# Model Content Reports

## Timestamp: 2013-August-23 02:53:00 ()

## Overview

This was generated from [$project.getTitle()] on $date.get('yyyy-MM-dd H:m:s').

### Interpreting This Section

This section shows each of the components of the model with their OWL construct names where applicable. These are:

| **Construct Name** | **Description** |
| --- | --- |
| **Model Section:** | A grouping of ontologies with some common theme. These also share a namespace fragment in the corresponding OWL files. |
| **owlOntology** | A single OWL ontology. |
| **owlClass** | An OWL Class, that is a set theoretic construct representing a common set of properties, possession of which would make any individual a member of this set. |
| **owlObjectProperty** | The Class named as “Range” for the relationship represents something in terms of which the meaning of the relationship is framed.  Known as “Relationship fact” in business spreadsheets. |
| **rdfsSubClassOf** | **“is a”** relationships - these have no definition. This relationship indicates that the Class is a sub-class of the Class named as the “Range” in the relationship.  Known as “Parent” in business spreadsheets. |
| **owlDatatypeProperty** | Some property framed in terms of some simple type of information such as text or a “yes or no” value.  Known as “Simple Fact” in business spreadsheets. |
| **owlDatatypeProperty Range** | The type of information in which the OWL Datatype Property is framed  Known as “Simple Type” in business spreadsheets.  NOTE: for some datatype properties, the range is a DataEnumeration (see below).  NOTE: For some datatype properties, the fact type is given as a Class e.g. Monetary Amount. In such cases, this is intended to be an OWL Object Property. The use of this style of object property is a convenience for diagrams production. This will be corrected in future versions of this specification. |
| **DataEnumeration** | These item represent a selection of possible values, which are intended to be taken as literal (e.g. textual) values. A “Simple Fact” (OWL Datatype Property) may identify one of these as the Simple Fact Type; this means that any one of the values in the list may be a possible value for this property. |
| **UnionClass** | This corresponds to a logical union of Classes. The membership of the union is not shown in this report. |
| **disjointWith** | Identifies two sets of which no one individual may be a member of both.  Known as “mutually exclusive” in business spreadsheets. |
| **Definition** | The SKOS Definition annotation, giving the formal definition of the item |
| **Editorial Note** | The SKOS Editorial Note annotation, giving additional narrative about the term and definition. Includes line breaks and additional narrative headings within this annotation, i.e. everything up to the next annotation or construct entry is part of this annotation. |
| **Scope Note** | The SKOS Editorial Note annotation, giving notes about the scope and application of the term. |
| **Term Origin** | A temporary annotation, to be replaced by a range of FIBO-specific annotations derived from the Dublin Core “source” property. These will include:  TermOriginDocument  TermOriginStandard  TermOriginalTerm |
| **Definition Origin** | A temporary annotation, to be replaced by a range of FIBO-specific annotations derived from the Dublin Core “source” property. These will include:  DefinitionOrigin  DefinitionAdaptedFrom |
| **Consensus** | An annotation from the EDM Council working sessions, this will not be included in the formal submission of this specification and these will be removed. |

### Model: Vocabulary, Annotation and Other Supporting Terms

#### Dublin Core

#### SKOS

#### OMG AB Specification Metadata

#### Common Ontology Metadata

#### Ontology: Annotation Vocabulary

## Model: FIBO-Foundations

### Model Section: <Need Name Here>

## Facts (with Predicates) are implemented with <<fact>> Dependency relationships

#macro (trimAndPeriod $rawString)

#set ($trimString = $rawString.trim())

#if ($trimString.length() == (1 + $trimString.lastIndexOf(‘.’)))

#set ($periodString = $trimString)

#else

#set ($periodString = $trimString + “.”)

#end

#end

## end of macro trimAndPeriod

##

#macro (getFacts $forElement $relatedFacts)

#set ($relatedFacts = $array.createArray())

#set ($thisRelationships = $report.getRelationship($forElement))

#foreach ($rel in $thisRelationships)

#if ($rel.humanName.contains(“fact”))

#set ($consume\_lhs = $relatedFacts.add($rel))

#end

#end

#end

## end of the macro getFacts

##

## Predicates are implemented with <<predicate>> Dependency relationships to <<facts>>

#macro (getMapPredicateFact $forFacts $mapPredicateFact)

#set ($mapPredicateFact = $map.createHashMap())

#foreach ($fact in $forFacts)

#set ($thisRelationships = $report.getRelationship($fact))

#foreach ($rel in $thisRelationships)

#if ($rel.humanName.contains(“predicate”))

#set ($predName = $rel.supplier.get(0).name)

#set ($consume\_lhs = $mapPredicateFact.put($predName, $fact))

#end

#end

#end

#end

## end of the macro getMapPredicateFact

##

## using a map of predicate names to facts for the current ontology,

## lookup the fact for a predicate name and obtain the string for the

## value of the literal at the end of the fact

#macro (getFactValueByPredicateName $mapPredicateNamesToFact $key)

#if ($mapPredicateNamesToFact.containsKey($key))

#set ($fact = $mapPredicateNamesToFact.get($key))

#set ($literal = $fact.supplier.get(0))

#set ($value = $report.createValueSpecificationText($literal.specification))

#else

#set ($value = $null)

#end

#end

## end of the macro getFactValueByPredicateName

##

## Ontology References are implemented with <<references>> Package Import relationships

#macro (getReferencedOntologies $forOntology)

#set ($referencedOntologies = $array.createArray())

#set ($thisRelationships = $report.getRelationship($forOntology))

#foreach ($rel in $thisRelationships)

#if ($report.getAppliedStereotypeByName($rel, “references”))

#set ($referent = $rel.target.get(0))

#if ($referent != $forOntology)

#set ($consume\_lhs = $referencedOntologies.add($referent))

#end

#end

#end

#end

## end of the macro getReferencedOntologies

##

## Ontology Imports are implemented with <<owlImports>> Package Imports relationships

#macro (getImportedOntologies $forOntology)

#set ($importedOntologies = $array.createArray())

#set ($thisRelationships = $report.getRelationship($forOntology))

#foreach ($rel in $thisRelationships)

#if ($report.getAppliedStereotypeByName($rel, “owlImports”))

#set ($referent = $rel.target.get(0))

#if ($referent != $forOntology)

#set ($consume\_lhs = $importedOntologies.add($referent))

#end

#end

#end

#end

## end of the macro getImportedOntologies

##

## obtain the name of a referenced or imported ontology

#macro (getReferencedName $referent)

#set ($referentName = “Identifier Not Found. Check Model Content.”)

#set ($tagValue = $report.getStereotypeProperty($referent, “owlOntology”, “identifier”))

#if ($tagValue)

#set ($referentName = $report.createValueSpecificationText($tagValue.get(0).specification))

#end

#end

## end of the macro getReferencedName

## find all Packages that are <<owlOntology>> packages

#macro (getOntologyPackages)

#set ($ontologyPackages = $array.createArray())

#foreach($pkg in $report.filter($elements, “humanType”, [“owlOntology”]))

#set ($consume\_lhs = $ontologyPackages.add($pkg))

#end

#end

## end of the getOntologyPackages macro

##

#macro (getExemplarDiagram $thisCls)

#set ($exemplarDiagram = $null)

#foreach($dep in $report.filter($report.getRelationship($thisCls), “name”, [“Exemplar Diagram”]))

#set ($exemplarDiagram = $dep.target.get(0))

#end

#end

## end of the getExemplarDiagram macro

##

#macro (getParents $thisClass $parents )

#foreach ($rel in $report.getRelationship($thisClass))

#foreach ($stereo in $rel.appliedStereotype)

#if ($stereo.name.contains(“subClassOf”))

#set ($parent = $rel.general)

#if ($parent != $thisClass)

#foreach ($parentStereo in $parent.appliedStereotype)

#if ($parentStereo.name.contains(“owlClass”))

#set ($consume\_lhs = $parents.add($parent))

#end

#end

#end

#end

#end

#end

## end of foreach $rel (Parents)

#end

## end of the getParents macro

##

#macro (getDisjoints $thisClass $disjoints )

#foreach ($rel in $report.getRelationship($thisClass))

#foreach ($stereo in $rel.appliedStereotype)

#if ($stereo.name.contains(“disjointWith”))

#set ($disjoint = $rel.target.get(0))

#if ($disjoint != $thisClass)

#foreach ($disjointStereo in $disjoint.appliedStereotype)

#if ($disjointStereo.name.contains(“owlClass”))

#set ($consume\_lhs = $disjoints.add($disjoint))

#end

#end

#end

#end

#end

#end

## end of foreach $rel (Disjoints)

#end

## end of the getDisjoints macro

##

#macro (serializeParents $parents )

#if ($parents.size() > 0)

Parents

#foreach ($parent in $sorter.sort($parents))

* $parent.name

## end of foreach $rel (Parents)

#end

#end

#end

## end of the serializeParents macro

##

#macro (serializeDisjoints $disjoints )

#if ($disjoints.size() > 0)

Disjoint With

#foreach ($disjoint in $sorter.sort($disjoints))

* $disjoint.name

## end of foreach $rel (Disjoints)

#end

#end

#end

## end of the serializeDisjoints macro

##

#macro (getRestrictions $thisClass $restrictions )

#foreach ($rel in $report.getRelationship($thisClass))

#foreach ($stereo in $rel.appliedStereotype)

#if ($stereo.name.contains(“subClassOf”))

#set ($restriction = $rel.general)

#if ($restriction != $thisClass)

#foreach ($restrictStereo in $restriction.appliedStereotype)

#if ($restrictStereo.name.contains(“owlRestriction”))

#set ($consume\_lhs = $restrictions.add($restriction))

#end

#end

#end

#end

#end

#end

#end

## end of macro getRestrictions

##

#macro (serializeRestrictions $thisClass $restrictions )

#if (0 < $restrictions.size())

Subtyping Restrictions

These restrictions each define a "necessary" condition for an entity being a member of the $thisClass.name class.

#foreach ($restriction in $restrictions)

#set ($onClass = $null)

#set ($onProperty = $null)

#set ($allValuesFrom = $null)

#set ($someValuesFrom = $null)

#foreach ($restrictRel in $report.getRelationship($restriction))

#foreach ($restrictStereo in $restrictRel.appliedStereotype)

#if ($restrictStereo.name.contains(“onClass”))

#set ($onClass = $restrictRel.supplier.get(0))

#end

#if ($restrictStereo.name.contains(“onProperty”))

#set ($onProperty = $restrictRel.supplier.get(0))

#end

#if ($restrictStereo.name.contains(“allValuesFrom”))

#set ($allValuesFrom = $restrictRel.supplier.get(0))

#end

#if ($restrictStereo.name.contains(“someValuesFrom”))

#set ($someValuesFrom = $restrictRel.supplier.get(0))

#end

#end

#end

#if (!$report.isEmpty($onProperty))

#if (!$report.isEmpty($onClass))

#set ($cardinality = $report.getStereotypePropertyString($restriction, “owlRestriction”, “cardinality”))

#if (!$report.isEmpty($cardinality))

* CardinalityRestriction $cardinality on property $onProperty.name qualified by class $onClass.name

#end

#set ($minCardinality = $report.getStereotypePropertyString($restriction, “owlRestriction”, “minCardinality”))

#if (!$report.isEmpty($minCardinality))

* MinCardinalityRestriction $minCardinality on property $onProperty.name qualified by class $onClass.name

#end

#set ($maxCardinality = $report.getStereotypePropertyString($restriction, “owlRestriction”, “maxCardinality”))

#if (!$report.isEmpty($maxCardinality))

* MaxCardinalityRestriction $maxCardinality on property $onProperty.name qualified by class $onClass.name

#end

#set ($hasSelf = $report.getStereotypePropertyString($restriction, “owlRestriction”, “hasSelf”))

#if (!$report.isEmpty($hasSelf))

* HasSelfRestriction $hasSelf on property $onProperty.name qualified by class $onClass.name

#end

#else

## no onClass but has onProperty

#if (!$report.isEmpty($allValuesFrom))

* AllValuesFromRestriction on property $onProperty.name *only* values from $allValuesFrom.name

#elseif (!$report.isEmpty($someValuesFrom))

* SomeValuesFromRestriction on property $onProperty.name *some* values from $someValuesFrom.name

#else

#set ($cardinality = $report.getStereotypePropertyString($restriction, “owlRestriction”, “cardinality”))

#if (!$report.isEmpty($cardinality))

* CardinalityRestriction $cardinality on property $onProperty.name

#end

#set ($minCardinality = $report.getStereotypePropertyString($restriction, “owlRestriction”, “minCardinality”))

#if (!$report.isEmpty($minCardinality))

* MinCardinalityRestriction $minCardinality on property $onProperty.name

#end

#set ($maxCardinality = $report.getStereotypePropertyString($restriction, “owlRestriction”, “maxCardinality”))

#if (!$report.isEmpty($maxCardinality))

* MaxCardinalityRestriction $maxCardinality on property $onProperty.name

#end

#set ($hasSelf = $report.getStereotypePropertyString($restriction, “owlRestriction”, “hasSelf”))

#if (!$report.isEmpty($hasSelf))

* HasSelfRestriction $hasSelf on property $onProperty.name

#end

#end

#end

#end

#end

#end

## end of foreach $rel

#end

## end of the getRestrictions macro

##

#macro (getClasses $thisOnto $ontoClasses)

#foreach ($elem in $thisOnto.ownedElement)

#foreach($stereo in $elem.appliedStereotype)

#if ($stereo.name.contains(“owlClass”))

#set ($consume\_lhs = $ontoClasses.add($elem))

#end

#end

#end

#end

## end of macro getClasses

#macro (getObjPropProperties $thisObj $properties $asRole )

## get the facts for this property

#set ($propertyFacts = $array.createArray())

#set ($propPredicateFactMap = $map.createHashMap())

#getFacts( $thisObj, $propertyFacts )

#getMapPredicateFact( $propertyFacts, $propPredicateFactMap )

#set ($propMap = $map.createHashMap())

#set ($propName = $thisObj.name)

#set ($consume\_lhs = $propMap.put(“name”, $propName))

#set ($definition = “”)

#getFactValueByPredicateName($propPredicateFactMap, “definition”)

#if (!$report.isEmpty($value))

#trimAndPeriod($value)

#set ($definition = $periodString)

#end

#getFactValueByPredicateName($propPredicateFactMap, “explanatoryNote”)

#if (!$report.isEmpty($value))

#trimAndPeriod($value)

#set ($definition = “Definition: “ + $definition + “ Explanatory note: “ + $periodString)

#end

#set ($consume\_lhs = $propMap.put(“annotation”, $definition ))

#set ($consume\_lhs = $propMap.put(“type”, $asRole.name ))

##

## get any parent or inverseOf objectProperties of the objectProperty

#foreach ($propRel in $report.getRelationship($thisObj))

#foreach ($stereo in $propRel.appliedStereotype)

#if ($stereo.name.contains(“subPropertyOf”))

#set ($parent = $propRel.general)

#if ($parent != $thisObj)

#set ($consume\_lhs = $propMap.put(“parent”, $parent.name ))

#end

#else

#if ($stereo.name.contains(“inverseOf”))

#set ($inverse = $propRel.supplier.get(0))

#if ($inverse != $thisObj)

#set ($consume\_lhs = $propMap.put(“inverse”, $inverse.name ))

#end

#end

#end

#end

#end

##

#set ($consume\_lhs = $properties.add($propMap))

#end

## end of the getObjPropProperties macro

##

#macro (getClassDatatypeProperties $thisClass $clsDataProperties )

#foreach ($rel in $report.getRelationship($thisClass))

#foreach ($stereo in $rel.appliedStereotype)

#if ($stereo.name.contains(“datatypeProperty”))

#set ($prop = $rel)

#foreach($role in $prop.memberEnd)

#if ($role.type != $thisClass)

#if ($role.navigable)

#getObjPropProperties( $prop, $clsDataProperties, $role.type )

#end

#end

#end

#end

#end

#end

#end

## end of the getClassDatatypeProperties macro

##

#macro (getClassProperties $thisClass $clsProperties )

#foreach ($rel in $report.getRelationship($thisClass))

#foreach ($stereo in $rel.appliedStereotype)

#if ($stereo.name.contains(“objectProperty”))

#set ($prop = $rel)

#foreach($role in $prop.memberEnd)

#if ($role.type != $thisClass)

#if ($role.navigable)

#getObjPropProperties( $prop, $clsProperties, $role.type )

#end

#end

#end

#end

#end

#end

#end

## end of the getClassProperties macro

##

#macro (getOntoProperties $thisOnto $ontoProperties)

#foreach ($elem in $thisOnto.ownedElement)

#foreach($stereo in $elem.appliedStereotype)

#if ($stereo.name.contains(“objectProperty”))

#set ($consume\_lhs = $ontoProperties.add($elem))

#end

#end

#end

#end

## end of macro getOntoProperties

##

#macro (getOntoUnions $thisOnto $ontoUnions)

#foreach ($outerPkg in $thisOnto.nestedPackage)

#foreach ($pkg in $outerPkg.nestedPackage)

#if ($pkg.name.contains(“Blank Nodes”))

#foreach ($elem in $pkg.ownedElement)

#foreach($stereo in $elem.appliedStereotype)

#if ($stereo.name.contains(“UnionClass”))

#set ($consume\_lhs = $ontoUnions.add($elem))

#end

#end

#end

#end

#end

#end

#end

## end of macro getOntoUnions

##

#macro (getUnionsOf $thisUnion $unionsOf)

#set ($thisRelationships = $report.getRelationship($thisUnion))

#foreach ($rel in $thisRelationships)

#if ($rel.humanName.contains(“unionOf”))

#set ($unionOf = $rel.target.get(0))

#set ($consume\_lhs = $unionsOf.add($unionOf))

#end

#end

#end

## end of macro getUnionsOf

##

#macro (getOntoDatatypes $thisOnto $ontoDatatypes)

#foreach ($elem in $thisOnto.ownedElement)

#foreach($stereo in $elem.appliedStereotype)

#if ($stereo.name.contains(“rdfsDatatype”))

#set ($consume\_lhs = $ontoDatatypes.add($elem))

#end

#end

#end

#end

## end of macro getOntoDatatypes

##

#macro (getOntoEnums $thisOnto $ontoEnums)

#foreach ($elem in $thisOnto.ownedElement)

#foreach($stereo in $elem.appliedStereotype)

#if ($stereo.name.contains(“rdfsDatatype”))

#foreach ($rel in $report.getRelationship($elem))

#if ($rel.humanName.contains(“equivalentDatatype”))

#set ($equivDatatype = $rel.target.get(0))

#foreach($relStereo in $equivDatatype.appliedStereotype)

#if ($relStereo.name.contains(“DataEnumeration”))

#set ($consume\_lhs = $ontoEnums.add($elem))

#end

#end

#end

#end

#end

#end

#end

#end

## end of macro getOntoEnums

##

#macro (getEnumValues $thisEnum $enumValues)

#set ($thisRelationships = $report.getRelationship($thisEnum))

#foreach ($rel in $thisRelationships)

#if ($rel.humanName.contains(“equivalentDatatype”))

#set ($enumDatatype = $rel.target.get(0))

#foreach ($enumRel in $report.getRelationship($enumDatatype))

#if ($enumRel.humanName.contains(“oneOf”))

#set ($enumLiteral = $enumRel.target.get(0))

#set ($enumValue = $report.createValueSpecificationText($enumLiteral.specification))

#set ($consume\_lhs = $enumValues.add($enumValue))

#end

#end

#end

#end

#end

## end of macro getEnumDatatype

##

#macro (getOntoDiagrams $thisOnto $ontoDiags )

#foreach ($pkg in $thisOnto.nestedPackage)

#if ($pkg.name.contains(“Tech Diagrams”))

#set ($diagPkg = $pkg )

#foreach ($diag in $diagPkg.ownedElement)

#if ($diag.humanType.contains(“Diagram”))

#set ($consume\_lhs = $ontoDiags.add($diag))

#end

#end

#end

#end

#end

## end of the getOntoDiagrams macro

##

#import('js', 'com.nomagic.reportwizard.tools.script.JavaScriptTool')

#macro (scaleImage $unscaledImage)

#set ($consume\_lhs = $image.setScalingQuality(5))

#set ($thisImage = $unscaledImage)

#set ($width = $thisImage.width)

#set ($height = $thisImage.height)

#if ($width > $height)

#set ($scale = $js.eval(‘500/$width’))

#else

#set ($scale = $js.eval(‘500/$height’))

#end

#set ($scaledImage = $image.scale( $thisImage, $scale ))

#set ($width = $scaledImage.width)

#set ($height = $scaledImage.height)

#end

##

#macro (serializeOntoDiagrams $ontoDiags )

#if (!$report.isEmpty( $ontoDiags ))

##### Diagrams

This section presents the diagrams that illustrate the content within the $thisOntology.name ontology.

#foreach ($diag in $ontoDiags)

#scaleImage ($diag.image)

$scaledImage

Figure : $diag.name

#end

#end

#end

## end of the serializeOntoDiagrams macro

##

## Find and visit each <<owlOntology>> Package available in the supplied set of

## model Packages

##

#getOntologyPackages()

#foreach ($thisOntology in $sorter.sort($ontologyPackages))

#set ($ontoDiagrams = $array.createArray())

#getOntoDiagrams( $thisOntology $ontoDiagrams )

#### Ontology: $thisOntology.name

## look up the <<fact>> relationships for this ontology

#set ($ontoFacts = $array.createArray())

#getFacts($thisOntology $ontoFacts)

## obtain the map for Predicates by Name to Fact relationship

#set ($ontoMapPredicateFact = $map.createHashMap())

#getMapPredicateFact($ontoFacts, $ontoMapPredicateFact)

##

#serializeOntoDiagrams( $ontoDiagrams )

## Visit each Class of the current Ontology

#set ($ontologyClasses = $array.createHashSet())

#getClasses( $thisOntology, $ontologyClasses )

#if (0 < $ontologyClasses.size())

##### Classes

This section presents the OWL Classes that are directly defined within this $thisOntology.name ontology. Each such class appears in its own subsection with an elaboration of its defined properties.

#foreach ($cls in $sorter.sort($ontologyClasses))

#set ($clsFacts = $array.createArray())

#set ($clsMapPredicateFact = $map.createHashMap())

#getFacts($cls, $clsFacts)

#getMapPredicateFact($clsFacts, $clsMapPredicateFact)

###### Class: $cls.name

#getFactValueByPredicateName($clsMapPredicateFact, “definition”)

#if (!$report.isEmpty($value))

#trimAndPeriod($value)

Definition: $periodString

#else

A definition has yet to be specified for the Owl Class, $cls.name.

#end

#getExemplarDiagram($cls)

#if (!$report.isNull($exemplarDiagram))

#scaleImage ($exemplarDiagram.image)

$scaledImage

Figure : $exemplarDiagram.name

#end

## Serialize each if any Parent

#set ($clsParents = $array.createHashSet())

#getParents( $cls, $clsParents )

#serializeParents( $clsParents )

##

## Serialize each if any Disjoint

#set ($clsDisjoints = $array.createHashSet())

#getDisjoints( $cls, $clsDisjoints )

#serializeDisjoints( $clsDisjoints )

##

## Enumerate the Restrictions for this Class

#set ($clsRestrictions = $array.createArray())

#getRestrictions( $cls, $clsRestrictions )

#serializeRestrictions( $cls, $clsRestrictions )

##

## Emit a table of Datatype Properties for the Class

#set ($clsDatatypeProperties = $array.createArray())

#getClassDatatypeProperties( $cls, $clsDatatypeProperties )

#if (0 < $clsDatatypeProperties.size())

Datatype Properties

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Annotation | Type | Parent | Inverse |
| #forrow ($property in $clsDatatypeProperties)$property.get(“name”) | $property.get(“annotation”) | $property.get(“type”) | $property.get(“parent”) | $property.get(“inverse”)#endrow |

#end

##

## Emit a table of Properties for the Class

#set ($clsProperties = $array.createArray())

#getClassProperties( $cls, $clsProperties )

#if (0 < $clsProperties.size())

Object Properties

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Annotation | Type | Parent | Inverse |
| #forrow ($property in $clsProperties)$property.get(“name”) | $property.get(“annotation”) | $property.get(“type”) | $property.get(“parent”) | $property.get(“inverse”)#endrow |

#end

##

#end

#end

## UnionClasses of the Ontology

#set ($ontologyUnions = $array.createHashSet())

#getOntoUnions( $thisOntology, $ontologyUnions )

#if (0 < $ontologyUnions.size())

##### Union Classes

This section presents the UnionClass Classes that are defined within the Auxiliary Elements of this $thisOntology.name ontology. Each such UnionClass appears in its own subsection with an elaboration of its defined UnionOf Classes.

#foreach ($union in $sorter.sort($ontologyUnions))

#set ($unionFacts = $array.createArray())

#set ($unionPredicateFactMap = $map.createHashMap())

#getFacts($union, $unionFacts)

#getMapPredicateFact( $unionFacts, $unionPredicateFactMap )

#set ($unionsOf = $array.createHashSet())

#getUnionsOf( $union, $unionsOf )

#if ($report.isEmpty($union.name))

#set ($uName = “<Anonymous> Blank Node”)

#else

#set ($uName = $union.name)

#end

###### UnionClass: $uName

#getFactValueByPredicateName( $unionPredicateFactMap, “definition” )

#if (!$report.isEmpty($value))

#trimAndPeriod($value)

UnionClass, $uName, has the definition: $periodString

#else

A definition has yet to be specified for the Union Class, $uName. Perhaps it is eponymous or its definition has simply been neglected?

#end

#if (0 < $unionsOf.size())

**Is the Union Of:**

#foreach( $unionOf in $sorter.sort($unionsOf))

* $unionOf.name

#end

#end

#end

#end

##

## Visit each Datatype of the current Ontology

#set ($ontologyDatatypes = $array.createHashSet())

#getOntoDatatypes( $thisOntology, $ontologyDatatypes )

#if (0 < $ontologyDatatypes.size())

##### Datatypes

This section presents the RDFS Datatypes that are directly defined within this $thisOntology.name ontology. Each such Datatype appears in its own subsection with an elaboration of its defined properties.

#foreach ($data in $sorter.sort($ontologyDatatypes))

#set ($dataFacts = $array.createArray())

#set ($dataMapPredicateFact = $map.createHashMap())

#getFacts($data, $dataFacts)

#getMapPredicateFact($dataFacts, $dataMapPredicateFact)

###### Datatype: $data.name

#getFactValueByPredicateName($dataMapPredicateFact, “definition”)

#if (!$report.isEmpty($value))

#trimAndPeriod($value)

RDFS Datatype, $data.name, has the definition: $periodString

#else

A definition has yet to be specified for the RDFS Datatype, $data.name. Perhaps it is eponymous or its definition has simply been neglected?

#end

#end

#end

## Enumerations of the Ontology

#set ($ontologyEnums = $array.createHashSet())

#getOntoEnums( $thisOntology, $ontologyEnums )

#if (0 < $ontologyEnums.size())

##### Enumerations

This section presents the rdfsDatatype Enumerations that are directly defined within this $thisOntology.name ontology. Each such Enumeration appears in its own subsection with an elaboration of its defined properties.

#foreach ($enum in $sorter.sort($ontologyEnums))

#set ($enumFacts = $array.createArray())

#set ($enumPredicateFactMap = $map.createHashMap())

#getFacts($enum, $enumFacts)

#getMapPredicateFact( $enumFacts, $enumPredicateFactMap )

#set ($enumeratedValues = $array.createHashSet())

#getEnumValues( $enum, $enumeratedValues )

###### Enumeration: $enum.name

#getFactValueByPredicateName( $enumPredicateFactMap, “definition” )

#if (!$report.isEmpty($value))

#trimAndPeriod($value)

Enumeration, $enum.name, has the definition: $periodString

#else

A definition has yet to be specified for the Enumeration, $enum.name. Perhaps it is eponymous or its definition has simply been neglected?

#end

#if (0 < $enumeratedValues.size())

**Allowed Values**

#foreach( $enumValue in $sorter.sort($enumeratedValues))

* $enumValue

#end

#end

#end

#end

##

#set ($unsortedProperties = $array.createArray())

#getOntoProperties( $thisOntology, $unsortedProperties )

#set ($ontologyProperties = $sorter.sort($unsortedProperties))

#if (0 < $ontologyProperties.size())

##### Object Properties

This section presents the OWL object properties that are directly defined within this $thisOntology.name ontology.

#foreach ($ontoProp in $ontologyProperties )

#set ($objFacts = $array.createArray())

#set ($objMapPredicateFact = $map.createHashMap())

#getFacts($ontoProp, $objFacts)

#getMapPredicateFact($objFacts, $objMapPredicateFact)

###### Object Property: $ontoProp.name

#getFactValueByPredicateName($objMapPredicateFact, “definition”)

#if (!$report.isEmpty($value))

#trimAndPeriod($value)

Definition: $periodString

#else

A definition has yet to be specified for the Object Property, $ontoProp.name.

#end

## Emit a table of Properties for the ObjectProperty

#set ($objPropProperties = $array.createArray())

#foreach($role in $ontoProp.memberEnd)

#if ($role.navigable)

#getObjPropProperties( $ontoProp, $objPropProperties, $role.type )

#end

#end

#if (0 < $objPropProperties.size())

Properties

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Definition | Range | Parent | Inverse |
| #forrow ($property in $objPropProperties)$property.get(“name”) | $property.get(“definition”) | $property.get(“type”) | $property.get(“parent”) | $property.get(“inverse”)#endrow |

#end

#end

#end

#end

## end foreach thisOntology