Configuration Properties

Table of Contents

Chapter 1. Configuration Properties

Apache Isis' own configuration properties are simple key-value pairs, typically held in the `WEBINF/isis.properties` file and other related files. This guide describes how to configure an Apache Isis application.

Configuration properties for the viewers can be found in the Wicket Viewer guide and the RestfulObjects viewer guide. Likewise[details of configuring security (Apache Shiro) can be found in the Security guide.

Also, note that by default the configuration values are part of the built WAR file. Details on how to override these configuration properties externally for different environments can be found in the Beyond the Basics guide, (deployment chapter).

1.1. Other Guides

Apache Isis documentation is broken out into a number of user, reference and "supporting procedures" guides.

The user guides available are:

- Fundamentals
- Wicket viewer
- Restful Objects viewer
- DataNucleus object store
- Security
- Testing
- Beyond the Basics

The reference guides are:

- Annotations
- Domain Services
- Configuration Properties (this guide)
- Classes, Methods and Schema
- Apache Isis Maven plugin
- Framework Internal Services

The remaining guides are:

- Developers' Guide (how to set up a development environment for Apache Isis and contribute back to the project)
- Committers' Guide (release procedures and related practices)

Chapter 2. Deployment Types

Apache Isis distinguishes between the application being run in development mode vs running in production mode. The framework calls this the "deployment type" (corresponding internally to the `DeploymentType` class).

(For mostly historical reasons) development mode is actually called `SERVER_PROTOTYPE`, while production mode is called just `SERVER`. (There is also a deprecated mode called `SERVER_EXPLORATION`; for all intents and purposes this can considered as an alias of `SERVER_PROTOTYPE`).

When running in development/prototyping mode, certain capabilities are enabled; most notably any actions restricted to prototyping mode (using `@Action#restrictTo()`) will be available.

2.1. Using the Wicket Viewer

Most of the you're likely to run Apache Isis using the Wicket viewer. In this case Apache Isis' "deployment type" concept maps to Wicket's "configuration" concept:

Table 1. Apache Isis' deployment type corresponds to Apache Wicket's configuration

Apache Isis (Deployment Type)	Apache Wicket (Configuration)	Notes
`SERVER_PROTOTYPE`	`development`	running in development/prototyping mode
`SERVER`	`deployment`	running in production mode

Wicket's mechanism for specifying the "configuration" is to use a context parameter in `web.xml`; Apache Isis automatically infers its own deployment type from this. In other words:

- to specify SERVER (production) mode, use:

  `web.xml`

  ```
  <context-param>
      <param-name>configuration</param-name>
      <param-value>deployment</param-value>
  </context-param>
  ```

- to specify `SERVER_PROTOTYPING` (development) mode, use:

  `web.xml`

  ```
  <context-param>
      <param-name>configuration</param-name>
      <param-value>development</param-value>
  </context-param>
  ```

2.2. Restful Objects viewer only

Most Apache Isis applications will consist of at least the Wicket viewer and optionally the RestfulObjects viewer. When both viewers are deployed in the same app, then the bootstrapping is performed by Wicket, and so the deployment type is configured as described in the previous section.

In some cases though you may be using Apache Isis to provide a REST API only, that is, you won't have deployed the Wicket viewer. In these cases your app will be bootstrapped using Apache Isis' `IsisWebAppBootstrapper`.

In this case the deployment type is specified through an Apache Isis-specific context parameter, called `isis.deploymentType`:

- to specify `SERVER` (production) mode, use:

  `web.xml`

  ```
  <context-param>
      <param-name>isis.deploymentType</param-name>
      <param-value>server</param-value>
  </context-param>
  ```

- to specify `SERVER_PROTOTYPE` (development) mode, use:

  `web.xml`

  ```
  <context-param>
      <param-name>isis.deploymentType</param-name>
      <param-value>server-prototype</param-value>
  </context-param>
  ```

2.3. Overriding the deployment type

If bootstrapping the application using Apache Isis' `org.apache.isis.WebServer` then it is possible to override the deployment type using the `-t` (or `--type`) flag.

For example:

```
java -jar ... org.apache.isis.WebServer -t SERVER
```

where "..." is the (usually rather long) list of JAR files and class directories that will make up your application.

This works for both the Wicket viewer and the RestfulObjects viewer.

Chapter 3. Configuration Files

When running an Apache Isis webapp, configuration properties are read from configuration files held in the `WEB-INF` directory.

The `WEBINF/isis.properties` file is always read and must exist.

In addition, the following other properties are searched for and if present also read:

- `viewer_wicket.properties` - if the Wicket viewer is in use
- `viewer_restfulobjects.properties` - if the RestfulObjects viewer is in use
- `viewer.properties` - for any other viewer configuration (but there are none currently)
- `persistor_datanucleus.properties` - assuming the JDO/DataNucleus objectstore is in use
- `persistor.properties` - for any other objectstore configuration.

  This typically is used to hold `JDBC URL`s, which is arguably a slight violation of the file (because there's nothing in Apache Isis to say that persistors have to use `JDBC`. However, it is generally convenient to put these `JDBC` settings into a single location. If you want, they could reside inin any of `persistor_datanucleus.properties`, `persistor.properties` or (even) `isis.properties`

- `authentication_shiro.properties`, `authorization_shiro.properties`

  assuming the Shiro Security is in use (but there are no security-related config properties currently; use shiro.ini for Shiro config)

- `authentication.properties`, `authorization.properties`

  for any other security-related config properties (but there are none currently).

You can if you wish simply store all properties in the `isis.properties` file; but we think that breaking properties out into sections is preferable.

Chapter 4. Specifying components

Bootstrapping an Apache Isis application involves identifying both:

- the major components (authentication, persistence mechanisms, viewers) of Apache Isis, and also
- specifying the domain services and persistent entities that make up the application itself.

As of 1.9.0 there are two different ways to perform this bootstrapping. The recommended (newer) approach is to use an `AppManifest`, specified either programmatically or through the configuration properties. This allows the components, services and entities to be specified from a single class. The alternative (and older, pre 1.9.0) approach is to specify this information individually, through configuration properties.

To specify the `AppManifest` as a configuration property, use:

Table 2. Core Configuration Properties (ignored if `isis.appManifest` is present)

Property	**Value** (*default value*)	**Implements**
`isis.appManifest`	`FQCN`	`o.a.i.applib.AppManifest` By convention this implementation resides in an `myapp-app` Maven module (as opposed to `myapp-dom` or `myapp-fixture`). See the SimpleApp archetype for details.

From this the framework can determine the domain services, persistent entities and security (authentication and authorization) mechanisms to use. Other configuration (including fixtures) can also be specified this way.

If the `AppManifest` approach is *not* being used, then the following configuration properties are used to specify the major components of Apache Isis to use:

Table 3. Core Configuration Properties (ignored if `isis.appManifest` is present)

Property	**Value** (*default value*)	**Implements**
`isis.authentication`	`shiro`, `bypass`, `FQCN` (`shiro`)	`o.a.i.core.runtime.authentication.AuthenticationManagerInstaller` This property is IGNORED if the `isis.appManifest` configuration property is specified, or if an `AppManifest` is provided programmatically.
`isis.authorization`	`shiro`, `bypass`, `FQCN` (`shiro`)	`o.a.i.core.runtime.authorization.AuthorizationManagerInstaller` This property is IGNORED if the `isis.appManifest` configuration property is specified, or if an `AppManifest` is provided programmatically.

Property	Value (*default value*)	Implements
isis.persistor	datanucleus (datanucleus)	o.a.i.core.runtime.installerregistry.installerapi. PersistenceMechanismInstaller This property is IGNORED completely in 1.9.0+; the datanucleus implementation is always used.
isis.services-installer	configuration, configuration-and-annotation, FQCN (configuration)	org.apache.isis.core.runtime.services.ServicesInstaller The mechanism to discover and load domain services: * configuration-and-annotation will search for @DomainService-annotated classes and also read from isis.services configuration property * configuration will only read from the isis.services configuration property. * Otherwise an alternative implementation of the o.a.i.core.runtime.services.ServicesInstaller internal API can be provided. This property is IGNORED if the isis.appManifest configuration property is specified, or if an AppManifest is provided programmatically. This property is also IGNORED completely in 1.13.0+; the configuration-and-annotation implementation is always used.

The values "shiro", "bypass" etc are actually aliases for concrete implementations. It is also possible to specify a fully qualified class name to replace either of the two security components, implementing the appropriate interface.

If the AppManifest is *not* being used then there are number of other configuration properties that also must be specified: isis.services, isis.services.ServicesInstallerFromAnnotation.packagePrefix and isis.persistor.datanucleus.RegisterEntities.packagePrefix and isis.fixtures; these are listed in the sections below.

4.1. Viewer Configuration

Viewers are specified by way of the filters and servlets in the web.xml file; these are not bootstrapped by the framework, rather it is the other way around.

In versions prior to 1.13.0, the "isis.viewers" context parameter was used to hint which configuration files should be read (corresponding to the viewers in use). As of 1.13.0, however, the configuration property has no effect: the viewer_wicket.properties and viewer_restulobjects.properties are always loaded if available.

Chapter 5. Configuring Core

This section lists the core/runtime configuration properties recognized by Apache Isis.

Configuration properties for the JDO/DataNucleus objectstore can be found in the Configuring DataNucleus section later in this chapter, while configuration properties for the viewers can be found in the their respective chapters, here for Wicket viewer, and here for the Restful Objects viewer.

5.1. Domain Events

Table 4. Core Configuration Properties for Domain Events

Property	Value (default value)	Description
`isis.reflector.facet.actionAnnotation.domainEvent.postForDefault`	`true`,`false` (`true`)	Whether an event should be posted if `@Action#domainEvent()` is not specified (is set to `ActionDomainEvent.Default`).
`isis.reflector.facet.collectionAnnotation.domainEvent.postForDefault`	`true`,`false` (`true`)	Whether an event should be posted if `@Collection#domainEvent()` is not specified (is set to `CollectionDomainEvent.Default`).
`isis.reflector.facet.propertyAnnotation.domainEvent.postForDefault`	`true`,`false` (`true`)	Whether an event should be posted if `@Property#domainEvent()` is not specified (is set to `PropertyDomainEvent.Default`).

5.2. Lifecycle Events

Table 5. Core Configuration Properties for Lifecycle Events

Property	Value (default value)	Description
`isis.reflector.facet.domainObjectAnnotation.createdLifecycleEvent.postForDefault`	`true`,`false` (`true`)	Whether an event should be posted if `@DomainObject#createdLifecycleEvent()` is not specified (is set to `ObjectCreatedEvent.Default`).
`isis.reflector.facet.domainObjectAnnotation.loadedLifecycleEvent.postForDefault`	`true`,`false` (`true`)	Whether an event should be posted if `@DomainObject#loadedLifecycleEvent()` is not specified (is set to `ObjectLoadedEvent.Default`).
`isis.reflector.facet.domainObjectAnnotation.persistingLifecycleEvent.postForDefault`	`true`,`false` (`true`)	Whether an event should be posted if `@DomainObject#persistingLifecycleEvent()` is not specified (is set to `ObjectPersistingEvent.Default`).

Property	Value (default value)	Description
isis.reflector.facet.domainObjectAnnotation.persistedLifecycleEvent.postForDefault	true,false (true)	Whether an event should be posted if @DomainObject#persistedLifecycleEvent() is not specified (is set to ObjectPersistedEvent.Default).
isis.reflector.facet.domainObjectAnnotation.removingLifecycleEvent.postForDefault	true,false (true)	Whether an event should be posted if @DomainObject#removingLifecycleEvent() is not specified (is set to ObjectRemovingEvent.Default).
isis.reflector.facet.domainObjectAnnotation.updatingLifecycleEvent.postForDefault	true,false (true)	Whether an event should be posted if @DomainObject#updatingLifecycleEvent() is not specified (is set to ObjectUpdatingEvent.Default).
isis.reflector.facet.domainObjectAnnotation.updatedLifecycleEvent.postForDefault	true,false (true)	Whether an event should be posted if @DomainObject#updatedLifecycleEvent() is not specified (is set to ObjectUpdatedEvent.Default).

5.3. UI Events

Table 6. Core Configuration Properties for UI Events

Property	Value (default value)	Description
isis.reflector.facet.domainObjectLayoutAnnotation.cssClassUiEvent.postForDefault	true,false (true)	Whether an event should be posted if @DomainObjectLayout#cssClassUiEvent() is not specified (is set to CssClassUiEvent.Default).
isis.reflector.facet.domainObjectLayoutAnnotation.iconUiEvent.postForDefault	true,false (true)	Whether an event should be posted if @DomainObjectLayout#iconUiEvent() is not specified (is set to IconUiEvent.Default).
isis.reflector.facet.domainObjectLayoutAnnotation.titleUiEvent.postForDefault	true,false (true)	Whether an event should be posted if @DomainObjectLayout#titleUiEvent() is not specified (is set to TitleUiEvent.Default).

5.4. Services

Table 7. Core Configuration Properties for Services

Property	Value (default value)	Description
isis.services	FQCN,FQCN2,...	Fully qualified class names of classes to be instantiated as domain services. Each entry can be optionally prefixed by "n:" specifying the relative order on the menu (corresponds to @DomainServiceLayout#menuOrder()). This property is IGNORED if the isis.appManifest configuration property is specified, or if an AppManifest is provided programmatically.
isis.services.applicationFeatures.init	lazy, eager (lazy)	Whether the application features repository (which surfaces the framework's metamodel) should be initialized lazily or eagerly. Lazy initialization can speed up bootstrapping, useful while developing and running tests. The default prior to 1.13.0 was eager initialization.
isis.services.audit.objects	all, none (all)	Whether the changed properties of objects should be automatically audited (for objects annotated with @DomainObject(auditing=Auditing.AS_CONFIGURED).
isis.services.command.actions	all, ignoreSafe, none (all)	Whether action invocations should be automatically reified into commands (for actions annotated with @Action(command=CommandReification.AS_CONFIGURED). ignoreQueryOnly is an alias for ignoreSafe.
isis.services.command.properties	all, none (all)	(Whether property edits should be automatically reified into commands (for properties annotated with @Property(command=CommandReification.AS_CONFIGURED).
isis.services.container.disableAutoFlush	true,false (false)	Whether the DomainObjectContainer should automatically flush pending changes prior to querying (via allMatches(), firstMatch() and so on).
isis.services.ContentNegotiationServiceXRoDomainType.prettyPrint	true,false (depends)	If a domain object has been mapped to the specified JAXB x-ro-domain-type, then determines whether the result is pretty-printed or not. + If no configuration property is available, then the defaults is determined by the deployment type: production mode disables pretty printing, while prototype mode enables it.

Property	Value (default value)	Description
isis.service.email.tls.enabled	true,false (true)	Whether to enable TLS for the email SMTP connection (used by EmailService). NB: note that the key is mis-spelt, (isis.service.email rather than isis.services.email)
isis.service.email.sender.hostname	host (smtp.gmail.com)	The hostname of the external SMTP provider (used by EmailService). NB: note that the key is mis-spelt, (isis.service.email rather than isis.services.email)
isis.service.email.port	port number (587)	The port number for the SMTP service on the the external SMTP host (used by EmailService). NB: note that the key is mis-spelt, (isis.service.email rather than isis.services.email)
isis.service.email.sender.address	email address	The email address to use for sending out email (used by EmailService). **Mandatory**. NB: note that the key is mis-spelt, (isis.service.email rather than isis.services.email)
isis.service.email.sender.password	email password	The corresponding password for the email address to use for sending out email (used by EmailService). **Mandatory**. NB: note that the key is mis-spelt, (isis.service.email rather than isis.services.email)
isis.services.eventbus.implementation	guava, axon, FQCN (guava)	which implementation to use by the EventBusService as the underlying event bus.
isis.services.eventbus.allowLateRegistration	true,false (false)	whether a domain service can register with the EventBusService after any events have posted. Since this almost certainly constitutes a bug in application code, by default this is disallowed.
isis.services.exceprecog.logRecognizedExceptions	true,false (false)	whether recognized exceptions should also be logged. Generally a recognized exception is one that is expected (for example a uniqueness constraint violated in the database) and which does not represent an error condition. This property logs the exception anyway, useful for debugging.

Property	Value (default value)	Description
isis.services.ExceptionRecognizerComposite-ForJdoObjectStore.disable	true,false (false)	whether to disable the default recognizers registered by ExceptionRecognizerCompositeForJdoObjectStore. This implementation provides a default set of recognizers to convert RDBMS constraints into user-friendly messages. In the (probably remote) chance that this functionality isn't required, they can be disabled through this flag.
isis.services.injector.injectPrefix	true,false (false)	(Whether the framework should support inject…() as a prefix for injecting domain services into other domain objects. + By default this is disabled. The default prior to 1.13.0 was enabled. If the setting is left as disabled then this may reduce application start-up times.
isis.services.injector.setPrefix	true,false (true)	Whether the framework should support set…() as a prefix for injecting domain services into other domain objects. + By default this is enabled (no change in 1.13.0). If the setting is changed to disabled then this may reduce application start-up times.
isis.services.publish.objects	all, none (all)	Whether changed objects should be automatically published (for objects annotated with @DomainObject(publishing=Publishing.AS_CONFIGURED).
isis.services.publish.actions	all, ignoreSafe, none (none)	Whether actions should be automatically published (for actions annotated with @Action(publishing=Publishing.AS_CONFIGURED).
isis.services.publish.properties	all, none (none)	Whether properties should be automatically published (for properties annotated with @Property(publishing=Publishing.AS_CONFIGURED).
isis.services.ServicesInstallerFromAnnotation.packagePrefix	fully qualified package names (CSV)	to search for domain services (including all subpackages). This property is IGNORED if the isis.appManifest configuration property is specified, or if an AppManifest is provided programmatically.
isis.services.translation.po.mode	read,write	Whether to force the TranslationService into either read or write mode. See i18n support to learn more about the translation service.

5.5. Other Config Properties

Table 8. Other Core Configuration Properties

Property	Value (default value)	Description
`isis.objects.editing`	`true`,`false` (`true`)	Whether objects' properties and collections can be edited directly (for objects annotated with `@DomainObject#editing()`); see below for further discussion.
`isis.persistor.disableConcurrencyChecking`	`true`,`false` (`false`)	Disables concurrency checking globally. Only intended for "emergency use" as a workaround while pending fix/patch to Apache Isis itself. (Note that there is no "datanucleus" in the property).
`isis.reflector.facet.cssClass.patterns`	regex:css1, regex2:css2,...	Comma separated list of key:value pairs, where the key is a regex matching action names (eg `delete.*`) and the value is a Bootstrap CSS button class (eg `btn-warning) to be applied (as per` `@CssClass()`) to all action members matching the regex. See UI hints for more details.
`isis.reflector.facet.cssClassFa.patterns`	regex:fa-icon,regex2:fa-icon2,...	Comma separated list of key:value pairs, where the key is a regex matching action names (eg `create.*`) and the value is a font-awesome icon name (eg `fa-plus`) to be applied (as per `@CssClassFa()`) to all action members matching the regex. See UI hints for more details.
`isis.reflector.facet.filterVisibility`	`true`,`false` (`true`)	Whether objects should be filtered for visibility. See section below for further discussion.
`isis.reflector.facets`	`FQCN`	This property is now ignored. + To customize the programming model, use `facets.exclude` and `facets.include`. See finetuning the programming model for more details.
`isis.reflector.facets.exclude`	`FQCN`,`FQCN2`,...	Fully qualified class names of (existing, built-in) facet factory classes to be included to the programming model. See finetuning the programming model for more details.

Property	Value (default value)	Description
isis.reflector.facets.ignoreDeprecated	true,false (false)	Whether deprecated facets should be ignored or honoured. + By default all deprecated facets are honoured; they remain part of the metamodel. If instead this property is set to true then the facets are simply not loaded into the metamodel and their semantics will be excluded. + In most cases this should reduce the start-up times for the application. However, be aware that this could also substantially alter the semantics of your application. To be safe, we recommend that you first run your application using isis.reflector.validator.allowDeprecated set to false; if any deprecated annotations etc. are in use, then the app will fail-fast and refuse to start.
isis.reflector.facets.include	FQCN,FQCN2,...	Fully qualified class names of (new, custom) facet factory classes to be included to the programming model. See finetuning the programming model for more details.
isis.reflector.layoutMetadataReaders	FQCN,FQCN2,...	Fully qualified class names of classes to be instantiated to read layout metadata, as used in for dynamic layouts. See Layout Metadata Reader for more information.
isis.reflector.validator	FQCN	Custom implementation of MetaModelValidator (in the org.apache.isis.core.metamodel.specloader.validator package) See Custom Validator to learn more.
isis.reflector.validator.allowDeprecated	true,false (true)	Whether deprecated annotations or naming conventions are tolerated or not. If not, then a metamodel validation error will be triggered, meaning the app won't boot (fail-fast). + See also isis.reflector.facets.ignoreDeprecated.
isis.viewers.paged.parented	positive integer (12)	Default page size for parented collections (as owned by an object, eg Customer#getOrders())
isis.viewers.paged.standalone	positive integer (25)	Default page size for standalone collections (as returned from an action invocation)
isis.viewers.propertyLayout.labelPosition	TOP, LEFT (LEFT)	Default for label position for all properties if not explicitly specified using @PropertyLayout#labelPosition()

5.5.1. Filtering visibility

The framework provides the `isis.reflector.facet.filterVisibility` configuration property that influences whether a returned object is visible to the end-user:

- Action invocations:

If an action returns a collection that includes the object, then the object will be excluded from the list when rendered. If it returns a single object and the user does not have access to that object, then the action will seemingly return `null`

- Collections:

If a parent object has a collection references another object to which the user does not have access, then (as for actions) the object will not be rendered in the list

- Properties:

If an parent object has a (scalar) reference some other object to which the user does not have access, then the reference will be rendered as empty.

- Choices and autoComplete lists:

If an object is returned in a list of choices or within an auto-complete list, and the user does not have access, then it is excluded from the rendered list.

The original motivation for this feature was to transparently support such features as multi-tenancy (as per the (non-ASF) Isis addons' security module). That is, if an entity is logically "owned" by a user, then the multi-tenancy support can be arranged to prevent some other user from viewing that object.

By default this configuration property is enabled. To disable the visibility filtering, set the appropriate configuration property to `false`:

```
isis.reflector.facet.filterVisibility=false
```

Filtering is supported by the Wicket viewer and the Restful Objects viewer, and also by the `WrapperFactory` domain service (provided the wrapper's execution mode is *not* "skip rules").

In order for the framework to perform this filtering of collections, be aware that the framework takes a *copy* of the original collection, filters on the collection, and returns that filtered collection rather than the original.

There are no major side-effects from this algorithm, other than the fact that the referenced objects will (most likely) need to be resolved in order to determine if they are visible. This could conceivably have a performance impact in some cases.

5.5.2. objects.editing

This configuration property in effect allows editing to be disabled globally for an application:

```
isis.objects.editing=false
```

We recommend enabling this feature; it will help drive out the underlying business operations (processes and procedures) that require objects to change; these can then be captured as business actions.

5.5.3. propertyLayout.labelPosition

If you want a consistent look-n-feel throughout the app, eg all property labels to the top, then it'd be rather frustrating to have to annotate every property.

Instead, a default can be specified in `isis.properties`:

```
isis.viewers.propertyLayout.labelPosition=TOP
```

or

```
isis.viewers.propertyLayout.labelPosition=LEFT
```

If these are not present then Apache Isis will render according to internal defaults. At the time of writing, this means labels are to the left for all datatypes except multiline strings.

Chapter 6. Configuring DataNucleus

Apache Isis programmatically configures DataNucleus; any Apache Isis properties with the prefix `isis.persistor.datanucleus.impl` are passed through directly to the JDO/DataNucleus objectstore (with the prefix stripped off, of course).

DataNucleus will for itself also and read the `META-INF/persistence.xml`; at a minimum this defines the name of the "persistence unit". n theory it could also hold mappings, though in Apache Isis we tend to use annotations instead.

Furthermore, DataNucleus will search for various other XML mapping files, eg `mappings.jdo`. A full list can be found here. The metadata in these XML can be used to override the annotations of annotated entities; see Overriding JDO Annotatons for further discussion.

6.1. Configuration Properties

These configuration properties are typically stored in `WEB-INF/persistor_datanucleus.properties`. However, you can place all configuration properties into `WEB-INF/isis.properties` if you wish (the configuration properties from all config files are merged together).

6.1.1. Configuration Properties for Apache Isis itself

Table 9. JDO/DataNucleus Objectstore Configuration Properties

Property	Value (*default value*)	Description
`isis.persistor.datanucleus.classMetadataLoadedListener`	`FQCN`	The default (`o.a.i.os.jdo.dn.CreateSchemaObjectFromClassMetadata`) creates a DB schema object
`isis.persistor.datanucleus.RegisterEntities.packagePrefix`	fully qualified package names (CSV)	that specifies the entities early rather than allow DataNucleus to find the entities lazily. Further discussion below. This property is IGNORED if the `isis.appManifest` configuration property is specified, or if an `AppManifest` is provided programmatically.
`isis.persistor.datanucleus.PublishingService.serializedForm`	zipped	

6.1.2. Configuration Properties passed through directly to DataNucleus.

Table 10. JDO/DataNucleus Objectstore Configuration Properties

Property	Value (*default value*)	Description
`isis.persistor.datanucleus.impl.*`		Passed through directly to Datanucleus (with `isis.persistor.datanucleus.impl` prefix stripped)

Property	Value (*default value*)	Description
`isis.persistor.datanucleus.impl.datanucleus.persistenceByReachabilityAtCommit`	`false`	We recommend this setting is disabled. Further discussion below.

6.2. `persistence.xml`

TODO

6.3. Eagerly Registering Entities

Both Apache Isis and DataNucleus have their own metamodels of the domain entities. Apache Isis builds its metamodel by walking the graph of types of the domain services. The JDO/DataNucleus objectstore then takes these types and registers them with DataNucleus.

In some cases, though, not every entity type is discoverable from the API of the service actions. This is especially the case if you have lots of subtypes (where the action method specifies only the supertype). In such cases the Isis and JDO metamodels is built lazily, when an instance of that (sub)type is first encountered.

Apache Isis is quite happy for the metamodel to be lazily created, and - to be fair - DataNucleus also works well in most cases. In some cases, though, we have found that the JDBC driver (eg HSQLDB) will deadlock if DataNucleus tries to submit some DDL (for a lazily discovered type) intermingled with DML (for updating). In any case, it's probably not good practice to have DataNucleus work this way.

The framework thus provide mechanisms to search for all `@PersistenceCapable` entities under specified package(s), and registers them all eagerly. In fact there are two:

- as of 1.9.0 the recommended (and simpler) approach is to specify an `AppManifest`, either as a `isis.appManifest` configuration property or programmatically.
- for earlier versions the `isis.persistor.datanucleus.RegisterEntities.packagePrefix` configuration property can be specified. To bootstrap as a webapp this is usually specified in `persistor_datanucleus.properties`. (This is also supported in 1.9.0 if no `AppManifest` is specified. For integration testing this can be specified programatically.

Further discussion on specifying the package(s) in integration testing (for either approach) can be found in the user guide.

6.4. Persistence by Reachability

By default, JDO/DataNucleus supports the concept of persistence-by-reachability. That is, if a non-persistent entity is associated with an already-persistent entity, then DataNucleus will detect this and will automatically persist the associated object. Put another way: there is no need to call Apache Isis' `DomainObjectContainer#persist(.)` or `DomainObjectContainer#persistIfNotAlready(.)`

methods.

However, convenient though this feature is, you may find that it causes performance issues.

DataNucleus' persistence-by-reachability may cause performance issues. We strongly recommend that you disable it.

One scenario in particular where this performance issues can arise is if your entities implement the `java.lang.Comparable` interface, and you have used Apache Isis' `ObjectContracts` utility class. The issue here is that `ObjectContracts` implementation can cause DataNucleus to recursively rehydrate a larger number of associated entities. (More detail below).

We therefore recommend that you disable persistence-by-reachability by adding the following to `persistor_datanucleus.properties`:

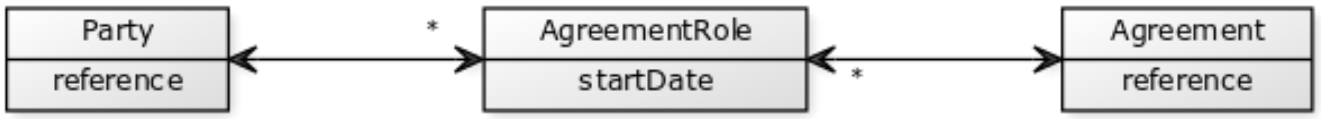
```
isis.persistor.datanucleus.impl.datanucleus.persistenceByReachabilityAtCommit=false
```

This change has been made to the SimpleApp archetype

If you do disable this feature, then you will (of course) need to ensure that you explicitly persist all entities using the `DomainObjectContainer#persist(.)` or `DomainObjectContainer#persistIfNotAlready(.)` methods.

6.4.1. The issue in more detail

Consider these entities (yuml.me/b8681268):



In the course of a transaction, the `Agreement` entity is loaded into memory (not necessarily modified), and then new `AgreementRole`s are associated to it.

All these entities implement `Comparable` using `ObjectContracts`, and the implementation of `AgreementRole`'s (simplified) is:

```
public class AgreementRole {
    ...
    public int compareTo(AgreementRole other) {
        return ObjectContracts.compareTo(this, other, "agreement","startDate",
"party");
    }
}
```

while `Agreement`'s is implemented as:

```
public class Agreement {
    ...
    public int compareTo(Agreement other) {
        return ObjectContracts.compareTo(this, other, "reference");
    }
}
```

and `Party`'s is similarly implemented as:

```
public class Party {
    ...
    public int compareTo(Party other) {
        return ObjectContracts.compareTo(this, other, "reference");
    }
}
```

DataNucleus's persistence-by-reachability algorithm adds the `AgreementRole` instances into a `SortedSet`, which causes `AgreementRole#compareTo()` to fire:

- the evaluation of the "agreement" property delegates back to the `Agreement`, whose own `Agreement#compareTo()` uses the scalar `reference` property. As the `Agreement` is already in-memory, this does not trigger any further database queries
- the evaluation of the "startDate" property is just a scalar property of the `AgreementRole`, so will already in-memory
- the evaluation of the "party" property delegates back to the `Party`, whose own `Party#compareTo()` requires the uses the scalar `reference` property. However, since the `Party` is not yet in-memory, using the `reference` property triggers a database query to "rehydrate" the `Party` instance.

In other words, in figuring out whether `AgreementRole` requires the persistence-by-reachability algorithm to run, it causes the adjacent associated entity `Party` to also be retrieved.

6.5. Using JNDI DataSource

Isis' JDO objectstore can be configured either to connect to the database using its own connection pool, or by using a container-managed datasource.

6.5.1. Application managed

Using a connection pool managed directly by the application (that is, by Apache Isis' JDO objectstore and ultimately by DataNucleus) requires a single set of configuration properties to be specified.

In the `WEB-INF\persistor_datanucleus.properties` file, specify the connection driver, url, username and password.

For example:

```
isis.persistor.datanucleus.impl.javax.jdo.option.ConnectionDriverName=net.sf.log4jdbc.
DriverSpy
isis.persistor.datanucleus.impl.javax.jdo.option.ConnectionURL=jdbc:log4jdbc:hsqldb:me
m:test
isis.persistor.datanucleus.impl.javax.jdo.option.ConnectionUserName=sa
isis.persistor.datanucleus.impl.javax.jdo.option.ConnectionPassword=
```

Those configuration properties that start with the prefix `isis.persistor.datanucleus.impl.` are passed through directly to DataNucleus (with the prefix removed).

6.5.2. Container managed (JNDI)

Using a datasource managed by the servlet container requires three separate bits of configuration.

Firstly, specify the name of the datasource in the `WEB-INF\persistor_datanucleus.properties` file. For example:

If connection pool settings are also present in this file, they will simply be ignored. Any other configuration properties that start with the prefix `isis.persistor.datanucleus.impl.` are passed through directly to DataNucleus (with the prefix removed).

Secondly, in the `WEB-INF/web.xml`, declare the resource reference:

```
<resource-ref>
    <description>db</description>
    <res-ref-name>jdbc/simpleapp</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
</resource-ref>
```

Finally, declare the datasource as required by the servlet container. For example, if using Tomcat 7, the datasource can be specified by adding the following to `$TOMCAT_HOME/conf/context.xml`:

```
<Resource name="jdbc/simpleapp"
  auth="Container"
  type="javax.sql.DataSource"
  maxActive="100"
  maxIdle="30"
  maxWait="10000"
  username="sa"
  password="p4ssword"
  driverClassName="com.microsoft.sqlserver.jdbc.SQLServerDriver"
  url="jdbc:sqlserver://127.0.0.1:1433;instance=.;databaseName=simpleapp"/>
```

You will also need to make sure that the JDBC driver is on the servlet container's classpath. For Tomcat, this means copying the driver to `$TOMCAT_HOME/lib`.

According to Tomcat's documentation, it is supposedly possible to copy the `conf/context.xml` to the name of the webapp, eg `conf/mywebapp.xml`, and scope the connection to that webapp only. I was unable to get this working, however.