Core Config’n 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.

This guide covers only the core configuration properties (relating to Apache Isis' metamodel and runtime management). 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, and details for configuring the DataNucleus Object Store can be found in the DataNucleus guide.

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.

The recommended 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.

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.

In the `AppManifest` itself, there are two methods which specify how authentication and authorisation are configured:

```
public interface AppManifest {
    ...
    String getAuthenticationMechanism();
    String getAuthorizationMechanism();
    ...
}
```

These can return either:

- "shiro" - enable integration with Apache Shiro, as described in the security user guide
- "bypass" - bypass security (in effect, configure a no-op implementation that allows everything).

Note that these 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.

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.

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 3. 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`).

In order for these events to fire the action/collection/propert must, at least, be configured with the relevant annotation (even if no attributes on that annotation are set).

5.2. Lifecycle Events

Table 4. 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`).

Property	Value (default value)	Description
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).
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).

In order for these events to fire the class must be annotated using `@DomainObject` (even if no attributes on that annotation are set).

5.3. UI Events

Table 5. 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).

In order for these events to fire the class must be annotated using `@DomainObjectLayout` (even if no attributes on that annotation are set).

5.4. Services

Table 6. Core Configuration Properties for Services

Property	Value (default value)	Description
`isis.services`	`FQCN`,`FQCN2`,...	NO LONGER REQUIRED; replaced by `AppManifest`. (It used to define the list of fully qualified class names of classes to be instantiated as domain services; this is now inferred from the list of modules provided to the app manifest).
`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` (`none`)	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` (`none`)	(Whether property edits should be automatically reified into commands (for properties annotated with `@Property(command=CommandReification.AS_CONFIGURED)`.
`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. This can help 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)`.

Property	Value (default value)	Description
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)	NO LONGER REQUIRED; replaced by AppManifest. (It used to define the list of packages to search for domain services; ; this is now inferred from the list of modules provided to the app manifest).

5.5. MetaModel Validation

Table 7. Metamodel Validation

Property	Value (default value)	Description
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.actionCollectionParameterChoices	true,false (true)	Whether to check that collection action parameters have a corresponding choices or autoComplete facet. In the current implementation such a facet is always required, so this configuration option has only been introduced as a feature flag in case it needs to be disabled for some reason.
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.reflector.validator.checkModuleExtent	true,false (true)	Whether to check that all domain objects discovered reside under the top-level module of the app manifest. Note that the application must be bootstrapped using an AppManifest2.

Property	Value (default value)	Description
isis.reflector.validator.ensureUniqueObjectTypes	true,false (true)	Whether to ensure that all classes in the metamodel map to a different object type (typically either as explicitly specified using @DomainObject(objectType=⋯), or their class name as a fallback).
isis.reflector.validator.explicitObjectType	true,false (false)	Whether to check that the class has an object type explicitly specified somehow. The object type is used by the framework as an alias for the object's concrete class; it is one part of the object's OID and can be seen in the URLs of the Wicket viewer and Restful Objects viewer, and is encoded in the Bookmarks returned by the BookmarkService. In this was it may also be persisted, for example in polymorphic associations or command or auditing tables. If the object type is not specified explicitly, then this can cause data migration issues if the class is subsequently refactored (eg renamed, or moved to a different package). This configuration property can be used to enforce a rule that the object type must always be specified (for persistent entities and view models).
isis.reflector.validator.jaxbViewModelNotAbstract	true,false (true)	Ensures that all JAXB view models are not abstract (so can be instantiated).
isis.reflector.validator.jaxbViewModelNotInnerClass	true,false (true)	Ensures that all JAXB view models are not inner classes (so can be instantiated).
isis.reflector.validator.jaxbViewModelNoArgConstructor	true,false (false)	Ensures that all JAXB view models have a public no-arg constructor. This isn't actually required (hence not enabled by default) but is arguably good practice.
isis.reflector.validator.jaxbViewModelReferenceTypeAdapter	true,false (true)	Ensures that for all JAXB view models with properties that reference persistent entities, that those entities are annotated with @XmlJavaTypeAdapter.
isis.reflector.validator.jaxbViewModelDateTimeTypeAdapter	true,false (true)	Ensures that for all JAXB view models with properties that are dates or times, that those properties are annotated with @XmlJavaTypeAdapter.

Property	Value (default value)	Description
isis.reflector.validator.jdoqlFromClause	true,false (true)	Whether to check that the class name in JDOQL FROM clause matches or is a supertype of the class on which it is annotated. Only "SELECT" queries are validated; "UPDATE" queries etc are simply ignored.
isis.reflector.validator.jdoqlVariablesClause	true,false (true)	Whether to check that the class name in JDOQL VARIABLES clause is a recognized class. Note that although JDOQL syntax supports multiple VARIABLES classes, currently the validator only checks the first class name found.
isis.reflector.validator.mixinsOnly	true,false (false)	Mixins provide a simpler programming model to contributed domain services. If enabled, this configuration property will treat any contributed service as invalid. This is by way of possibly deprecating and eventually moving contributed services from the Apache Isis programming model.
isis.reflector.validator.noParamsOnly	true,false (false)	When searching for disableXxx() or hideXxx() methods, whether to search only for the no-param version (or also for supporting methods that match the parameter types of the action). If enabled then will not search for supporting methods with the exact set of arguments as the method it was supporting (and any supporting methods that have additional parameters will be treated as invalid). Note that this in effect means that mixins must be used instead of contributed services.
isis.reflector.validator.serviceActionsOnly	true,false (false)	Domain services are stateless (at least conceptually) and so should not have any properties or collections; any that are defined will not be rendered by the viewers. If enabled, this configuration property will ensure that domain services only declare actions.

Also:

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.

5.6. UI Facet Config Properties

Table 8. UI Facet Configuration Properties

Property	Value (default value)	Description
`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.

5.7. Programming Model

Table 9. Programming Model

Property	Value (default value)	Description
isis.reflector.facets	FQCN	This property is now IGNORED. It was previously used to customize the programming model, this should now be done using 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.
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 file-based layouts. See Layout Metadata Reader for more information.

5.8. Policy

Table 10. Runtime Policy 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.reflector.explicitAnnotations.action	true,false (false)	Whether action methods need to be explicitly annotated using @Action. The default is that any non-@Programmatic methods that are not otherwise recognised as properties, collections or supporting methods, are assumed to be actions. Setting this property reverses this policy, effectively requiring that all actions need to be annotated with @Action. Note that properties and collections are still implicitly inferred by virtue of being "getters".

Property	Value (default value)	Description
`isis.reflector.facet.filterVisibility`	`true`,`false` (`true`)	Whether objects should be filtered for visibility. See section below for further discussion.

5.8.1. 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) Incode Platform's 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.

5.8.2. `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.