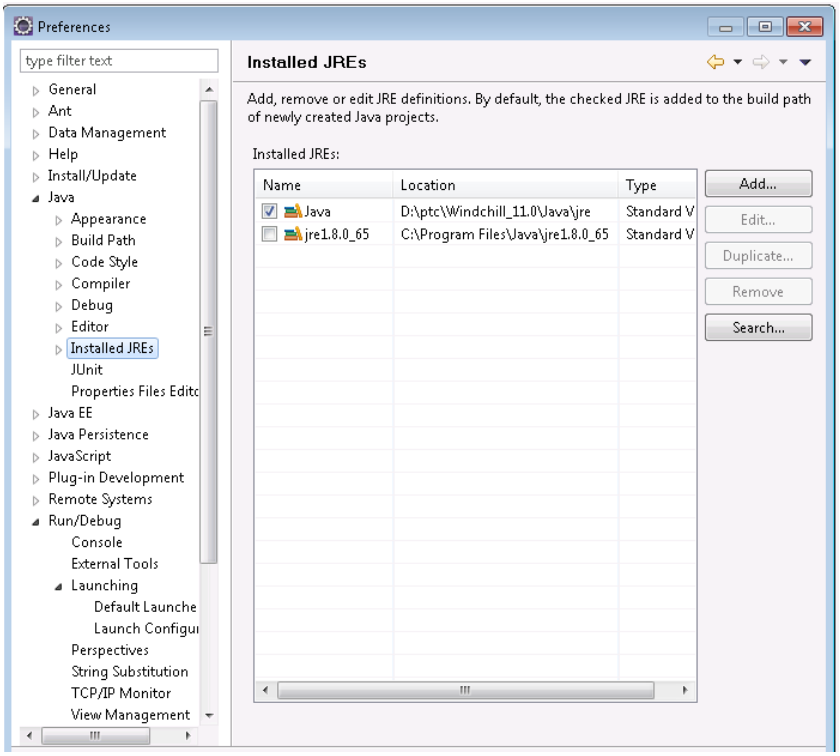
Images shown here are just for reference purpose. Project's path and name may vary depending upon student choice, it is a directory path where student have copied project folder.

- The imported project will be listed within the **Package Explorer**.



3. **Configure JAVA used by Eclipse**
   - Switch the Java JRE Eclipse uses from **Window** -> **Preferences** -> **Java** -> **Installed JREs**
   - Add the Windchill JRE and deselect the default.

Occasionally Errors in Eclipse can be resolved adding JRE 1.7 in libraries



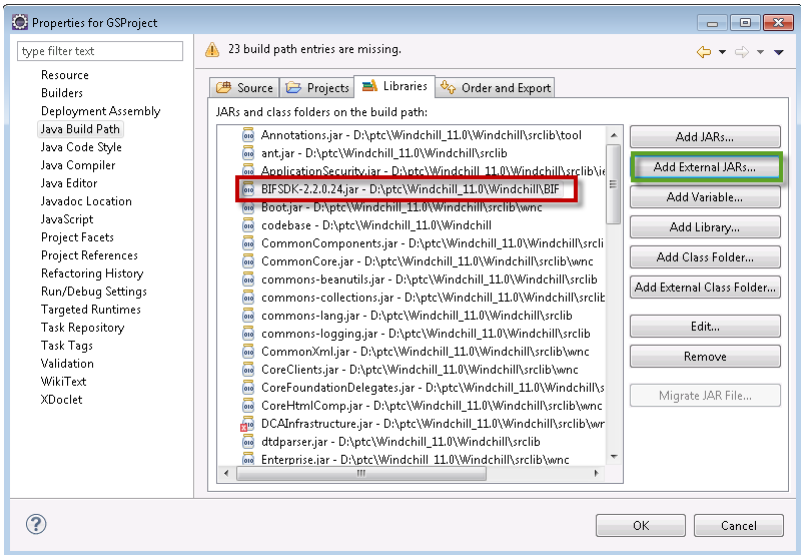
4. Modify Eclipse Preference not to delete Windchill codebase
   - Navigate to ***Window > Preferences > Java > Compiler > Building > Output folder***
   - Uncheck ***Scrub output folder when cleaning projects***

If you leave this option as “checked” then when you are building Windchill project then eclipse will DELETE your “codebase” directory, so better “uncheck” this option.

Task 6: *Configure Eclipse to execute BIF SDK*

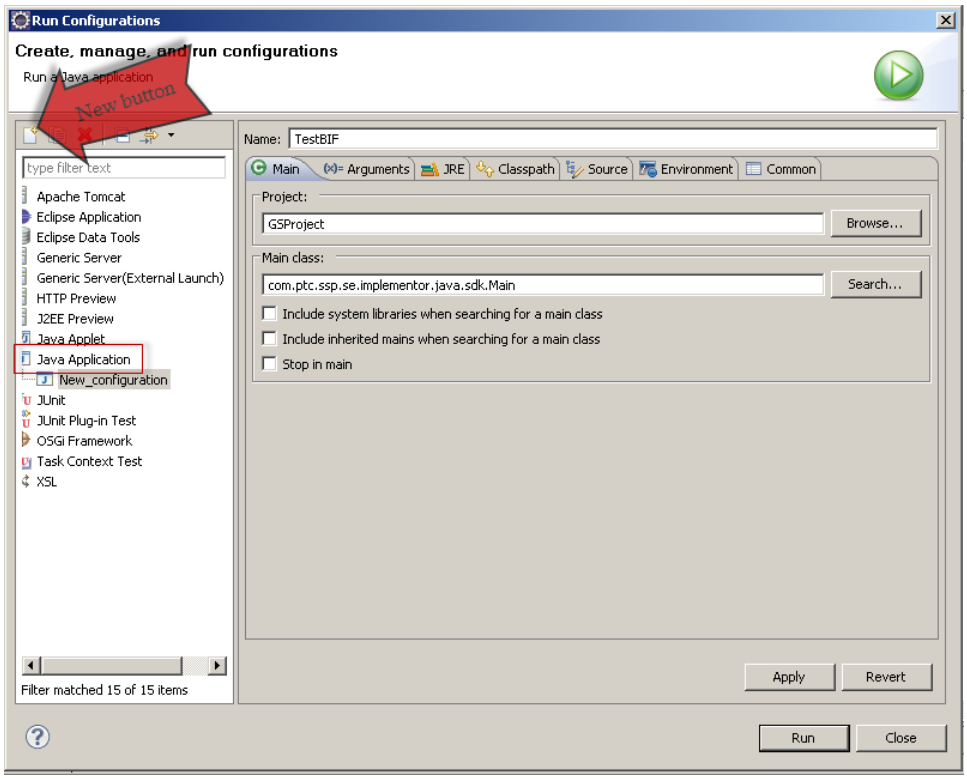
1. Add BIF SDK to the project's Build Path by selecting **Project > Properties**

   Within the Properties Window,
   - Select **Java Build Path**
   - Select the **Libraries** tab
   - Select the **Add External JARs...** button
   - Select the BIFSDK from the global location and click **OK**
   - Click **OK**

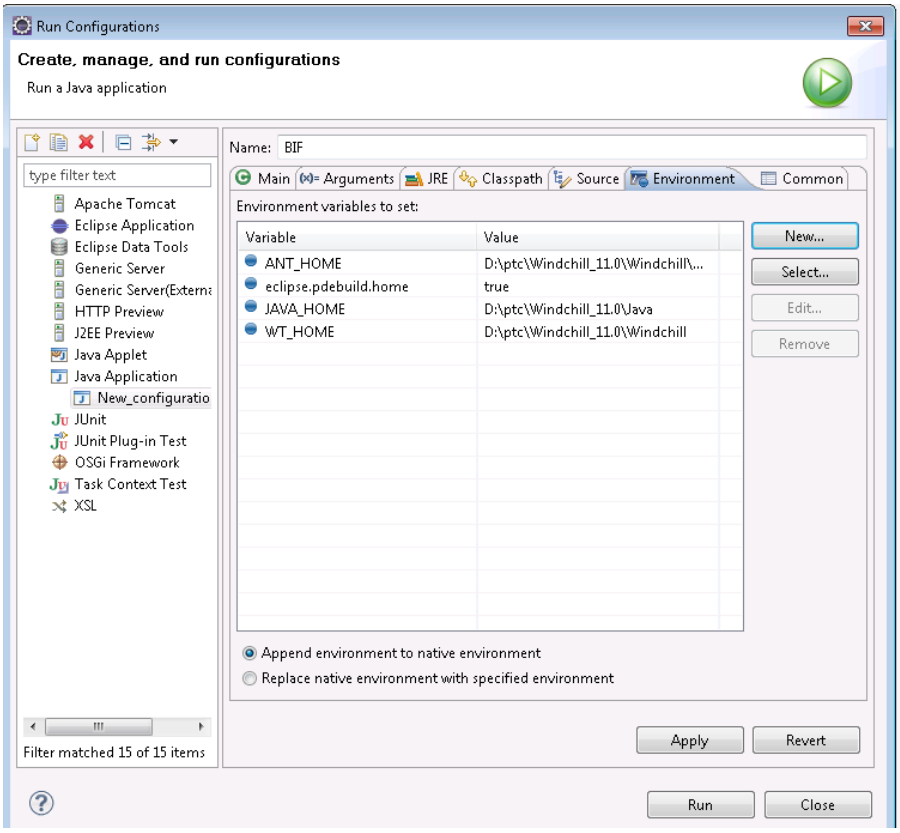


2. Configure your project to run with the BIF by selecting **Run > Run Configurations**
   - Highlight **Java Application** and click the **New** button
   - Enter *BIF* for the **Name** In the **Main** tab
     1. Enter a name in the **Project** field that the BIF will execute against.
     2. In the **Main class** field, enter *com.ptc.ssp.se.implementor.java.sdk.Main*

     I



   - In the **Environment** tab, create the following properties
     - **ANT_HOME** with the value of *<WT_HOME>/ant*
     - **eclipse.pdebuild.home** with the value of *true.*
     - **JAVA_HOME** with the value of *Windchill*'s version of JAVA.
     - **WT_HOME** with the value of *<WT_HOME>.*



- Click **Run** to execute the configuration or **Close** to save the configuration for a later date.

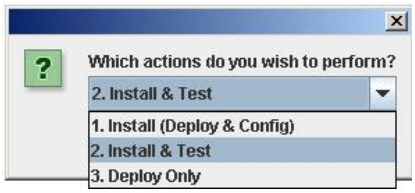
3. During the execution, the BIF will prompt the user which action to run.

For **First Time Execution of Component**, following actions are listed. Select **Install & Test**.

Which actions do you wish to perform:

1. 1. Install (Deploy & Config)
2. Install & Test
3. Deploy Only

Enter valid response [1-3]:



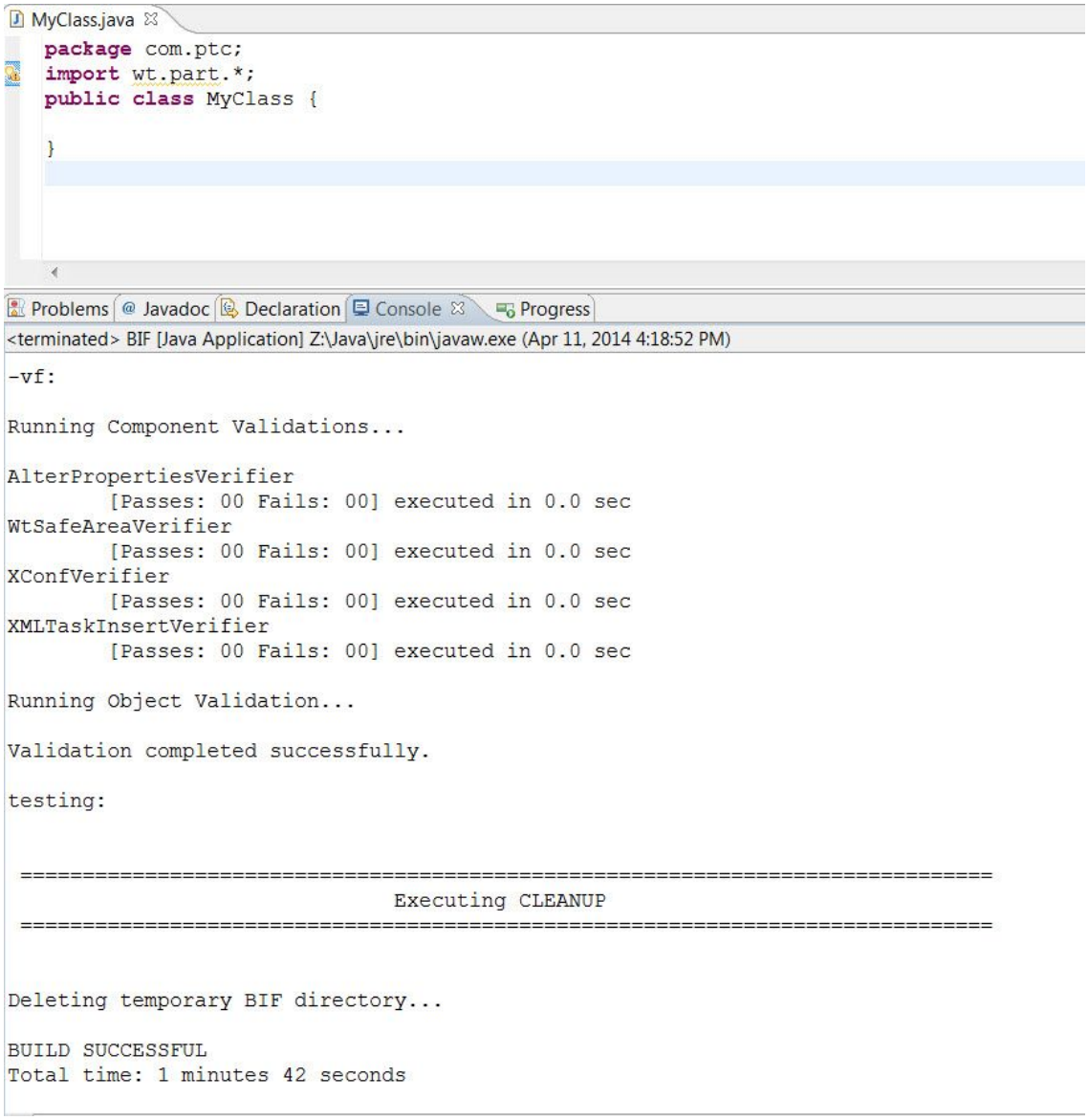
For **Subsequent Executions**, following actions are listed. You can select 'Test Only' action from it or any other action as per your requirement:
Which actions do you wish to perform:

1. Install (Deploy & Config)
2. Install & Test
3. Deploy Only
4. Config Only
5. Config & Test
6. Test Only

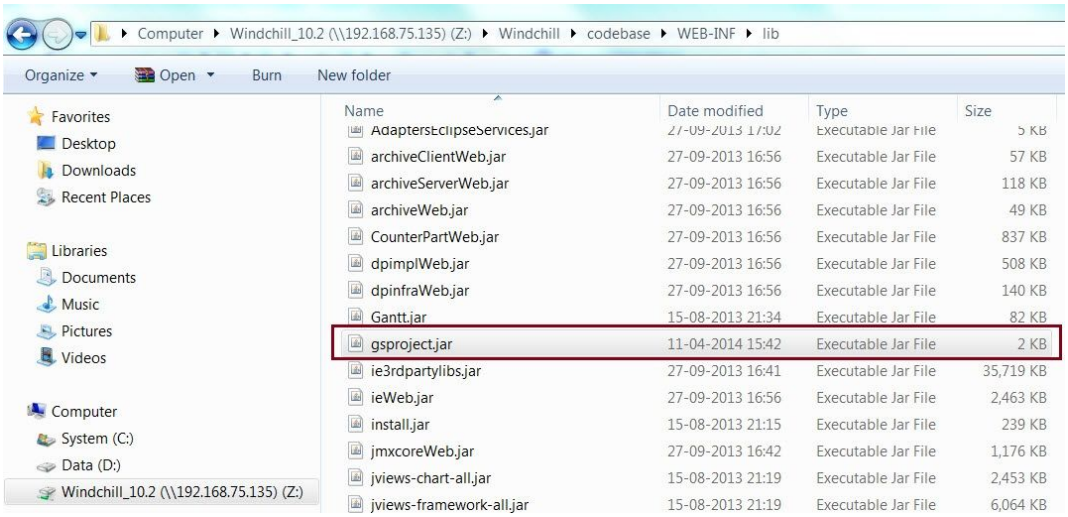
Enter valid response [1-6]:

4. In Console tab, you will see the BUILD SUCCESSFUL message.

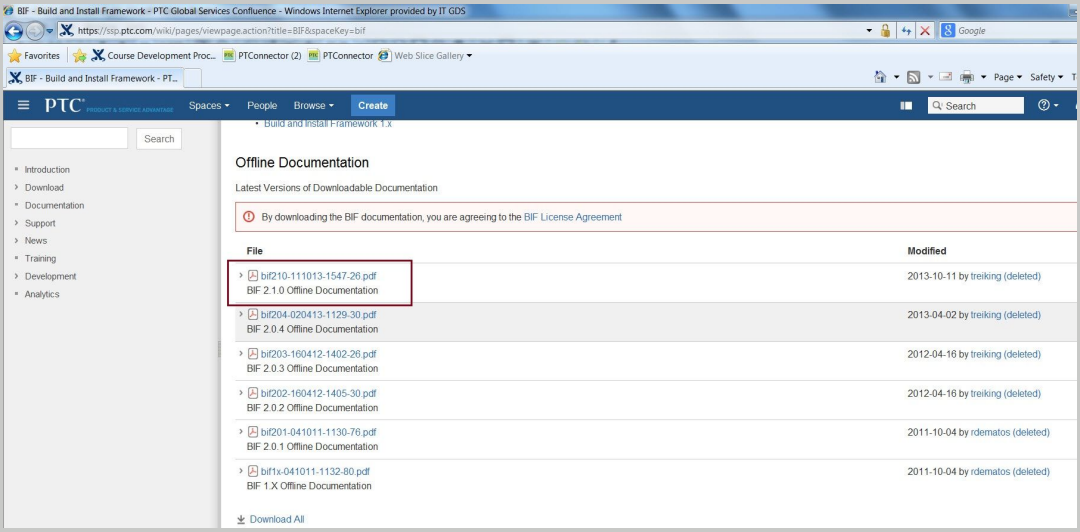
It may take considerable amount of time to build the project successfully.



5. A jar file with project name will be created under %WT_HOME%\codebase\WEB-INF\lib



It is recommended to read the BIF's documentation provided at below link that could help everyone in the future.
Link to download BIF's documentation: Go to https://ssp.ptc.com/wiki/display/bif/BIF and click on respective BIF release Offline Documentation under Documentation tab —> Offline Documentation.



Completion Criteria

1. Eclipse Configured for Windchill.

This completes the exercise.

Exercise 2: Create a Helper that Builds a New Part

Objectives

After successfully completing this exercise, you will be able to:

- Create a Helper
- Create a Service
- Access the Service from the Helper

Scenario

In this exercise, you will use eclipse to call a Helper which will then invoke a service on the method server to create a new part.

If java errors persists in Eclipse Compilation, installing JDK 1.7 and changing it at JRE system libraries at Configure Build Path should resolve them.

```
1  package com.ptc.serviceAcademy;
2
3  import wt.util.WTException;
4  import com.ptc.serviceAcademy.training.TrainingHelper;
5  public class CreatePartFromHelper {
6
7      public static void main(String[] args) {
8          try {
9              TrainingHelper.service.createFatherOf("Luke");
10         } catch (WTException e) {
11             e.printStackTrace();
12         }
13     }
14 }
```

Code is provided in LabFiles.zip.

Task 1: Create the Service

1. The service is simply an Interface using an annotation to specify that it can be called remotely — @RemoteInterface

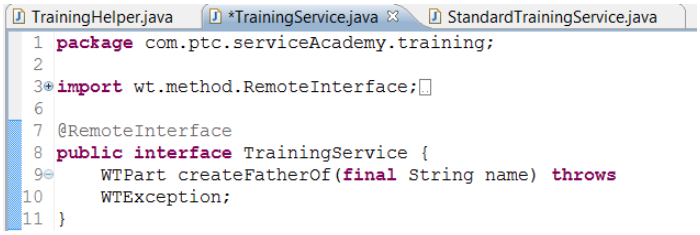
   Example:

   Start with the following template

   ```
   import wt.method.RemoteInterface;
   import wt.util.WTException;
   @RemoteInterface
   public interface xxxService {
   // Methods Go Here.
   }
   ```

2. Add the method definition.

Example:

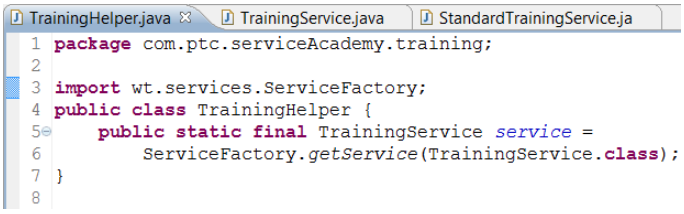


Task 2: Create the Helper that calls the Service

1. Use the following template
   Example:

```
import wt.services.ServiceFactory;
public class xxxHelper {
public static final xxxService
service =
ServiceFactory.getService(xxxService.
class);
}
```

2. Update the template to match the Service defined in the previous step.
   Result:



Task 3: Implement the “Standard Service” which will do the work on the Method Server.

1. Standard Service needs to implement the Service named above. Use the following template to create the standard service.
   Example:

```
public class StandardTrainingService extends StandardManager
implements TrainingService {
public static StandardxxxService newStandardxxxService() throws WTException {
 final StandardxxxService service = new StandardxxxService();
 service.initialize();
 return service;
}
@Override
// Custom Code here
 return (Object) PersistenceHelper.manager.store(theObject);
}
}
```

2. Complete the code as follows:
   Result:

TrainingHelper.java | TrainingService.java | StandardTrainingService.java

```
1  package com.ptc.serviceAcademy.training;
2
3  import wt.fc.PersistenceHelper;
4  import wt.part.WTPart;
5  import wt.services.StandardManager;
6  import wt.util.WTException;
7  import wt.util.WTPropertyVetoException;
8
9  public class StandardTrainingService extends
10 StandardManager implements TrainingService{
11     private static final long serialVersionUID = 1L;
12     public static StandardTrainingService
13     newStandardTrainingService() throws WTException {
14         final StandardTrainingService service = new
15         StandardTrainingService();
16         service.initialize();
17         return service;
18     }
19     @Override
20     public WTPart createFatherOf(String name) throws
21     WTException {
22         final WTPart part = WTPart.newWTPart();
23         try {
24             part.setName("I am your father, "+name);
25         } catch (WTPropertyVetoException wtpve) {
26             throw new WTException(wtpve);
27         }
28         return (WTPart) PersistenceHelper.manager.store(
29                 part);
30     }
31 }
```

Task 4: Create a main method that will call the Helper class.

1. Use the following code to create a part

```
1  package com.ptc.serviceAcademy;
2
3  import wt.util.WTException;
4  import com.ptc.serviceAcademy.training.TrainingHelper;
5  public class CreatePartFromHelper {
6
7      public static void main(String[] args) {
8          try {
9              TrainingHelper.service.createFatherOf("Luke");
10         } catch (WTException e) {
11             e.printStackTrace();
12         }
13     }
14 }
```

2. Compile all codes using "javac" command as shown in image below. Copy all class files created in workspace into <WT_Home>/src/serviceAcademy/*. Restart the Method Server

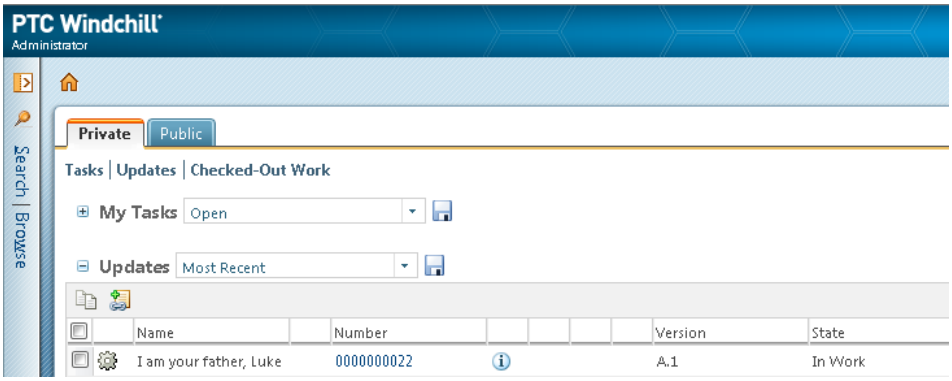
```
D:\ptc\Windchill_11.0\Windchill>windchill start
2017-05-12 09:41:42,538 INFO  [main] wt.manager.ServerLauncher - Starting ServerManager

D:\ptc\Windchill_11.0\Windchill>cd src

D:\ptc\Windchill_11.0\Windchill\src>javac D:\Workspace\GSProject\src\com\ptc\serviceAcademy\training\TrainingHelper.java

D:\ptc\Windchill_11.0\Windchill\src>javac D:\Workspace\GSProject\src\com\ptc\serviceAcademy\training\StandardTrainingService.java
```

3. Run the main method in Windchill shell using following command
   `Windchill com.ptc.serviceAcademy.CreatePartFromHelper`
4. Login to Windchill to see if the part was created.
   Example:



Click Home icon

Completion Criteria

1. A part named “I am your father, Luke” was created at **Site > Folders**.

This completes the exercise.

Module 6

Extending the Windchill Object Model

Module Overview

Object Model extension can be accomplished nicely with the use of predefined Windchill annotations. In this module, we demonstrate how this is accomplished.

Objectives

After completing this module, you will be able to:

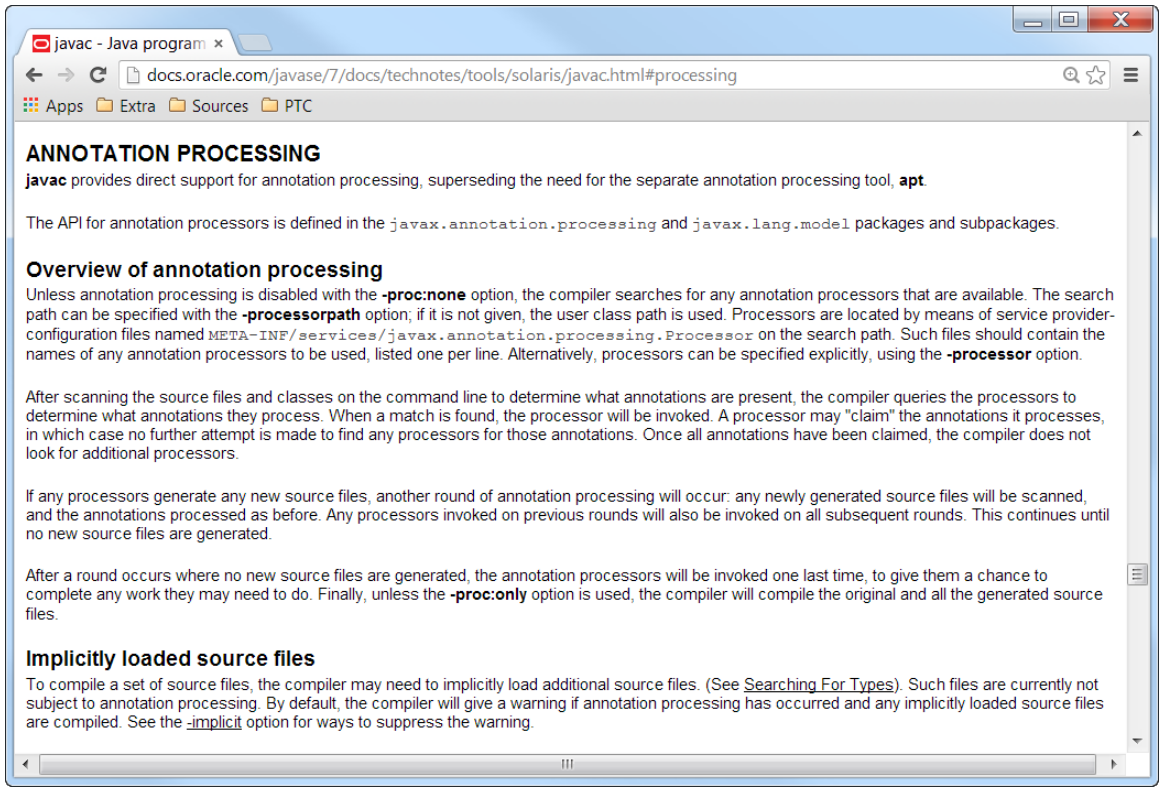
- Demonstrate the use of Annotations for creating additional code.
- Extend the Object Model

Annotation Processing

Annotation Processing builds additional code

Annotations can be used to generate repetitive code.

- Annotation Processors are compiled and Added to the Classpath
- When tags are included in the new source code, the annotation processor of the same name is automatically invoked
- The annotated file is the last element to be acted upon by the compiler.



http://docs.oracle.com/javase/7/docs/technotes/tools/solaris/javac.html#processing
javac provides direct support for annotation processing, superseding the need for the separate annotation processing tool, **apt**.
The API for annotation processors is defined in the javax.annotation.processing and javax.lang. model packages and subpackages.

Overview of annotation processing

Unless annotation processing is disabled with the **-proc:none** option, the compiler searches for any annotation processors that are available. The search path can be specified with the **-processorpathoption**; if it is not given, the user class path is used. Processors are located by means of service provider-configuration files named META-INF/services/javax.annotation.processing.Processor on the search path. Such files should contain the names of any annotation processors to be used, listed one per line. Alternatively, processors can be specified explicitly, using the **-processor** option.

After scanning the source files and classes on the command line to determine what annotations are present, the compiler queries the processors to determine what annotations they process. When a match is found, the processor will be invoked. A processor may "claim" the annotations it processes, in which case no further attempt is made to find any processors for those annotations. Once all annotations have been claimed, the compiler does not look for additional processors.

If any processors generate any new source files, another round of annotation processing will occur: any newly generated source files will be scanned, and the annotations processed as before. Any processors invoked on previous rounds will also be invoked on all subsequent rounds. This continues until no new source files are generated.

After a round occurs where no new source files are generated, the annotation processors will be invoked one last time, to give them a chance to complete any work they may need to do. Finally,unless the **-proc:only** option is used, the compiler will compile the original and all the generated source files.

Implicitly loaded source files

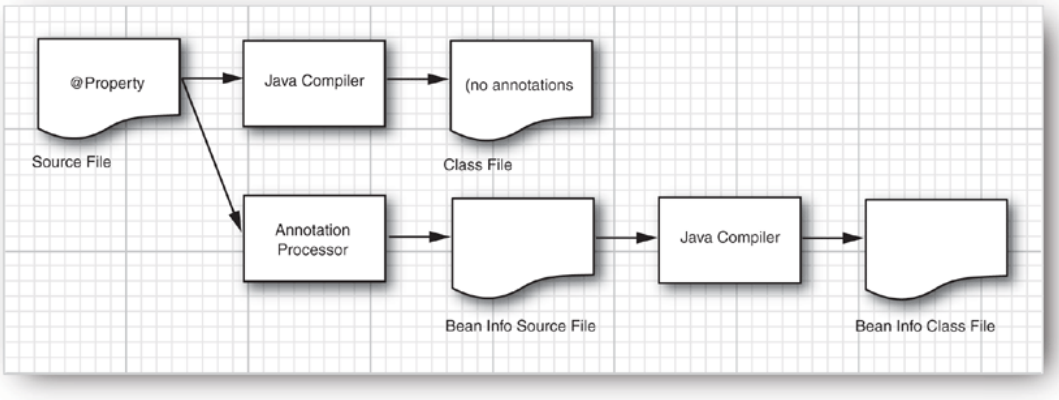
To compile a set of source files, the compiler may need to implicitly load additional source files. (See Searching For Types). Such files are currently not subject to annotation processing. By default, the compiler will give a warning if annotation processing has occurred and any implicitly loaded source files are compiled. See the -implicit option for ways to suppress the warning.

Order of Processing

Order is Important

The Processor needs to act on the processor before compilation

- The annotated file is the last element to be acted upon by the compiler.



Order of Processing

Compiler Options

javac at the Windchill shell

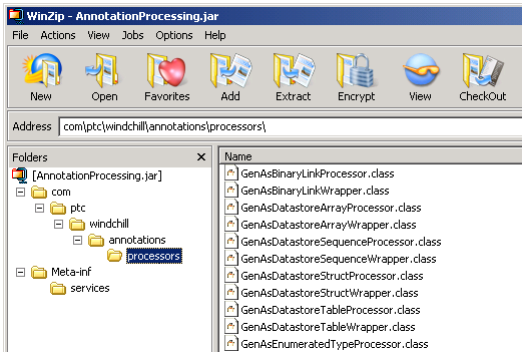
Annotations Processing in Java 8 is built-in to javac

- In order to invoke it, you must supply special arguments:

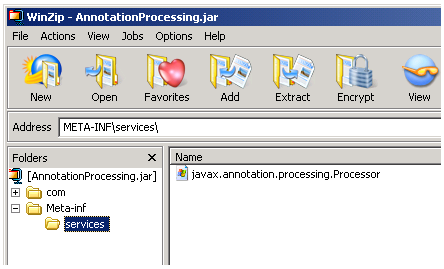
```
javac -processor
path.to.AnnotationProcessor
 AnnotatedJavaFile.java
```

Annotation Processors are bundled in AnnotationProcessing.jar

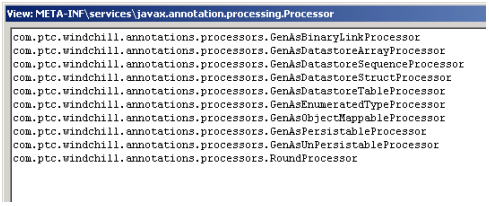
- InfoModeler is required to get AnnotationProcessing.jar
- Not included in the classpath of the Windchill shell.
- Processors are registered in Meta-inf/services



AnnotationProcessing.jar enables Object Model customization



Meta-inf/services



Registered Processors

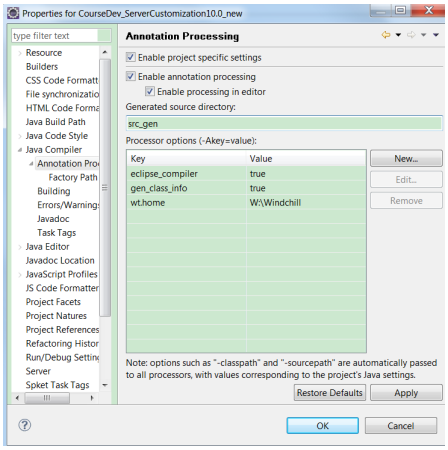
Compiler Options and Eclipse

Configuration in Eclipse is Intuitive

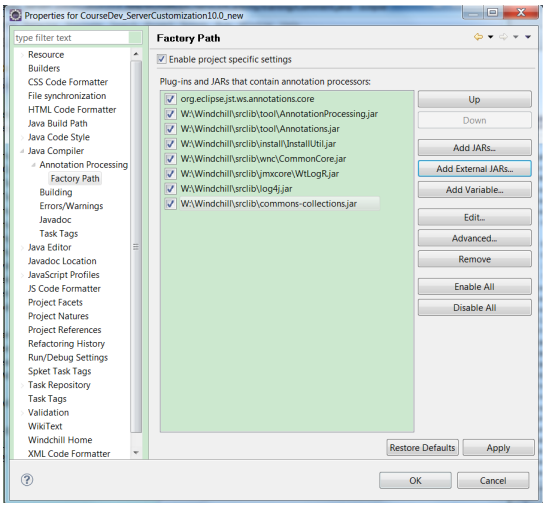
Eclipse can be configured easily.

Using Eclipse is intuitive:

- Eclipse's automatic compilations takes care of multiple rounds of compilation
- Files are simply generated without any effort apart from Save



Location of generated source



Annotation Processors used

Annotation Processing Example

Here is an example outside of Windchill and Eclipse

Consider a use case where a new Java Object needs to be created based on an existing Object. (this example was taken from the web)

- ChartBean.java is Graphical detail for a Chart.
- ChartBeanBeanInfo.java is the getters and setters for the ChartBean

Instead of creating the two objects manually:

1. Create ChartBean.java
2. Create an Annotation that identifies objects that need getters and setters — Property.java
3. Create a processor to read the annotations and create the new file. — BeanInfoAnnotationProcessor.java
4. Annotate the ChartBean.java
5. Compile the files in the proper order to create ChartBeanBeanInfo.java

Creating the Annotation Processor will be more difficult than coding ChartBeanBeanInfo.java manually

- But it can be reused for other beans in your application.

```
import java.awt.*;
import java.util.*;
import java.beans.*;
import java.io.*;
import com.ptc.serviceAcademy.annotations.*;

public class ChartBean extends Component
   implements Serializable
{  public void paint(Graphics g)
   {  if (values == null || values.length == 0) return;
      int i;
      double minValue = 0;
      double maxValue = 0;
      for (i = 0; i < values.length; i++)
      {  if (minValue > getValues(i)) minValue = getValues(i);
         if (maxValue < getValues(i)) maxValue = getValues(i);
      }
      if (maxValue == minValue) return;

      Dimension d = getSize();
      int clientWidth = d.width;
      int clientHeight = d.height;
      int barWidth = clientWidth / values.length;

      g.setColor(inverse ? color : Color.white);
      g.fillRect(0, 0, clientWidth, clientHeight);
      g.setColor(Color.black);
```

CharBean.java is just something that draws a window

Annotation Processing Example(cont'd)

BeanInfoAnnotationProcessor.java is Complex

Property.java is a regular annotation

```
package com.ptc.serviceAcademy.annotations;

import java.lang.annotation.Documented;

@Documented
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.SOURCE)
public @interface Property {
    String editor() default "";
}
```

Property.java

BeanInfoAnnotationProcessor.java is like a Template Processor that attempts to create a Java File

```
package com.ptc.serviceAcademy.corejava;

import java.beans.*;
import java.io.*;
import java.util.*;
import javax.annotation.processing.*;
import javax.lang.model.*;
import javax.lang.model.element.*;
import javax.tools.*;
import javax.tools.Diagnostic.*;
import com.ptc.serviceAcademy.annotations.*;

 * @version 1.10 2011-12-20

@SupportedAnnotationTypes("com.ptc.serviceAcademy.annotations.Property")
@SupportedSourceVersion(SourceVersion.RELEASE_6)
public class BeanInfoAnnotationProcessor extends AbstractProcessor
{
   @Override
   public boolean process(Set<? extends TypeElement> annotations, RoundEnvironment roundEnv)
   {
      for (TypeElement t : annotations)
      {
         Map<String, Property> props = new LinkedHashMap<String, Property>();
         String beanClassName = null;
         for (Element e : roundEnv.getElementsAnnotatedWith(t))
         {
            String mname = e.getSimpleName().toString();
            String[] prefixes = { "get", "set", "is" };
```

BeanInfoAnnotationProcessor.java

Annotation Processing Example Order of Events

Compile the Files in Order

Compile the Property.java first

```
D:\ptc\Windchill\Windchill\src>javac com/ptc/serviceAcademy/annotations/Property
.java
```

Compiling Property.java

Compile the Annotation Processor next.

```
D:\ptc\Windchill\Windchill\src>javac com/ptc/serviceAcademy/corejava/BeanInfoAnn
otationProcessor.java
```

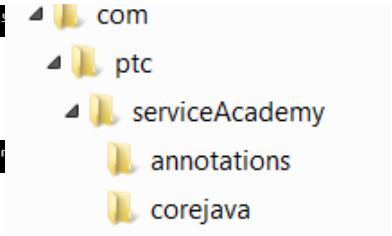
Compiling BeanInfoAnnotationProcessor.java

Invoke the Processor to create the ChartBeanBeanInfo.java as well as compiling ChartBean.java

Invoking the Annotation Processor during compilation

> XprintRounds was used to demonstrate the number of compilations — 3 in this case

Files are organized into hierarchy



Outcome is a Newly Generated Java File

```
package com.ptc.serviceAcademy.corejava;
public class ChartBeanBeanInfo extends java.beans.SimpleBeanInfo
{
   public java.beans.PropertyDescriptor[] getPropertyDescriptors()
   {
      try
      {
         java.beans.PropertyDescriptor titleDescriptor
            = new java.beans.PropertyDescriptor("title", ChartBean.class);
         return new java.beans.PropertyDescriptor[]
         {
            titleDescriptor
         };
      }
      catch (java.beans.IntrospectionException e)
      {
         e.printStackTrace();
         return null;
      }
   }
}
```

ChartBeanBeanInfo.java

Exercise 1: Create an Annotation Processor

Objectives

After successfully completing this exercise, you will be able to:

- Invoke an Annotation Processor using javac

Scenario

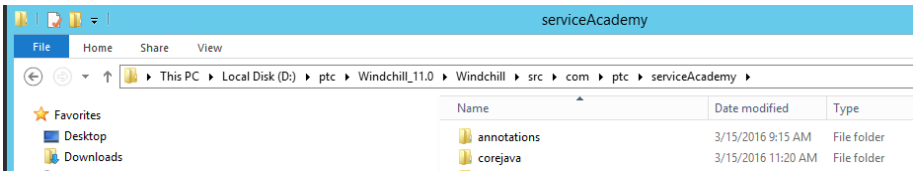
In this exercise, source files are supplied and the student is asked to compile these files such that a new file is created by the compiler. The purpose of this exercise is demonstrate how annotation processors are invoked and what they can do.

Task 1: Access the required files.

1. Copy files to the Windchill/src directory

   Code is provided in LabFiles.zip

   Result:



Files Properly copied to Windchill/src

Task 2: Compile Required Files

1. From the Windchill shell, compile Property.java using javac as described in the theory slides. [Ex: javac com/ptc/serviceAcademy/annotations/Property.java]
2. Copy Property.class to the codebase directory preserving the path
3. From the Windchill shell, compile BeanInfoAnnotationProcessor.java as described in the Annotation Theory before using javac
4. Copy BeanInfoAnnotationProcessor.class to the codebase directory.

Task 3: Invoke the Annotation Processor in Mustang (Java 1.8)

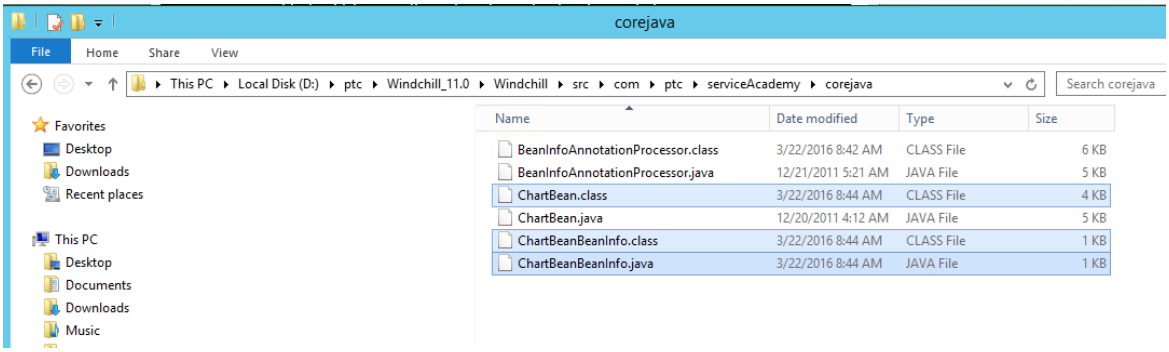
1. Invoke the annotation processor

   Example:

```
javac
-processor
 com.ptc.serviceAcademy.corejava.BeanInfoAnnotationProcessor
com/ptc/serviceAcademy/corejava/ChartBean.java
-XprintRounds
```

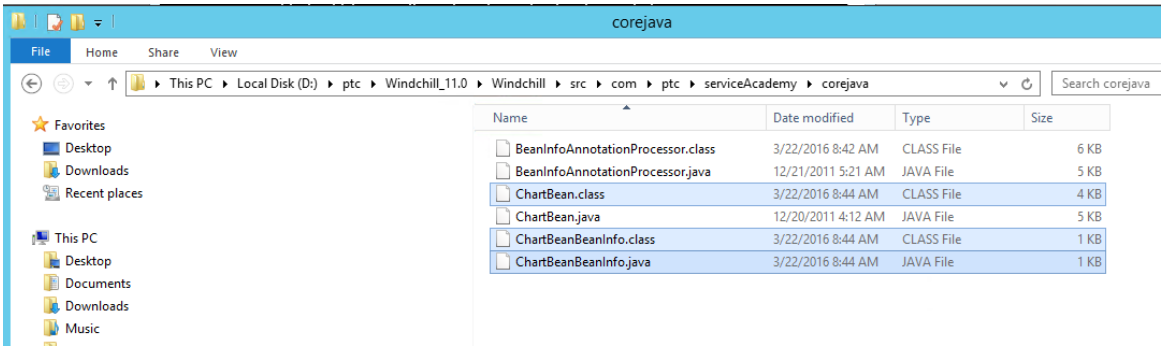
2. Review the files that were created by this command.

   Result:



Completion Criteria

1. Three files should be generated in the folder as follows:



This completes the exercise.

Compilation and Generation in Windchill

Windchill creates _Object files

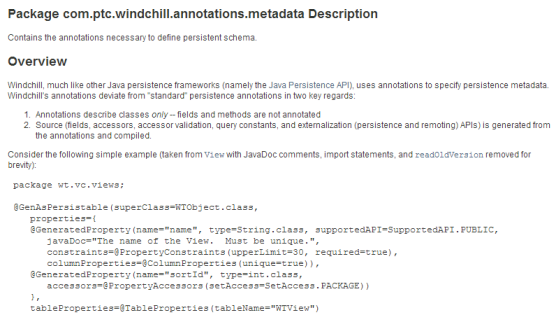
Annotation processing in the compiler is documented in the Javadoc: com.ptc.windchill.annotations.metadata

Windchill annotation processors generate an "_" source file.

- Annotation processing is done within the compiler itself and the compiler will automatically compile all source files generated by these annotation processors.

Compilation will, typically, result in three distinct annotation processor rounds:

1. Generate the "_" classes for:
   - Top-level annotated files
   - Foreign key link classes (annotated with GenAsBinaryLink) for each GeneratedForeignKey
2. Generate the "_" classes for:
   - (any) foreign key link classes produced by round one
3. Report that Windchill annotation processing is complete.



Package com.ptc.windchill.annotations.metadata Description

Contains the annotations necessary to define persistent schema.

Overview

Windchill, much like other Java persistence frameworks (namely the Java Persistence API), uses annotations to specify persistence metadata. Windchill's annotations deviate from "standard" persistence annotations in two key regards:

1. Annotations describe classes only -- fields and methods are not annotated.
2. Source (fields, accessors, accessor validation, query constants, and externalization (persistence and remoting) APIs) is generated from the annotations and compiled.

Consider the following simple example (taken from `View` with JavaDoc comments, import statements, and `readOldVersion` removed for brevity):

```
package wt.vc.views;

@GenAsPersistable(superClass=WTObject.class,
   properties={
   @GeneratedProperty(name="name", type=String.class, supportedAPI=SupportedAPI.PUBLIC,
      javaDoc="The name of the View.  Must be unique.",
      constraints=@PropertyConstraints(upperLimit=30, required=true),
      columnProperties=@ColumnProperties(unique=true)),
   @GeneratedProperty(name="sortId", type=int.class,
      accessors=@PropertyAccessors(setAccess=SetAccess.PACKAGE))
   },
   tableProperties=@TableProperties(tableName="WTView")
```

Javadoc

Annotations Processors in Windchill

Annotation Processors and Annotations are stored in a common location

http://ah-grok.ptcnet.ptc.com/xref/x-24/wcmod/modules/AnnotationProcessing/src/com/ptc/windchill/annotations/processors/AbstractGenAsPersistableWrapper.java

One example is the GenAsPersistable processor that is used for extending the Persistable Objects

```
/* bcwti
 *
 * Copyright (c) 2010 Parametric Technology Corporation (PTC). All Rights Reserved.
 *
 * This software is the confidential and proprietary information of PTC
 * and is subject to the terms of a software license agreement. You shall
 * not disclose such confidential information and shall use it only in accordance
 * with the terms of the license agreement.
 *
 * ecwti
 */
package com.ptc.windchill.annotations.processors;

import javax.annotation.processing.SupportedAnnotationTypes;
import javax.annotation.processing.SupportedSourceVersion;
import javax.lang.model.SourceVersion;

@SupportedAnnotationTypes("com.ptc.windchill.annotations.metadata.GenAsPersistable")
@SupportedSourceVersion(SourceVersion.RELEASE_7)
public class GenAsPersistableProcessor extends AbstractGenAsPersistableProcessor {
   @Override GenAsPersistableWrapper getWrapper() { return wrappers.paw; }
}

```

Use of @GenAsPersistable in Windchill

Annotation Processors are used instead of Rational Rose to simplify object extensions

Out-of-the-box code demonstrates the usage of the annotations to generate classes

- View extends _View
- _View Is generated by the @GenAsPersistable annotation

```
@GenAsPersistable(superClass=WTObject.class, versions={-4333025332245130711},
   properties={
   @GeneratedProperty(name="name", type=String.class, supportedAPI=SupportedAPI.PUBLIC,
      javaDoc="The name of the View.  Must be unique.",
      constraints=@PropertyConstraints(upperLimit=30, required=true),
      columnProperties=@ColumnProperties(unique=true)),
   @GeneratedProperty(name="sortId", type=int.class,
      accessors=@PropertyAccessors(setAccess=SetAccess.PACKAGE))
   },
   tableProperties=@TableProperties(tableName="WTView")
)
public final class View extends _View {
   static final long serialVersionUID = 1;
   public static final String VIEW_SORT_COLUMN = SORT_ID;

   public static View newView( String viewName )
            throws WTException {
      View instance = new View();
      instance.initialize( viewName );
      return instance;
   }

   protected void initialize( String viewName )
            throws WTException {
      try {
         setName(viewName);
      }
      catch (WTPropertyVetoException wtpve) {
         throw new ViewException(RESOURCE, viewsResource.INVALID_VIEW_NAME,
               new Object[]{viewName});
      }
   }

   @Override
   public String getIdentity() {
      return getName();
   }

   boolean readVersion_4333025332245130711( ObjectInput input, long readSerialVersionUID, boolean superDone )
            throws IOException, ClassNotFoundException {
      if ( !superDone )                                    // if not doing backward compatibility
         super_readExternal_View( input );                    // handle super class

      name = (String)input.readObject();
      sortId = input.readInt();

      return true;
   }
}
```

View Inheriting from WTObject

Using Annotations in Windchill

Annotations help focus effort on Custom Code

@GenAs annotations drive annotation processing

- Purely user-edited Java files are annotated to describe schema (e.g. WTPart.java)
- Purely compiler-generated "_"parent files provide implementation (e.g. _WTPart.java)
  - Implements interfaces on your behalf
  - Generates constants, fields, accessors, accessor validation, externalization, etc.
  - Foreign key link classes
  - ClassInfo files, too!
- Generation (and compilation of generated files) all during regular javac compilation
- You specify metadata for persistence using annotations (in your IDE instead of via Rose/UML)
- The compiler generates the implementation, much as before, into "_" files

Primary Annotations

Customizations will start with these Processors

- **GenAsUnPersistable** non-persistent interfaces implemented by persistent classes and Evolvable classes (usage should be rare)
- **GenAsObjectMappable** persistent objects stored as columns (aka “cookies”)
- **GenAsPersistable** persistent objects stored as tables (aka “persistables”)
- **GenAsBinaryLink** specialize persistables associating a role A/B persistables
- **GenAsEnumeratedType** extensions of wt.fc.EnumeratedType
- **GenAsPrimitiveType** classes that can be reduced to a simple java “primitive” type
- **GenAsDatastoreSequence** database sequences
- **GenAsDatastoreStruct** database structures
- **GenAsDatastoreArray** arrays of database column types, including structs
- **GenAsDatastoreTable** non-persistable persistables

Other Annotations

Secondary and Tertiary Annotations are Properties of other Annotations

Secondary annotations (properties of primary annotations)

- **GeneratedProperty** field consisting of Java “primitives” and cookies
- **GeneratedForeignKey** reference to persistable stored locally in this persistable
- **GeneratedRole** role A/B of a binary link
- **DerivedProperty** convenience accessors to above
- **TableProperties** table name, composite indexes, and composite unique indexes
- **IconProperties** standard/open icon (this may be obsolete)

Tertiary annotations (properties of secondary annotations)

- **PropertyAccessors** getter/setter access/exceptions
- **PropertyConstraints** string case, upper/lower limits, required, changeability
- **ColumnProperties** column name/type, indexing, uniqueness, persistence
- **ForeignKeyRole** target persistable’s role (master in master/iteration)
- **MyRole** my role as the target sees me (iteration in master/iteration)

A Simple Extension of WTObject

@GenAsPersistable example

You must extend the "_" class, which implements most of your logic

- This is true even for interfaces
- Use superClass for true parent
- Use interfaces for any interfaces you want to implement

Certain things now need to be coded:

- serialVersionUID must be assigned to "1"
  - Windchill handles externalization for you
- Factory methods hand-coded

Overview

- Annotated files are handled by annotation processors during compilation
- These annotation processors produce "_" files in the src_gen directory
  - Everything needed by persistence is generated into these files
- Generated files are then recognized and compiled in the second round of processing.
- Since this is all standard Java, Eclipse does this behind the scenes.

```
@GenAsPersistable(superClass=WTObject.class, interfaces={WTContained.class},
    properties={
    @GeneratedProperty(name="name", type=String.class,
        constraints=@PropertyConstraints(required=true))
    })
public class SimplyNamed extends _SimplyNamed {
    static final long serialVersionUID = 1;

    public static SimplyNamed newSimplyNamed() throws WTException {
        final SimplyNamed instance = new SimplyNamed();
        instance.initialize();
        return instance;
    }

    @Override
    public void checkAttributes() throws InvalidAttributeException {
        super.checkAttributes();
        try {
            nameValidate(getName());
        } catch (WTPropertyVetoException wtpve) {
            throw new InvalidAttributeException(wtpve);
        }
    }
}
```

Code

```
7  public class TestSimplyNamed {
8      public static void main(String[] args) {
9          try {
10             SimplyNamed sn = SimplyNamed.newSimplyNamed();
11             sn.setName("Hello from @GenAsPersistable!");
12             sn.checkAttributes();
13             PersistenceHelper.manager.store(sn);
14         } catch (Exception e) {
15             e.printStackTrace();
16         }
17     }
18 }
19
```

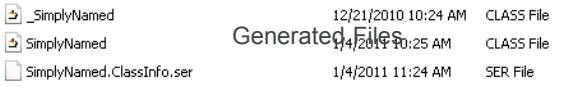
Test SimplyNamed

Completing the Extension

Before you restart the Method Server

By Compiling this class, you will get the following:

- Additional Source
  - _SimplyNamed.java (Round 1)
- Compiled Classes
  - SimplyNamed.class (Round 1)
  - _SimplyNamed.class (Round 2)
- Serialization Files
  - SimplyNamed.ClassInfo.ser
- Updated Windchill Registry
  - descendentRegistry.properties
  - associationRegistry.properties
  - modelRegistry.properties



Generated Files

Database Tables need to be generated using tools.xml

- ant -f bin/tools.xml class -Dclass.includes= com/ptc/serviceAcademy/training/*
- ant -f bin/tools.xml sql_script -Dgen.input= com.ptc.serviceAcademy.training.*

Help is available for tools.xml — ant -f bin/tools.xml <target>.help

Execute the following scripts in Windchill shell:

- cd %wt_home%/db/sql3
- sqlplus installpds/installpds
- @Make_pkg_sql3_Table.sql
- @Make_pkg_sql3_Index.sql

PL/SQL Developer

A Link

Coding a Link is Similar

A different Annotation is used to specify the new object as a descendant of ObjectToObjectLink but the characteristics are similar:

- Some hard coding required — roleA, roleB
- superClass is denoted in the Annotation
- serialVersionUID is specified manually
- Extends "_" class of the same name — and hence the code is short.

```
@GenAsBinaryLink(superClass=ObjectToObjectLink.class,
    roleA=@GeneratedRole(name="father", type=SimplyNamed.class),
    roleB=@GeneratedRole(name="son", type=SimplyNamed.class))
public class FatherSonLink extends _FatherSonLink {
    static final long serialVersionUID = 1;

    public static FatherSonLink newFatherSonLink(final SimplyNamed father, final SimplyNamed son) throws WTException {
        final FatherSonLink instance = new FatherSonLink();
        instance.initialize(father, son);
        return instance;
    }
}
```

FatherSonLink.java

Test FatherSonLink

Exercise 2: Create a Simple Object Extension and Link

Objectives

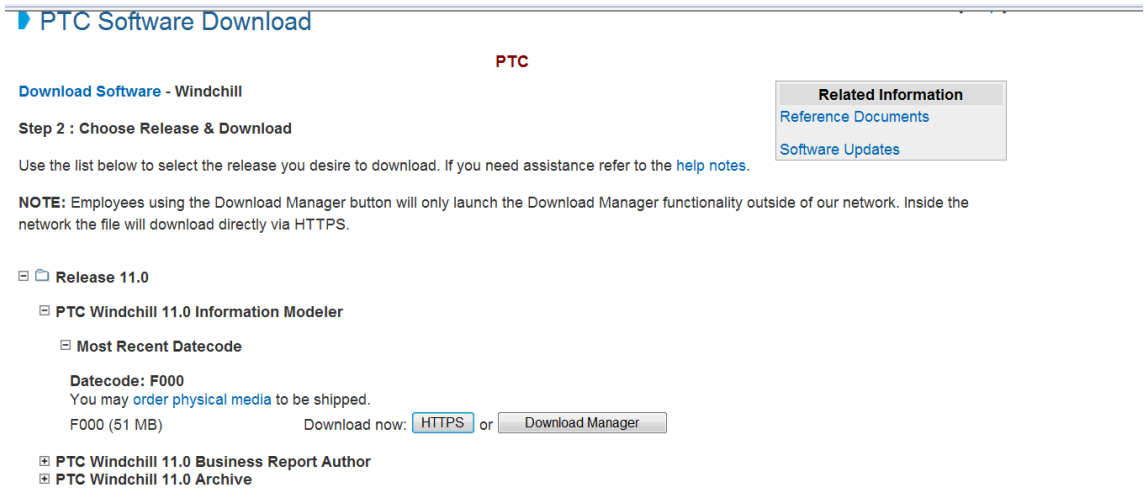
After successfully completing this exercise, you will be able to:

- Create Simple Object Extensions in Windchill
- Create Links in Windchill
- Use out-of-the-box annotations and annotation processors.

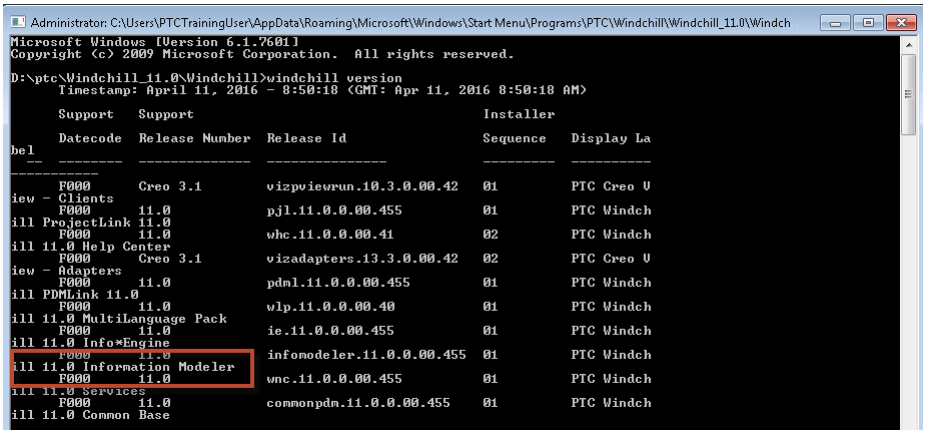
Prerequisites

Update 11.0 F000 using PSI and add Information Modeller.

1. Download the PSI from and Information Modeller from
2. As shown in figure get the recent datecode MED-2101-CD-110_F000



3. Open PSI setup and choose to update the existing windchill Installation. Update the windchill components. Mention the path of cdimage where we downloaded the supported Info Modeller and update the installation.
4. Verify the Windchill Version by Open Windchill shell- type windchill —version



Scenario

In this exercise, we create simple custom objects using annotations.

Sample code is provided in LabFiles.zip.

Extract Oracle SQL Developer from the Student LabFiles.zip and run the application.

Task 1: Create and test an extension of WTObject

1. Unzip and copy the SimplyNamed.java from Q drive to WT_HOME/src/com/ptc/serviceAcademy/training folder and TestSimplyNamed.java to WT_HOME/src/com/ptc/serviceAcademy.
2. Open a Windchill shell and execute the following command to generate class.

   Open windchill shell and run below commands from <wt_home/bin>. The class files get created in codebase\com\ptc\serviceAcademy\training folders

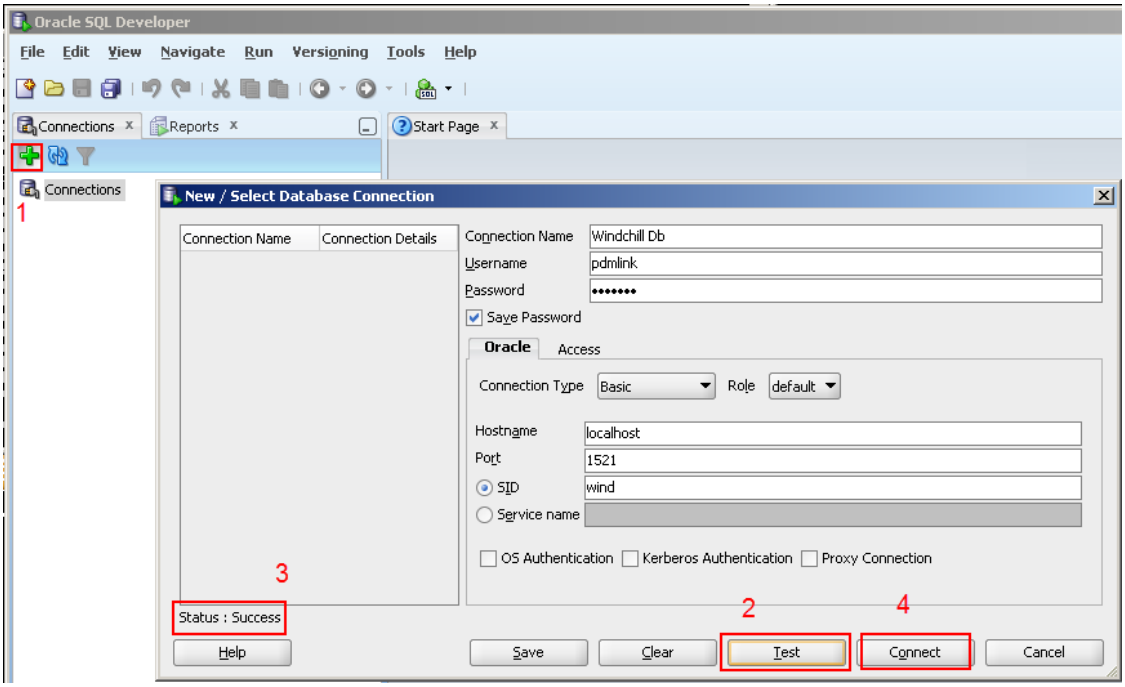
   ```
   ant -f tools.xml class -Dclass.includes=com/ptc/serviceAcademy/training/*
   ant -f tools.xml class -Dclass.includes=com/ptc/serviceAcademy/*
   ```
3. Execute the following command to build the database tables.
   ```
   ant -f tools.xml sql_script -Dgen.input=com.ptc.serviceAcademy.training.*
   ```
4. Switch to the db/sql3 sub directory
5. Execute: sqlplus

   installpds/installpds is the username/password for your Oracle.
6. Execute: @Make_pkg_sql3_Table.sql
7. Execute: @Make_pkg_sql3_Index.sql

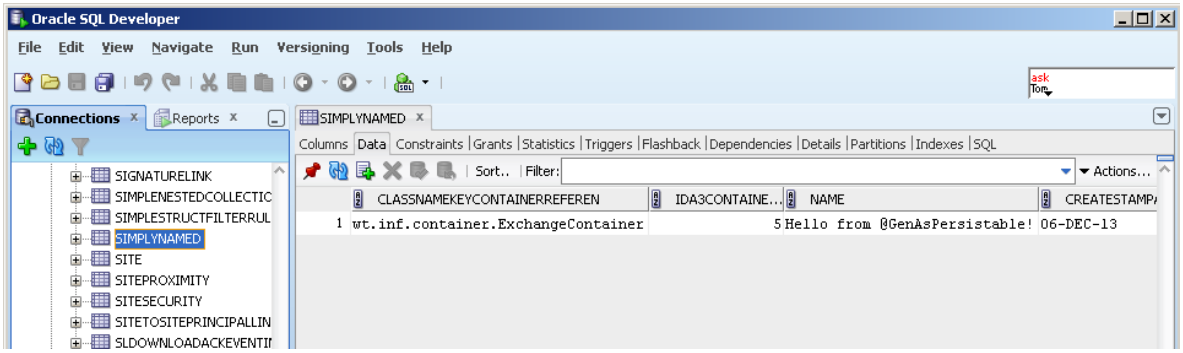
   If the database used by Windchill is sqlServer, the scripts are as follows:
   - <WT_HOME>/db/sqlServer/Make_pkg_sqlServer_Table.sql
   - <WT_HOME>/db/sqlServer/Make_pkg_sqlServer_Index.sql
8. Restart the method server.
9. Run the Test code as an example:
   Example:
   ```
   windchill com.ptc.serviceAcademy.TestSimplyNamed
   ```

   If running windchill command does not create the table in SQL developer Please execute the Steps from Step 4 using the appropriate SQL.
10. Open the Oracle SQL Developer and Check that data was created in the table.

    Use WIND as database and installpds/installpds as login/password for connection.



Result:



Task 2: Create and test an Link Object between of the extensions created in the previous Task.

1. Unzip and copy FatherSonLink.java to WT_HOME/src/com/ptc/serviceAcademy/training and TestFatherSonLink.java to WT_HOME/src/com/ptc/serviceAcademy directory.
2. Open a Windchill shell and execute the following command to generate class.

   Open windchill shell and run below commands from <wt_home/bin>. The class files get created in codebase\com\ptc\serviceAcademy\training folders

```
ant -f tools.xml class -Dclass.includes=com/ptc/serviceAcademy/training/*
ant -f tools.xml class -Dclass.includes=com/ptc/serviceAcademy/*
```

3. Execute the following command to build the database tables.

```
ant -f tools.xml sql_script -Dgen.input=com.ptc.serviceAcademy.training.*
```

4. Switch to the db/sq3l sub directory
5. Execute: sqlplus

> installpds/installpds is the username/password for your Oracle.

6. Execute: @Make_pkg_sql3_Table.sql
7. Execute: @Make_pkg_sql3_Index.sql

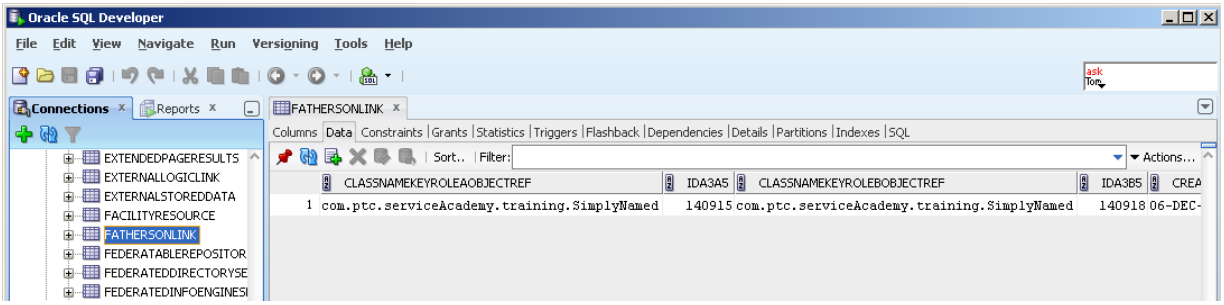
> If the database used by Windchill is sqlServer, the scripts are as follows:
> - <WT_HOME>/db/sqlServer/Make_pkg_sqlServer_Table.sql
> - <WT_HOME>/db/sqlServer/Make_pkg_sqlServer_Index.sql

8. Restart the method server.
9. Run the Test code as example:
   Example:

```
windchill com.ptc.serviceAcademy.TestFatherSonLink
```

> If running windchill command does not create the table in SQL developer Please execute the Steps from Step 4 using the appropriate SQL.

10. Open the Oracle SQL Developer and Check that data was created in the table.
    Result:



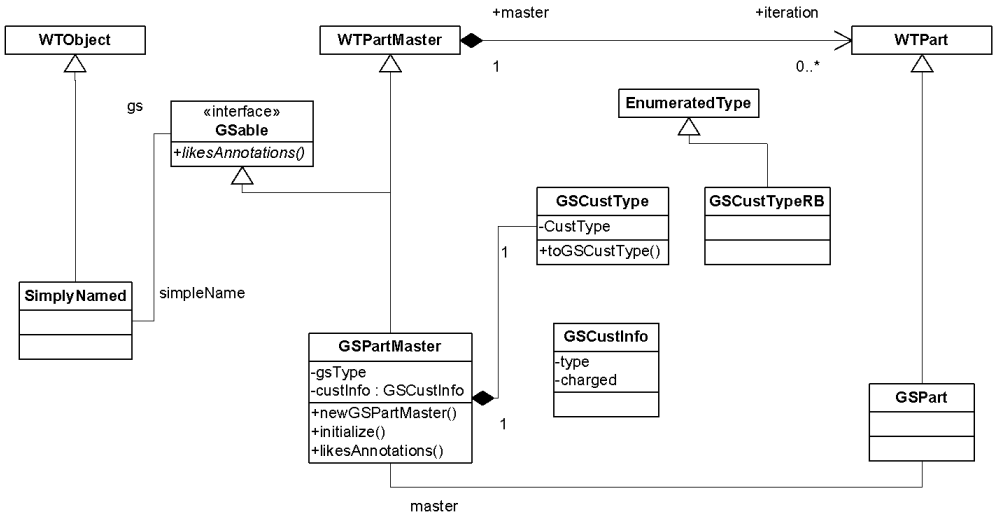
Completion Criteria

1. Data was created in the table SIMPLYNAMED.
2. Data was created in the table FATHERSONLINK.

This completes the exercise.

Crescendo Object Model

Fictitious model has a bit of everything



Object Model

GSable interface

- Has foreign key link to SimplyNamed
- Requires implementer to implement "likesAnnotations"

GSCustType — enumerated type

- Creating a bundle the old way still works
- ant -f bin/tools.xml bundle -Dbundle.input= com.ptc.training.*
- **ResourceBuild**

GSCustInfo — cookie

- Demonstrates primitives, initial values
- Dubiously follows factory pattern

GSPartMaster

- Implements GSAble, including "likesAnnotations"
- Aggregates GSCustType (note initial value is assigned in initialize)

GSPart

- Constrains master/iteration foreign key link

Crescendo — GSable

GSable

GSable:

```
@GenAsPersistable(
    foreignKeys={
    @GeneratedForeignKey(
        foreignKeyRole=@ForeignKeyRole(name="simpleName", type=SimplyNamed.class, autoNavigate=true,
            constraints=@PropertyConstraints(required=true)),
        myRole=@MyRole(name="gs")
        )
    })
public interface GSable extends _GSable {
    boolean likesAnnotations();
}
```

properties/foreignKeys: append to avoid column renaming

GSCustType:

```
@GenAsEnumeratedType
public class GSCustType extends _GSCustType {
    public static final GSCustType X = toGSCustType("x");
    public static final GSCustType Y = toGSCustType("y");
    public static final GSCustType Z = toGSCustType("z");
}
```

GSCustTypeRB:

```
x.value=x
x.shortDescription=x
x.order=10

y.value=y
y.shortDescription=y
y.order=20

z.value=z
z.shortDescription=z
z.order=30
z.defaultValue=true
```

yes, this is manual

Crescendo — GSCustInfo

GSCustInfo

```
@GenAsObjectMappable(
    properties={
    @GeneratedProperty(name="type", type=GSCustType.class, initialValue="GSCustType.X"),
    @GeneratedProperty(name="charged", type=boolean.class, initialValue="false")
    })
public class GSCustInfo extends _GSCustInfo {
    static final long serialVersionUID = 1;

    public static GSCustInfo newGSCustInfo() throws WTException {
        final GSCustInfo instance = new GSCustInfo();
        instance.initialize();
        return instance;
    }
    protected void initialize() throws WTException { }
}
```

GSCustInfo Module

Crescendo — GSPartMaster

GSPartMaster

```
@GenAsPersistable(superClass=WTPartMaster.class, interfaces={GSable.class},
    properties={
    @GeneratedProperty(name="custInfo", type=GSCustInfo.class)
    },
    derivedProperties={
    @DerivedProperty(name="gsType", derivedFrom="custInfo.type")
    })
public class GSPartMaster extends _GSPartMaster {
    static final long serialVersionUID = 1;

    public static GSPartMaster newGSPartMaster() throws Exception {
        final GSPartMaster instance = new GSPartMaster();
        instance.initialize();
        return instance;
    }
    @Override
    protected void initialize() throws WTException {
        super.initialize();
        custInfo = GSCustInfo.newGSCustInfo();
    }

    @Override
    public boolean likesAnnotations() {
        return getSimpleName() != null && getSimpleName().getName().equals("I like annotations!");
    }
}
```

derivedProperties: “.” for cookie traversal, “>” for persistable traversal; You can let the IDE “implement” the interface method for you

Crescendo — GSPart

GSPart

```
@GenAsPersistable(superClass=WTPart.class,
    foreignKeys={
    @GeneratedForeignKey(
        foreignKeyRole=@ForeignKeyRole(name="master", type=GSPartMaster.class))
    })
public class GSPart extends _GSPart {
    static final long serialVersionUID = 1;

    public static GSPart newGSPart() throws Exception {
        final GSPart instance = new GSPart();
        instance.initialize();
        return instance;
    }
}
```

GSPart

Crescendo Test

TestGSPart

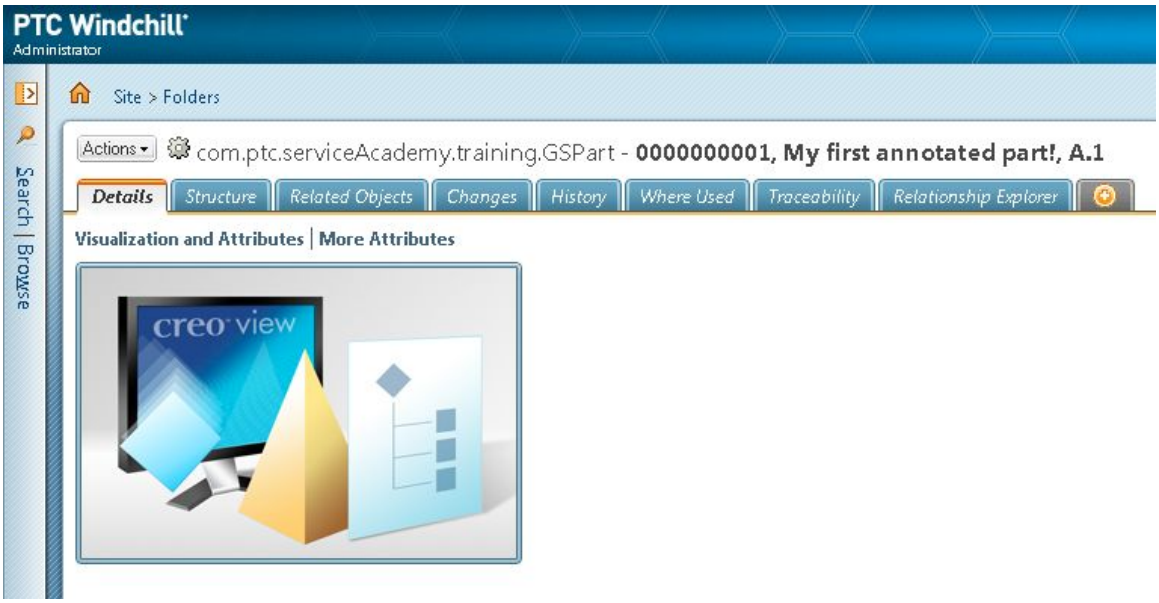
1. Use the following class test the GSPart

```
public static void main(String[] args) {
    try {
        SimplyNamed sn = SimplyNamed.newSimplyNamed();
        sn.setName("I like annotations!");
        sn = (SimplyNamed) PersistenceHelper.manager.store(sn);
        GSPart gsp = GSPart.newGSPart();
        gsp.setName("My first annotated part!");
        GSPartMaster gspm = (GSPartMaster) gsp.getMaster();
        gspm.setSimpleName(sn);
        gsp = (GSPart) PersistenceHelper.manager.store(gsp);
        gspm.likesAnnotations();
        sn.setName("I hate annotations.");
        sn = (SimplyNamed) PersistenceHelper.manager.modify(sn);
        gsp = (GSPart) PersistenceHelper.manager.refresh(gsp);
        gspm.likesAnnotations();
        System.out.println("==========GSPart: '" + gsp.getName()
                + "' has been created, its number is " + gsp.getNumber());
    } catch (WTException e) {
        e.printStackTrace();
    } catch (WTPropertyVetoException e) {
        e.printStackTrace();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

TestGSPart.java

Crescendo Results

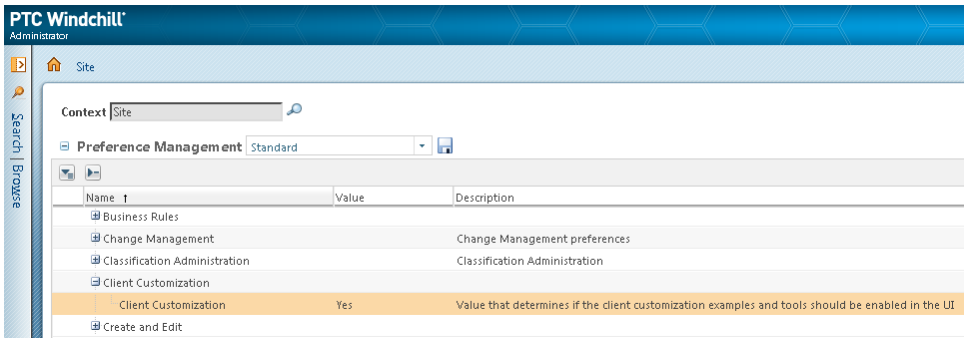
After Creating the Object ...



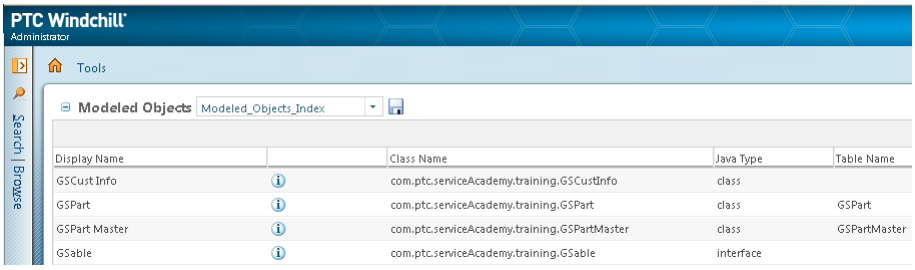
Crescendo Object Model in Windchill

Object Model can be Viewed in Windchill

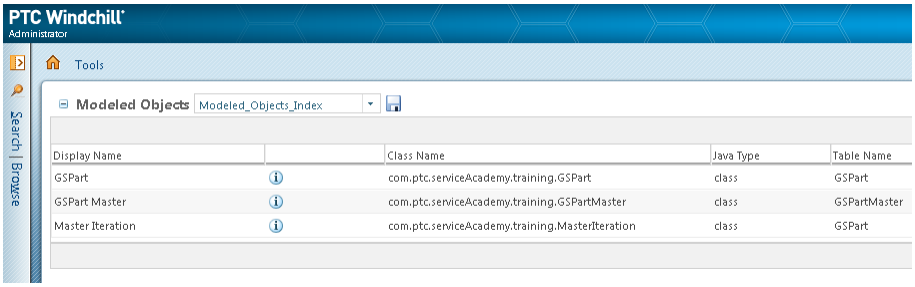
- Site>>Utilities>>Preference Manager>>Client Customization



Set Client Customization=Yes



List of Modeled Objects

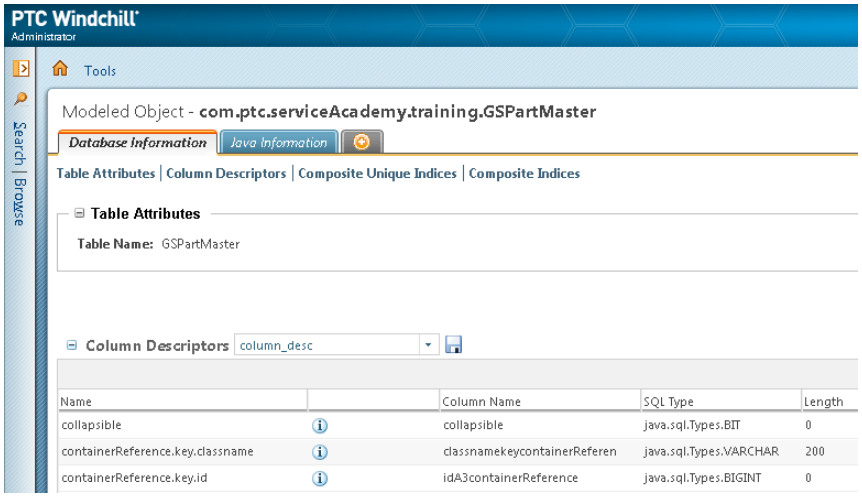


Searching for GSPart in Modeled Objects

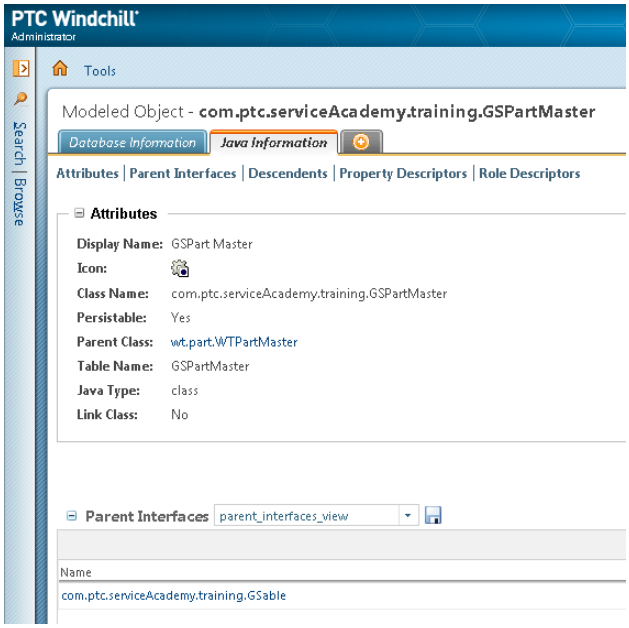
This is on the Customization Tools Tab

GSPartMaster in Windchill

GSPartMaster



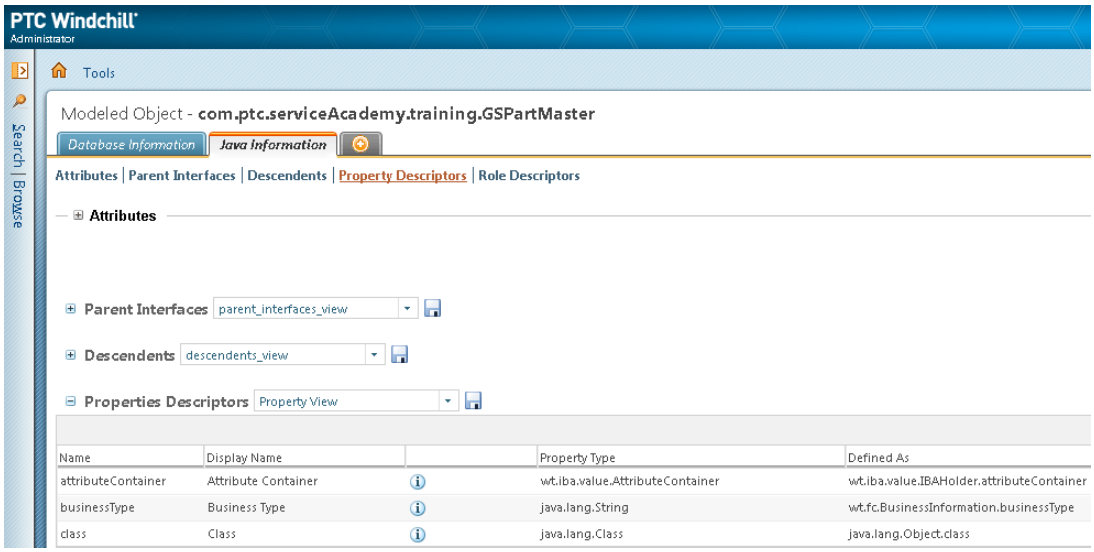
Results of GSPartMaster



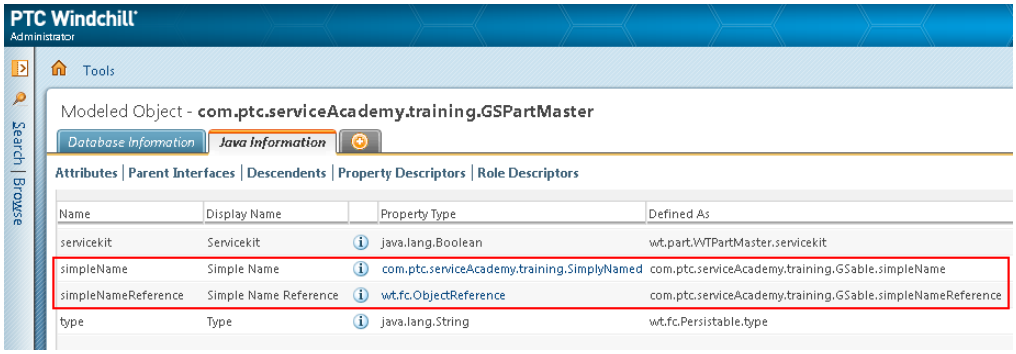
Results of GSPartMaster

GSPartMaster in Windchill

Cookie/Interface Demonstration (cont)



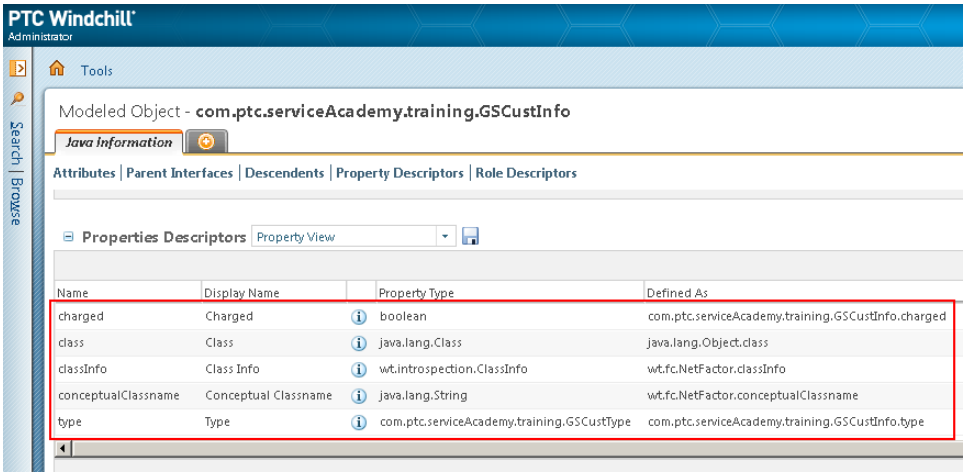
GSPartMaster

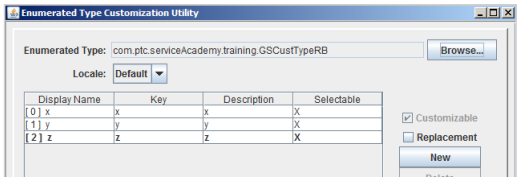


More GSPartMaster

GSCustInfo in Windchill

GSCustInfo





GSCustTypeRB

Reversing Out Customization

How do you uninstall a customization?

Delete something “from the model”?

- Registries are automatically updated when compiling annotated classes
- If you delete a class from the model, you’ll need to manually delete entries from:
  - associationRegistry.properties
  - descendentRegistry.properties
  - modelRegistry.properties

Errors in Round 1 Compilation

Round 1 will produce Errors

If you compile at the command line (or using BIF), you will see errors.

These errors can be ignored.

- They indicate that the "_" classes were missing in Round 1. — And they were since the source code was generated in this Round.
- Only errors after the last Round are true.

Exercise 3: Implement Crescendo

Objectives

After successfully completing this exercise, you will be able to:

- Deploy an advanced Windchill customization
- Create real objects in Windchill.

Scenario

In this exercise, you will implement a ready made Windchill customization and review the elements of that customization

Source code for this example is provided in LabFiles.zip

Task 1: Copy and install the code.

1. Unzip and copy following files from Q drive to WT_HOME/src/com/ptc/serviceAcademy/training directory:
   - GSable.java
   - GSCustInfo.java
   - GSCustType.java
   - GSPart.java
   - GSPartMaster.java
   - GSTestPart.java
2. Add the resource bundle (GSCustTypeRB.rbInfo) to src/com/ptc/serviceAcademy/training
3. Open a Windchill shell and perform a ResourceBuild
   Example:

   ResourceBuild com.ptc.serviceAcademy.training.GSCustTypeRB

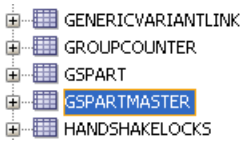
Task 2: Create the Database tables.

1. Run the command to create the sql scripts.
   Example:
   - ant -f bin/tools.xml class -Dclass.includes=com/ptc/serviceAcademy/training/*
   - ant -f bin/tools.xml sql_script -Dgen.input=com.ptc.serviceAcademy.training.*

   The above can also be run from <WT_HOME/bin> and remove bin from command
2. Switch to the db/sql3 directory
3. Execute: sqlplus installpds/installpds

   installpds/installpds is the username/password for your Oracle.
4. Execute: @Make_pkg_sql3_Table.sql
5. Execute: @Make_pkg_sql3_Index.sql

   If the database used by Windchill is sqlServer, the scripts are as follows:
   - <WT_HOME>/db/sqlServer/Make_pkg_sqlServer_Table.sql
   - <WT_HOME>/db/sqlServer/Make_pkg_sqlServer_Index.sql
6. Open the Oracle SQL Developer and Review the tables.
   Result:



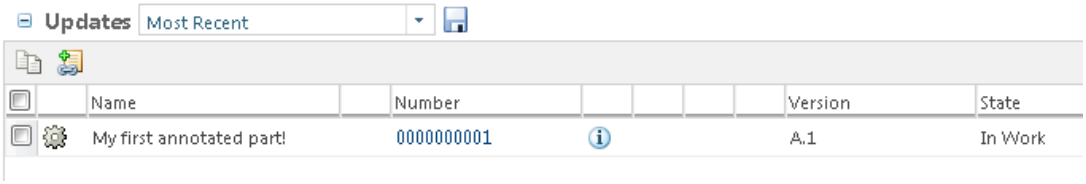
Task 3: Run the Test code.

1. Restart the method server.
2. When the method server starts, run the following command in your Windchill shell on Windchill **Server side**
   `windchill com.ptc.serviceAcademy.training.GSTestPart`

   If running windchill command does not create the table in SQL developer Please execute the Steps from Step 4 using the appropriate SQL.

Task 4: Inspect the objects that were created.

1. Login to Windchill to see if the part was created.
   Example:



Completion Criteria

1. Tables GSPart and GSPartMaster were created.
2. Part named “My first annotated part” was created.

This completes the exercise.

Exercise 4: Implement a Discussion Forum

Objectives

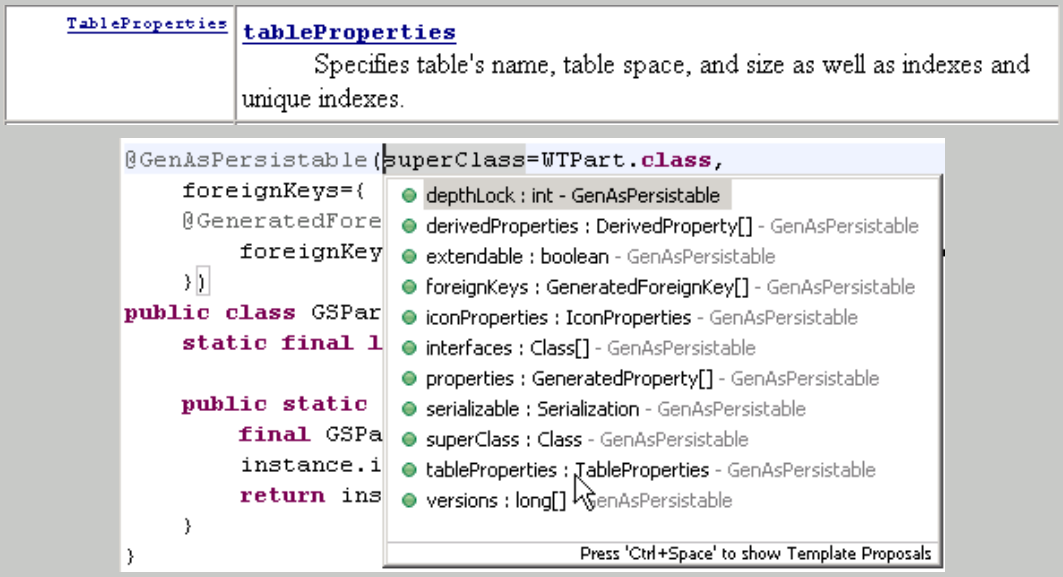
After successfully completing this exercise, you will be able to:

- Create a simple object model extension

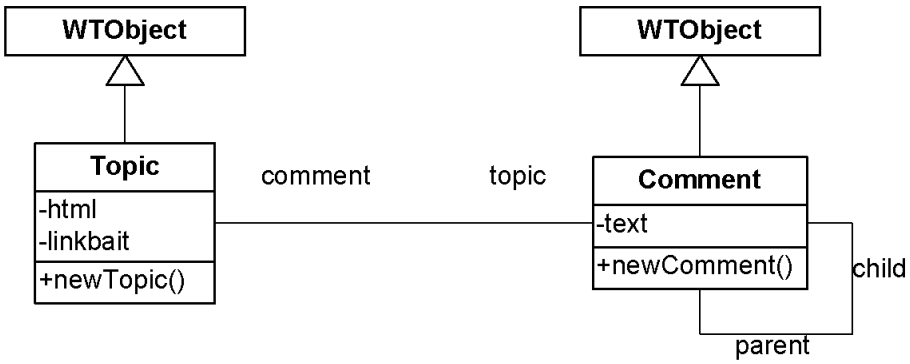
Scenario

A topic consists of a hyperlink and a short title. A comment has text and is authored by a principal. It may or may not be in reply to another comment.

Comment is a reserved word so the table will need to be “WTComment”. This can be implemented by using this the following argument in the “GenAsPerisistable” annotation: tableProperties=@TableProperties(tableName="WTComment")



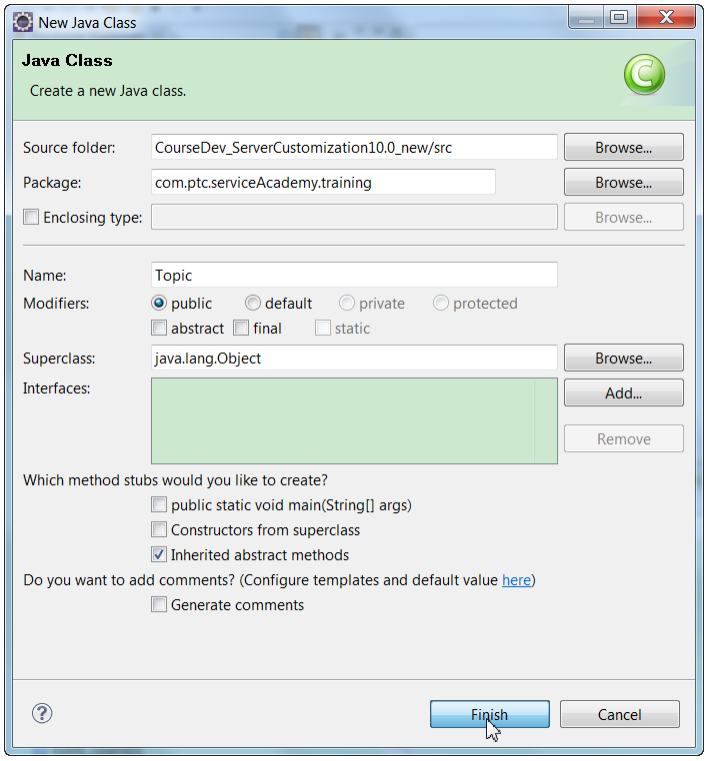
Source code is also available in student LabFiles.zip



UML Diagram for Exercise

Task 1: Create Topic.java

1. New class
   Example:



2. Add **extends** _Topic
3. Add "static final long serialVersionUID = 1;"
4. Add a constructor
   Example:

```
public static Topic newTopic() throws WTException {
final Topic instance = new Topic();
instance.initialize();
return instance;
}
```

5. Add Annotation for GenAsPersistable
   Example:

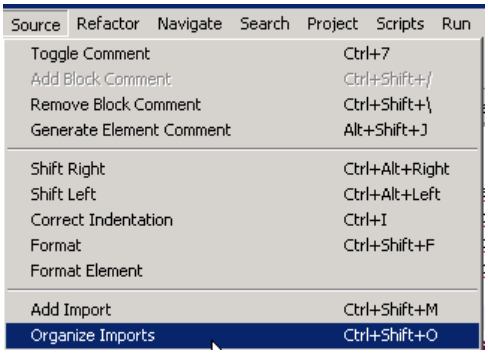
   @GenAsPersistable( )
6. Add arguments to GenAsPersistable for superClass
   Example:

```
superClass=WTObject.class
```

7. Add another argument for properties
   Example:

```
properties={
@GeneratedProperty(name="html", type=String.class),
@GeneratedProperty(name="linkBait", type=String.class)
}
```

8. Add Imports
   Example:



Result:

```
package com.ptc.serviceAcademy.training;

import wt.fc.WTObject;

@GenAsPersistable(superClass=WTObject.class,
        properties={
    @GeneratedProperty(name="html", type=String.class),
    @GeneratedProperty(name="linkBait", type=String.class)
    })
public class Topic extends _Topic{
    static final long serialVersionUID = 1;
    public static Topic newTopic() throws WTException {
        final Topic instance = new Topic();
        instance.initialize();
        return instance;
    }
}
```

9. Save — Errors should go away as the _Topic.java is created and compiled.

Result:

```
package com.ptc.serviceAcademy.training;

import wt.fc.WTObject;
import wt.util.WTException;

import com.ptc.windchill.annotations.metadata.GenAsPersistable;
import com.ptc.windchill.annotations.metadata.GeneratedProperty;

@GenAsPersistable(superClass=WTObject.class,
        properties={
    @GeneratedProperty(name="html", type=String.class),
    @GeneratedProperty(name="linkBait", type=String.class)
    })
public class Topic extends _Topic {
    static final long serialVersionUID = 1;
    public static Topic newTopic() throws WTException {
        final Topic instance = new Topic();
        instance.initialize();
        return instance;
    }
}
```

Task 2: In a similar way create Comment.java

1. Create a new class.
2. Add extends _Comment.java
3. Add “static final long serialVersionUID = 1;”

4. Add a constructor
5. Add an annotation for “GenAsPersistable”
6. Add superClass
7. Add properties
   Example:

```
properties={
@GeneratedProperty(name="text", type=String.class,
constraints=@PropertyConstraints(upperLimit=4000))
},
```

8. Add ForeignKeys
   Example:

```
foreignKeys={
@GeneratedForeignKey(
foreignKeyRole=@ForeignKeyRole(name="topic", type=Topic.class),
myRole=@MyRole(name="comment")),
@GeneratedForeignKey(
foreignKeyRole=@ForeignKeyRole(name="parent", type=Comment.class),
myRole=@MyRole(name="child"))
},
```

9. Add tableProperties to override the table name
   Example:

```
tableProperties=@TableProperties(tableName="WTComment"))
```

10. Organize Imports
11. Save — Errors should all go away
    Result:

```
package com.ptc.serviceAcademy.training;

import wt.fc.WTObject;

@GenAsPersistable(superClass=WTObject.class,
        properties={
    @GeneratedProperty(name="text", type=String.class,
    constraints=@PropertyConstraints(upperLimit=4000))
    },
    foreignKeys={
    @GeneratedForeignKey(
    foreignKeyRole=@ForeignKeyRole(name="topic", type=Topic.class),
    myRole=@MyRole(name="comment")),
    @GeneratedForeignKey(
    foreignKeyRole=@ForeignKeyRole(name="parent", type=Comment.class),
    myRole=@MyRole(name="child"))
    },
    tableProperties=@TableProperties(tableName="WTComment"))
public class Comment extends _Comment {

    static final long serialVersionUID = 1;
    public static Comment newComment() throws WTException{
        final Comment instance = new Comment();
        instance.initialize();
        return instance;
    }
}
```

Comment.java

12. Copy following files to WT_HOME/src/com/ptc/serviceAcademy/training
    - Topic.java
    - Comment.java

Task 3: Build Tables

1. Generate SQL
   - ant -f bin/tools.xml class -Dclass.includes=com/ptc/serviceAcademy/training/*
   - ant -f bin/tools.xml sql_script -Dgen.input=com.ptc.serviceAcademy.training.*
2. Switch to the db/sql directory

3. Execute: sqlplus pdmlink/pdmlink

pdmlink/pdmlink is the username/password for your Oracle.

4. Execute SQL
   - @Make_pkg_sql_Table.sql
   - @Make_pkg_sql_Index.sql

Task 4: Restart the Method Server

1. Restart the Method Server.

Change the database if required.

Completion Criteria

1. Tables TOPIC and WTCOMMENT were created.
2. The two object model extensions should be created.

This completes the exercise.

Module 7

RESTful Web Services

Module Overview

In this module, we highlight what are restful web services, Framework in Windchill, and AngularJS overview.

Objectives

After completing this module, you will be able to:

- Understand what are restful web services
- Framework in Windchill
- AngularJS overview

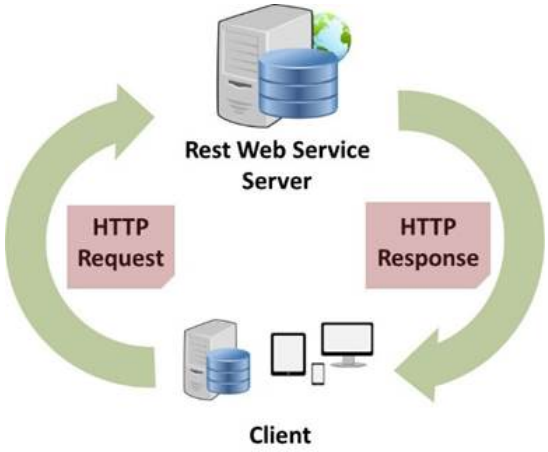
RESTful Web Services

What are Restful Web Services?

In Java EE 6, JAX-RS provides the functionality for Representational State Transfer (RESTful) web services. REST is well suited for basic, ad hoc integration scenarios. RESTful web services, often better integrated with HTTP than SOAP-based services are, do not require XML messages or WSDL service–API definitions.

The following principles encourage RESTful applications to be simple, lightweight, and fast:

- Resource identification through URI
- Uniform interface
- Self-descriptive messages
- Stateful interactions through hyperlinks



RESTful web services are built to work best on the Web. Representational State Transfer (REST) is an architectural style that specifies constraints, such as the uniform interface, that if applied to a web service induce desirable properties, such as performance, scalability, and modifiability, that enable services to work best on the Web. In the REST architectural style, data and functionality are considered resources and are accessed using Uniform Resource Identifiers (URIs), typically links on the Web. The resources are acted upon by using a set of simple, well-defined operations. The REST architectural style constrains an architecture to a client/server architecture and is designed to use a stateless communication protocol, typically HTTP. In the REST architecture style, clients and servers exchange representations of resources by using a standardized interface and protocol.

The following principles encourage RESTful applications to be simple, lightweight, and fast:

- **Resource identification through URI:** A RESTful web service exposes a set of resources that identify the targets of the interaction with its clients. Resources are identified by URIs, which provide a global addressing space for resource and service discovery.
- **Uniform interface :**Resources are manipulated using a fixed set of four create, read, update, delete operations: PUT, GET, POST, and DELETE. PUT creates a new resource, which can be then deleted by using DELETE. GET retrieves the current state of a resource in some representation. POST transfers a new state onto a resource.
- **Self-descriptive messages :**Resources are decoupled from their representation so that their content can be accessed in a variety of formats, such as HTML, XML, plain text, PDF, JPEG, JSON, and others. Metadata about the resource is available and used, for example, to control caching, detect transmission errors, negotiate the appropriate representation format, and perform authentication or access control.
- **Stateful interactions through hyperlinks :**Every interaction with a resource is stateless; that is, request messages are self-contained. Stateful interactions are based on the concept of explicit state transfer. Several techniques exist to exchange state, such as URI rewriting, cookies, and hidden form fields. State can be embedded in response messages to point to valid future states of the interaction.

HTTP Methods & Status Code

The following HTTP methods are most commonly used in a REST based architecture.

- **GET** − Provides a read only access to a resource.
- **PUT** − Used to create a new resource.
- **DELETE** − Used to remove a resource.
- **POST** − Used to update an existing resource or create a new resource.
- **OPTIONS** − Used to get the supported operations on a resource.

HTTP Status Code	Description/Reason
200	Successful response.
400	If one of the URL or query parameters is not in the correct format.
404	If the specified objects does not exist.
500	If an unexpected error occurs.

HTTP Status Code

RESTful Web Services make use of HTTP protocols as a medium of communication between client and server. A client sends a message in form of a HTTP Request and the server responds in the form of an HTTP Response. This technique is termed as Messaging. These messages contain message data and metadata i.e. information about message itself.

Following HTTP methods are most commonly used in a REST based architecture:

- GET operations are read only and are safe.
- PUT and DELETE operations are idempotent, which means their result will always be the same, no matter how many times these operations are invoked.
- PUT and POST operation are nearly the same with the difference lying only in the result where the PUT operation is idempotent and POST operation can cause a different result.

Some HTTPStatus Codes are explained in table above.

Annotations to map a resource as a web service resource

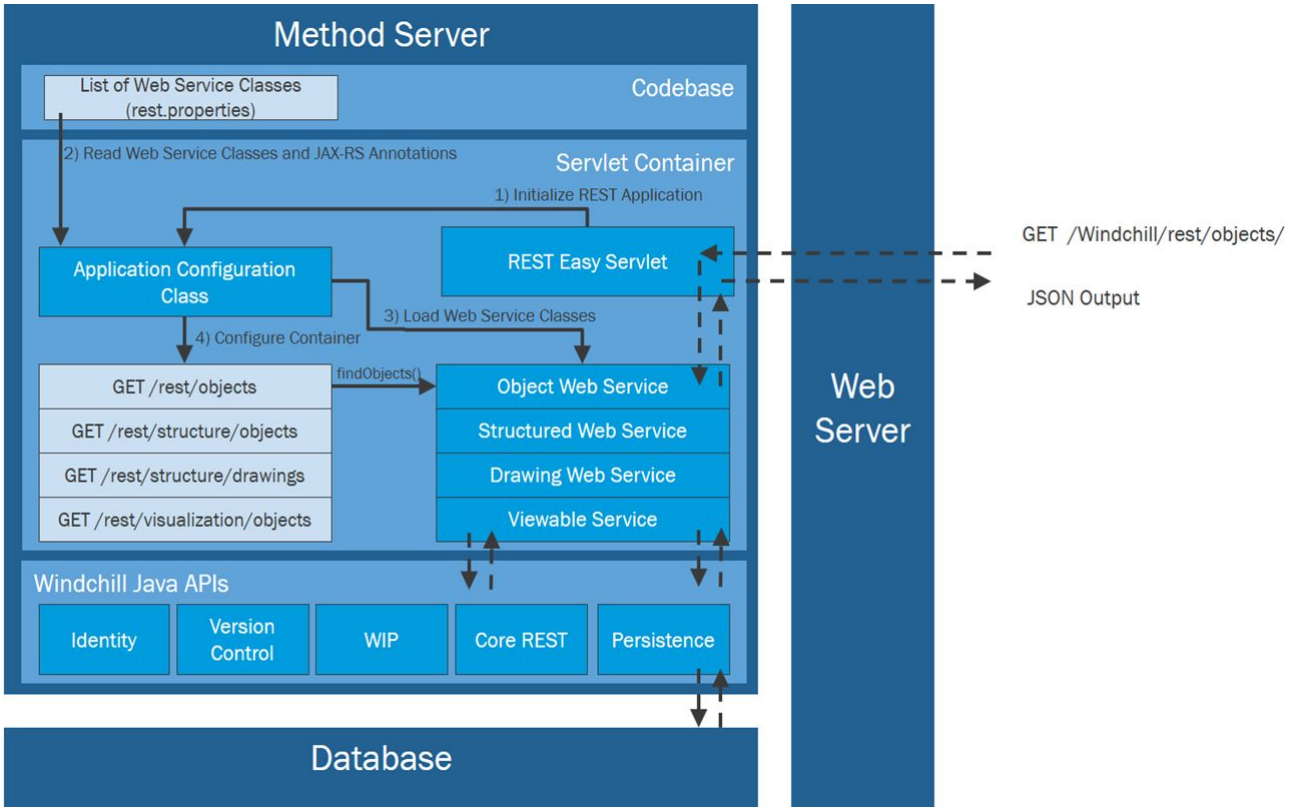
Following table lists some of the Java programming annotations that are defined by JAX-RS, with a brief description of how each is used.

Annotation	Description
`@Path`	The `@Path` annotation's value is a relative URI path indicating where the Java class will be hosted: for example, `/helloworld` . You can also embed variables in the URIs to make a URI path template. For example, you could ask for the name of a user and pass it to the application as a variable in the URI: `/helloworld/{username}` .
`@GET`	The `@GET` annotation is a request method designator and corresponds to the similarly named HTTP method. The Java method annotated with this request method designator will process HTTP GET requests. The behavior of a resource is determined by the HTTP method to which the resource is responding.
`@POST`	The `@POST` annotation is a request method designator and corresponds to the similarly named HTTP method. The Java method annotated with this request method designator will process HTTP POST requests. The behavior of a resource is determined by the HTTP method to which the resource is responding.
`@PUT`	The `@PUT` annotation is a request method designator and corresponds to the similarly named HTTP method. The Java method annotated with this request method designator will process HTTP PUT requests. The behavior of a resource is determined by the HTTP method to which the resource is responding.
`@DELETE`	The `@DELETE` annotation is a request method designator and corresponds to the similarly named HTTP method. The Java method annotated with this request method designator will process HTTP DELETE requests. The behavior of a resource is determined by the HTTP method to which the resource is responding.

JAX-RS is a Java programming language API designed to make it easy to develop applications that use the REST architecture.

The JAX-RS API uses Java programming language annotations to simplify the development of RESTful web services. JAX-RS annotations are runtime annotations; therefore, runtime reflection will generate the helper classes and artifacts for the resource.

Framework in Windchill



Architecture Showing R11 Web Services:

Above diagram shows architecture of Windchill showing R11 Web Services which consists of following steps:

- Initialize REST Application
- Read Web Service Classes and JAX-RS Annotations
- Load Web Service Classes
- Configure Container

Elements of a Web Service in Windchill R11

- **Java API for RESTful services JAX-RS**
  - Allows processing of GET/PUT/POST requests for URIs via methods in Java classes
  - Annotations are used to map HTTP requests for URIs to actual class methods
  - Windchill 11 uses RESTEasyimplementation of JAX-RS from Red Hat JBOSS
- **OData standard protocol for producing web service content in response to requests**
  - Standardized by OASIS (Organization for Advancement of Structured Information Standards)
  - Windchill 11 uses odata4j, a Java framework that implements the OData standard
- **Swagger**
  - Allows documentation and description of a RESTful API by programmers via code annotations
  - Understands JAX-RS annotations and helps build a UI that can be used to invoke RESTful APIs
  - Helps with testing of the RESTful API

These are the elements of a web service in Windchill R11. Java API for RESTful services JAX-RS allows processing of GET/PUT/POST requests for URIs via methods in Java classes. OData standard protocol for producing web service content in response to requests and Swagger UI allows documentation and description of a RESTful API by programmers via code annotations.

Windchill REST Endpoints

Object Web Service

- Resource: /objects
- Allows querying Windchill objects and their specified attributes
- Applies to Windchill business objects

Method Name	Description	Relative URL	HTTP Method
Get Object By Id	Queries the Windchill object specified by its object identifier string	/{oid}	GET
Find Objects	Returns a list of Windchill object ids that meet the specified query criteria	/	GET

Windchill REST Endpoints:

Object Web Service allows querying Windchill objects and their specified attributes.

Windchill REST Endpoints

Structure Web Service

- Resource: /structure/objects
- Allows navigation of objects that are linked together to form tree structures
- Applies to Parts, Documents, CAD Documents

Method Name	Description	Relative URL	HTTP Method
Get Object By Id	Queries the Windchill object specified by its object identifier string	/{oid}	GET
Find Objects	Returns a list of Windchill object ids that meet the specified query criteria	/	GET
Get Descendants	Gets child objects for a given Windchill object for multiple levels	/{oid}/descendants	GET
Get Ancestors	Gets the parent objects for a given Windchill object for multiple levels	/{oid}/ancestors	GET
Get Uses Summaries	Gets child objects for a given Windchill object with link information (supports multiple levels)	/{oid}/usesSummaries	GET
Get Used By Summaries	Gets parent objects for a given Windchill object with link information (supports multiple levels)	/{oid}/usedBySummaries	GET

Structure Web Service allows navigation of objects that are linked together to form tree structures.

Windchill REST Endpoints

Viewable Web Service

- Resource: /visualization/objects
- Allows access to Windchill visualization data
- Applies to Parts, CAD Documents

Method Name	Description	Relative URL	HTTP Method
Get Viewable	Gets links to representation and files for a given representable object	/{oid}	GET

Viewable Web Service allows access to Windchill visualization data.

Windchill REST Endpoints

Saved Search Web Service

- Resource: /search/saved-searches
- Allows access to searches saved in Windchill

Method Name	Description	Relative URL	HTTP Method
Find Saved Search	Find a saved search based on given criteria	/	GET
Get Saved Search	Get a saved search given its object identifier string	/{oid}	GET
Execute Saved Search	Execute a saved search identified by its object identifier string	/{oid}/results	POST

Saved Search Web Service allows access to searches saved in Windchill.

Windchill REST Endpoints

Drawing Web Service

- Resource: /structure/drawings
- Allows access to drawings tied to CAD Documents and Parts

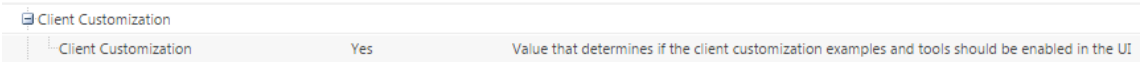
Method Name	Description	Relative URL	HTTP Method
Find Saved Search	Find a saved search based on given criteria	/	GET
Get Saved Search	Get a saved search given its object identifier string	/{oid}	GET
Execute Saved Search	Execute a saved search identified by its object identifier string	/{oid}/results	POST

Drawing Web Service allows access to drawings tied to CAD Documents and Parts

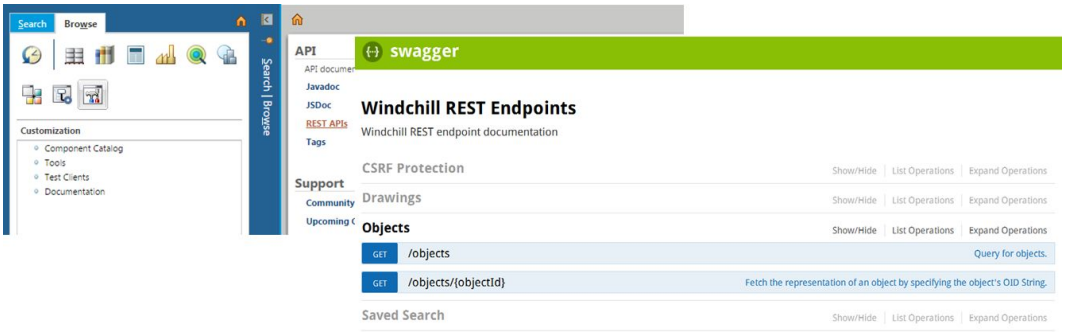
Swagger UI

Accessing Swagger UI

- Enable site preference “Client Customization”



- Navigate to **Customization > Documentation > REST APIs**to launch Swagger UI



Above figure explain about how to access Swagger UI.

Windchill Service Parts REST APIs to Access Parts List Information

These APIs are provided:

- Path—**/v1//partslists**

  Description—Returns all parts lists that match specified criteria
- Path—**/v1/partslists/{partsListId}**

  Description—Returns a parts list with a specified ID
- Path—**/v1/partslists/{partsListId}/items**

  Description—Returns all parts list items in a specified parts list
- Path—**/v1/partslists/{partsListId}/items/{partsListItemId}**

  Description—Returns a parts list item with a specified ID in a specified parts list
- Path—**/v1/partslists/{partsListId}/items/{partsListItemId}/part**

  Description—Returns the part associated with a specified parts list item in a specified parts list
- Path—**/v1/partslists/{partsListId}/itemrelations**

  Description—Returns all item relations (substitute and supplementary parts) in a specified parts list

Windchill Service Parts provides REST APIs to access parts list, parts list item and illustration data. The APIs expose read-only information to web services such as ThingWorx. They can be used by any client that is capable of making an HTTP request.

AngularJS Overview

Definition of AngularJS is as follows:

AngularJS is a structural framework for dynamic web apps. It lets you use HTML as your template language and lets you extend HTML's syntax to express your application's components clearly and succinctly. Angular's data binding and dependency injection eliminate much of the code you currently have to write. And it all happens within the browser, making it an ideal partner with any server technology.

Features:

- A powerful JavaScript based development framework
- Client side application development in a clean MVC way
- Cross-browser compliant
- Open source, completely free, and used by thousands of developers around the world

AngularJS is an open source web application framework. It is a very powerful JavaScript Framework. It is used in Single Page Application (SPA) projects. It extends HTML DOM with additional attributes and makes it more responsive to user actions. AngularJS is open source, completely free, and used by thousands of developers around the world.

Following are some features of AngularJS:

- AngularJS is a powerful JavaScript based development framework to create RICH Internet Application(RIA).
- AngularJS provides developers options to write client side application (using JavaScript) in a clean MVC (Model View Controller) way.
- Application written in AngularJS is cross-browser compliant. AngularJS automatically handles JavaScript code suitable for each browser.
- AngularJS is open source, completely free, and used by thousands of developers around the world. It is licensed under the Apache License version 2.0.

Overall, AngularJS is a framework to build large scale and high performance web application while keeping them as easy-to-maintain.

REFERENCES

- http://resteasy.jboss.org/downloads
- http://docs.jboss.org/resteasy/docs/3.0.16.Final/userguide/html/index.html
- https://docs.angularjs.org/api

Note: The REST framework for Windchill is still in progress and the REST API customization is not yet supported.

Exercise 1: To enable a REST service in Windchill to query WTPart and Use the AngularJS to display the results in tabular format.

Objectives

After successfully completing this exercise, you will be able to:

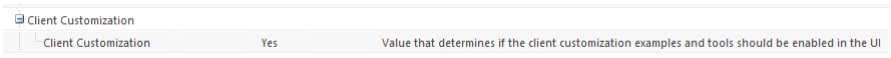
- .Enable a REST service in Windchill to query WTPart
- Use the AngularJS to display the results in tabular format

Scenario

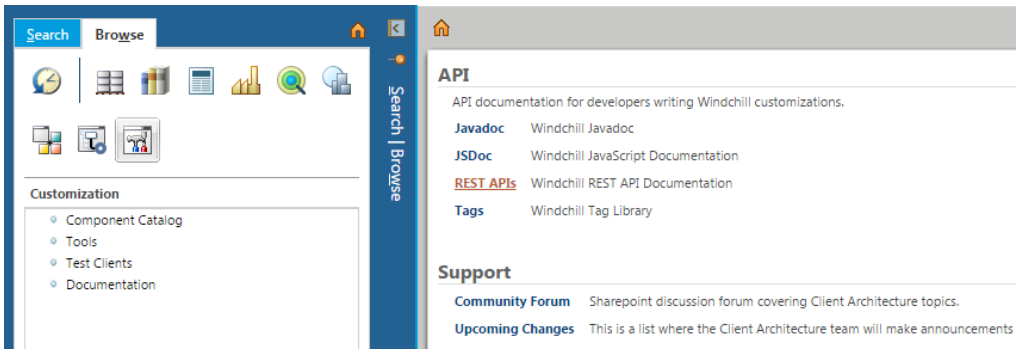
In this exercise, you will Enable a REST service in Windchill to query WTPart using Swagger UI and Use the AngularJS to display the results in tabular format.

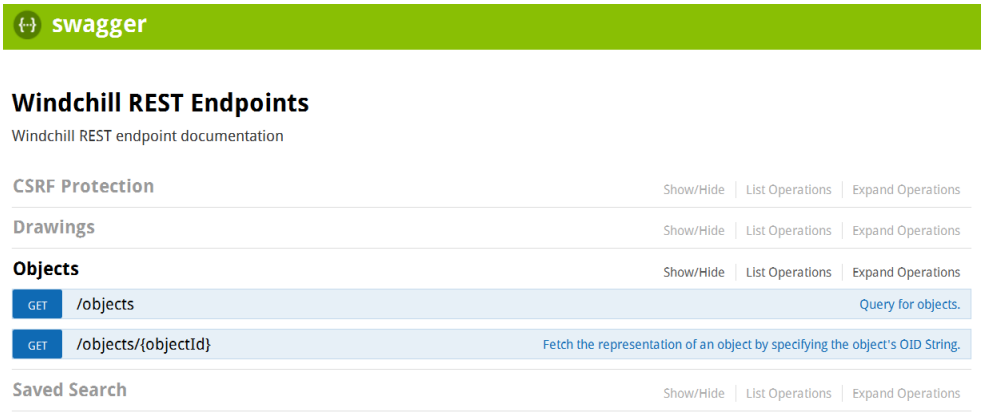
Task 1: Access Swagger UI .

1. Enable site preference “Client Customization” by using following steps:
   - Navigate to Site > Utilities > Preference Management.
   - Right click on Client Customization, select Set Preference and set the value to Yes.
   - Click Ok.



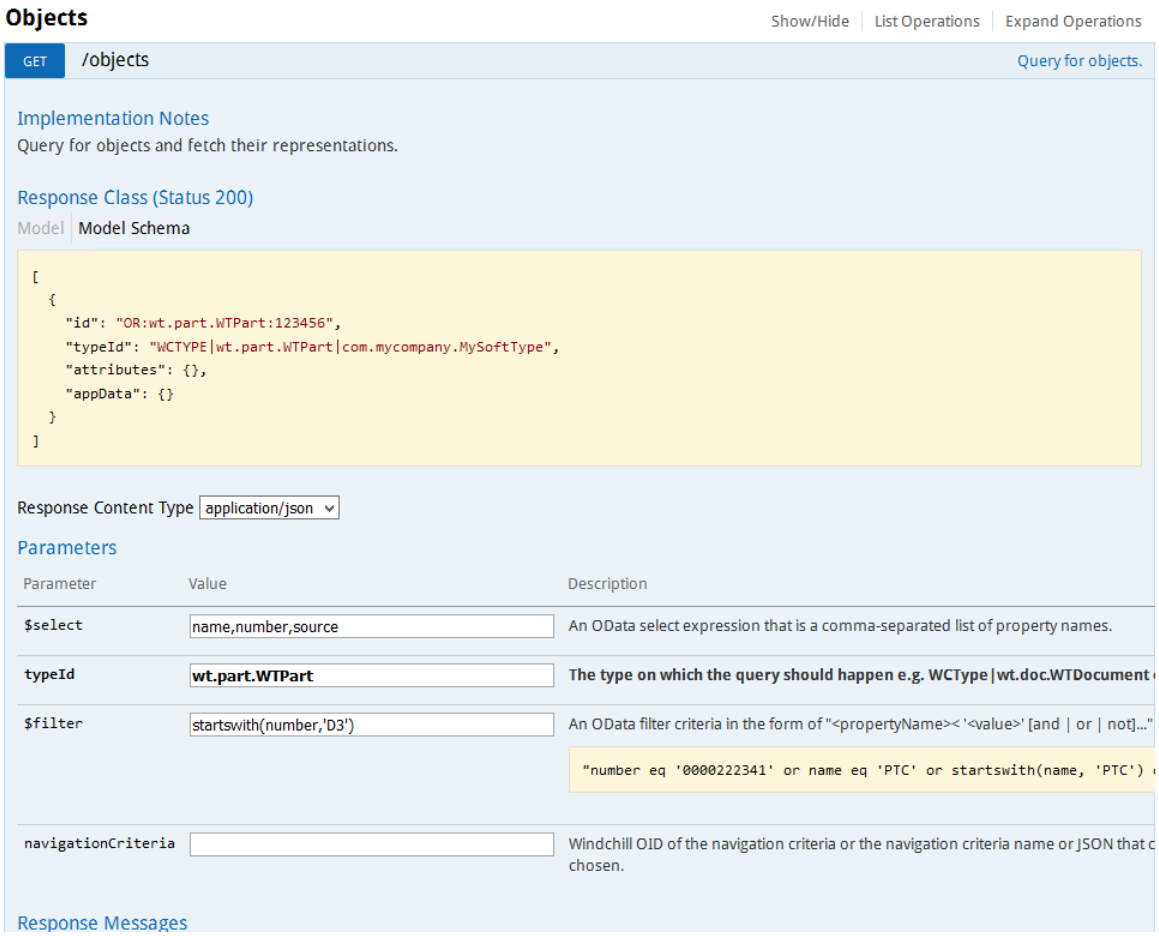
2. Navigate to Customization > Documentation > REST APIs to launch Swagger UI.



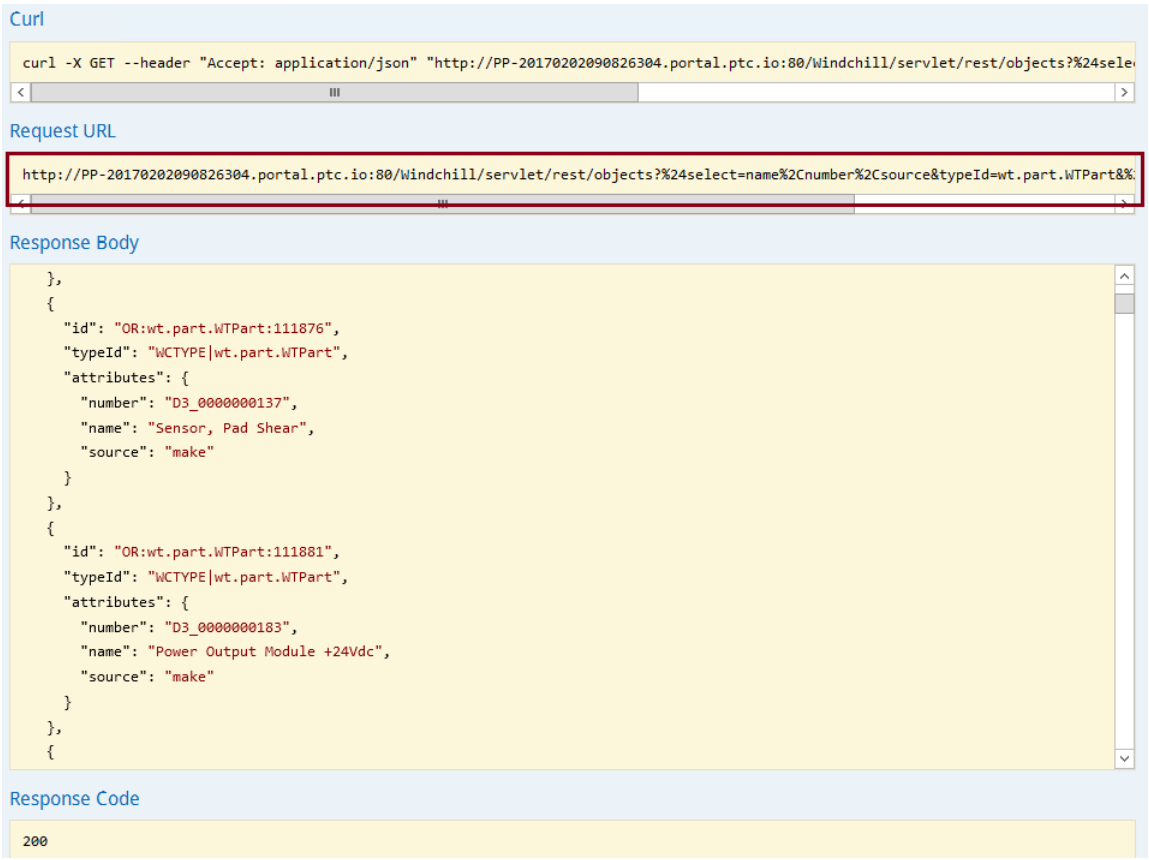


Task 2: Query WTPart by enabling a REST service in Windchill.

1. Enter parameter values for the RESTful service :
   - **$select** with the value of **name,number,source**
   - **typeId** with the value of **wt.part.WTPart**
   - **$filter** with the value of **startswith(number,’D3’)**

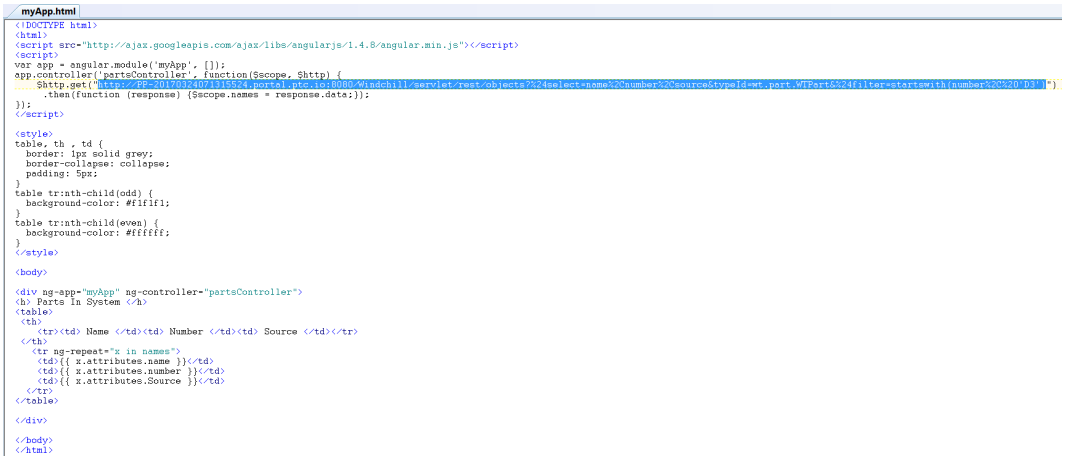


2. Swagger UI generates the HTTP request to call the service and displays results, copy the Request URL generated in this step.



Task 3: Use the AngularJS to display the results in tabular format.

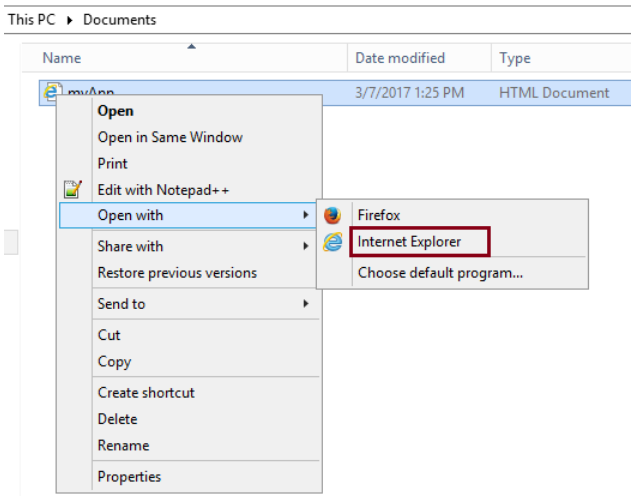
1. Open *lab files >Module 06 - Restful Web Services > myApp.htm* in any editor and replace the URL inside of *http.get("_")* highlighted in following code with the Request URL copied in previous step.



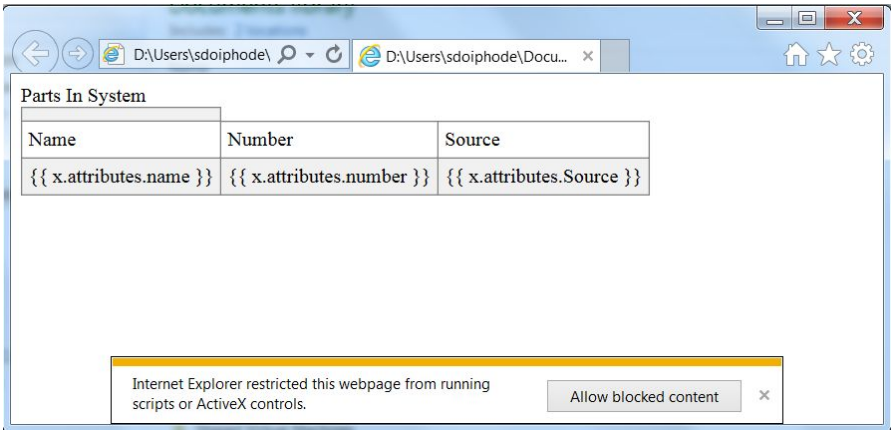
2. Save the changes.

Note: Save this file as .htm file.

3. Execute the code by right click on the file name and select “Open with-> Internet Explorer”.



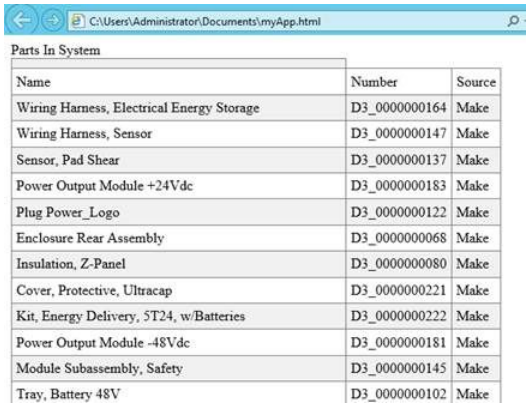
You may receive message in Internet Explorer as "Internet Explorer restricted this webpage from running scripts or ActiveX controls, Click on **Allow blocked content**.



Then, you will receive a pop-up dialog box asking for your username and password. Enter username as **wcadmin** and password as **wcadmin** and Click Ok.

Result:

The result is displayed as follows:



Name	Number	Source
Wiring Harness, Electrical Energy Storage	D3_0000000164	Make
Wiring Harness, Sensor	D3_0000000147	Make
Sensor, Pad Shear	D3_0000000137	Make
Power Output Module +24Vdc	D3_0000000183	Make
Plug Power_Logo	D3_0000000122	Make
Enclosure Rear Assembly	D3_0000000068	Make
Insulation, Z-Panel	D3_0000000080	Make
Cover, Protective, Ultracap	D3_0000000221	Make
Kit, Energy Delivery, 5T24, w/Batteries	D3_0000000222	Make
Power Output Module -48Vdc	D3_0000000181	Make
Module Subassembly, Safety	D3_0000000145	Make
Tray, Battery 48V	D3_0000000102	Make

Completion Criteria

1. WTPart list is displayed in Response body
2. Result is displayed in tabular format using AngularJS code.

This completes the exercise.