Framework Internal Services

Table of Contents

Chapter 1. Framework Internal Services

This guide documents a number of domain services that are not part of the framework's formal API, they use classes that are outside of the applib. They should be thought of as part of the internal design of the framework, and are liable to change from release to release.

We do not guarantee that semantic versioning will be honoured for these services.

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
- Security
- Testing
- Beyond the Basics

The reference guides are:

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

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. Presentation Layer

These domain services are internal to the framework, controlling various aspects of the presentation layer.

The table below summarizes the presentation layer internal SPIs defined by Apache Isis. It also lists their corresponding implementation, either a default implementation provided by Apache Isis itself, or provided by one of the (non-ASF) Isis Addons modules.

Table 1. Internal Services

SPI	Maven Module Impl'n (g: a:)	Implementation	Notes
`o.a.i.v.ro. rendering.service.conneg. ContentNegotiationService`	Encodes the algorithm that delegates to any registered `ContentMappingService`s.	`ContentNegotiationService-XRoDomainType` `o.a.i.core` `isis-core-viewer-restfulobjects-rendering`	
`o.a.i.v.ro. rendering.service. RepresentationService`	Generates the representations, delegating to any registered `ContentNegotiationService`s.	`RepresentationService-ForRestfulObjects` `o.a.i.core` `isis-core-viewer-restfulobjects-rendering`	

Key:

- `o.a.i` is an abbreviation for `org.apache.isis`
- `o.ia.m` is an abbreviation for `org.isisaddons.module`
- `o.a.i.c.m.s` is an abbreviation for `org.apache.isis.core.metamodel.services`
- `o.a.i.c.r.s` is an abbreviation for `org.apache.isis.core.runtime.services`
- `o.a.i.v.ro` is an abbreviation for `org.apache.isis.viewer.restfulobjects`

2.1. `ContentNegotiationService`

The `ContentNegotiationService` is a plug-in point for the RestfulObjects viewer so that it can generate representations according to HTTP `Accept` header of the request. This idea is discussed in section 34.1 of the Restful Objects spec v1.0.

The principal motivation is to allow more flexible representations to be generated for REST clients that (perhaps through their use of a certain Javascript library, say) expect, or at least works best with, a certain style of representation.

Another use case is to support "third party" REST clients over which you have no control. In this scenario you *must not* naively expose entities through the RO viewer, because over time those entities will inevitably evolve and change their structure. If the entities were exposed directly then those REST clients will break.

Instead you need to create some sort of stable facade over your domain entities, one which you will preserve even if the domain entities change. There are three ways in which you can do this:

- first is to solve the problem at the domain layer by defining a regular Apache Isis view model. This is then surfaced over the RO viewer.

  If the underlying entities change, then care must be taken to ensure that structure of the view model nevertheless is unchanged.

- a second option is to solve the problem at the persistence layer, but defining a (SQL) view in the database and then mapping this to a (read-only) entity. Again this is surfaced by the RO viewer.

  If the underlying tables change (as the result of a change in their corresponding domain entities) then once more the view must be refactored so that it still presents the same structure.

- our third option is to solve the problem at the presentation layer, using the `ContentNegotiationService` described in this section.

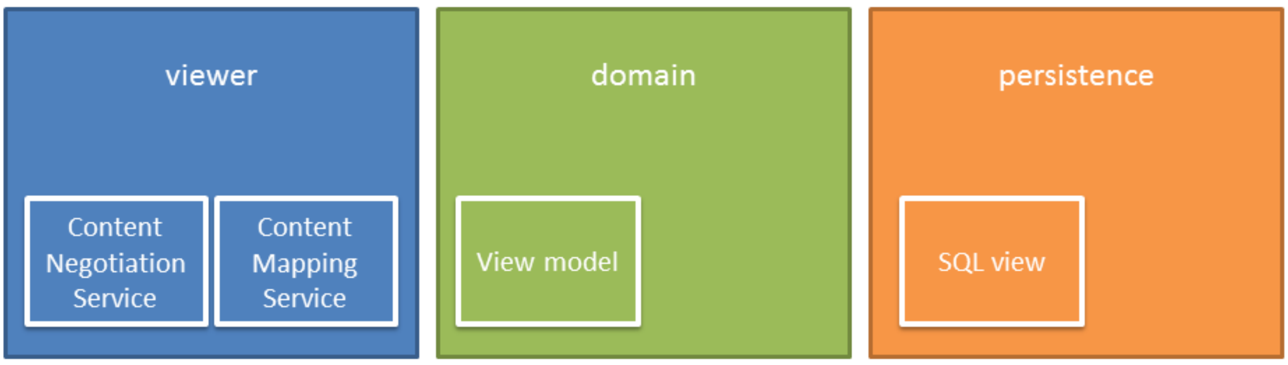
  The `ContentNegotiationService` is responsible for inspecting the HTTP `Accept` header, and use this to select the correct representation to render.

  The Apache Isis framework provides a default implementation of `ContentNegotiationService` which inspects the "x-ro-domaintype" component of the HTTP `Accept` header. If present, this implementation will delegate to the companion `ContentMappingService` service, if configured.

  A typical implementation of `ContentMappingService` will convert the domain object into some sort of DTO (data transfer object) as specified by the "x-ro-domaintype". If this DTO is annotated with JAXB or Jackson mappings, then the RO viewer (courtesy of the underlying RestEasy framework) can serialize these directly

  What all that means is that, if the underlying entities change, we are required to update the mappings in the `ContentMappingService` to map to the same DTOs.

This diagram illustrates the three options available:



2.1.1. SPI

The SPI defined by this service is:

```
public interface ContentNegotiationService {
    @Programmatic
    Response.ResponseBuilder buildResponse(                           ①
            RepresentationService.Context2 renderContext2,
            ObjectAdapter objectAdapter);
    @Programmatic
    Response.ResponseBuilder buildResponse(                           ②
            RepresentationService.Context2 renderContext2,
            ObjectAndProperty objectAndProperty);
    @Programmatic
    Response.ResponseBuilder buildResponse(                           ③
            RepresentationService.Context2 renderContext2,
            ObjectAndCollection objectAndCollection);
    @Programmatic
    Response.ResponseBuilder buildResponse(                           ④
            RepresentationService.Context2 renderContext2,
            ObjectAndAction objectAndAction);
    @Programmatic
    Response.ResponseBuilder buildResponse(                           ⑤
            RepresentationService.Context2 renderContext2,
            ObjectAndActionInvocation objectAndActionInvocation);
}
```

① representation of a single object, as per section 14.4 of the RO spec, v1.0

② representation of a single property of an object, as per section 16.4 of the RO spec v1.0

③ representation of a single collection of an object, as per section 17.5 of the RO spec v1.0

④ representation of a single action (prompt) of an object, as per section 18.2 of the RO spec v1.0

⑤ representation of the results of a single action invocation, as per section 19.5 of the RO spec v1.0

These methods provide:

- a `RepresentationService.Context2` which provides access to request-specific context (eg HTTP headers), session-specific context (eg authentication) and global context (eg configuration settings)
- an object representing the information to be rendered

  eg `ObjectAdapter`, `ObjectAndProperty`, `ObjectAndCollection` etc

In all cases, returning `null` will result in the regular RO spec representation being returned.

2.1.2. Implementation

`ContentNegotiationServiceAbstract` (in `o.a.i.v.ro.rendering.service.conneg`) provides a no-op implementation of the SPI, along with supporting methods:

```
public abstract class ContentNegotiationServiceAbstract implements
ContentNegotiationService {
    ...
    protected Object objectOf(final ObjectAdapter objectAdapter) { ... }
    protected Object returnedObjectOf(ObjectAndActionInvocation
objectAndActionInvocation) { ... }

    protected Class<?> loadClass(String cls) { ... }

    protected void ensureJaxbAnnotated(Class<?> domainType) { ... }
    protected void ensureDomainObjectAssignable(
        String xRoDomainType, Class<?> domainType, Object domainObject) { ... }
}
```

As discussed in the introduction, the framework also provides a default implementation, `o.a.i.v.ro.rendering.service.conneg.ContentNegotiationServiceXRoDomainType`. This handles content negotiation for two of the possible representations, object representations and for action result representations:

- For object representations it will handle requests with HTTP `Accept` headers of the form:
  - `application/json;profile=urn:org.restfulobjects:repr-types/object;x-ro-domain-type=…`
  - `application/xml;profile=urn:org.restfulobjects:repr-types/object;x-ro-domain-type=…`
- for action result representations it will similarly handle requests with HTTP `Accept` headers of the form:
  - `application/json;profile=urn:org.restfulobjects:repr-types/action-result;x-ro-domain-type=…`
  - `application/xml;profile=urn:org.restfulobjects:repr-types/action-result;x-ro-domain-type=…`

The value of the `x-ro-domain-type` parameter corresponds to the DTO to be mapped into by the `ContentMappingService`.

If the DTO is annotated with JAXB, then also note that the runtime type must be annotated with the JAXB `javax.xml.bind.annotation.XmlRootElement` so that RestEasy is able to unambiguously serialize it.

2.1.3. Usage

You can find an example of all these services in the (non-ASF) Isis addons' todoapp. This defines a `ToDoItemDto` class that is JAXB annotated (it is in fact generated from an XSD).

The example app also includes an implementation of `ContentMappingService` that maps `todoapp.dom.module.todoitem.ToDoItem` entities to `todoapp.dto.module.todoitem.ToDoItemDto` classes.

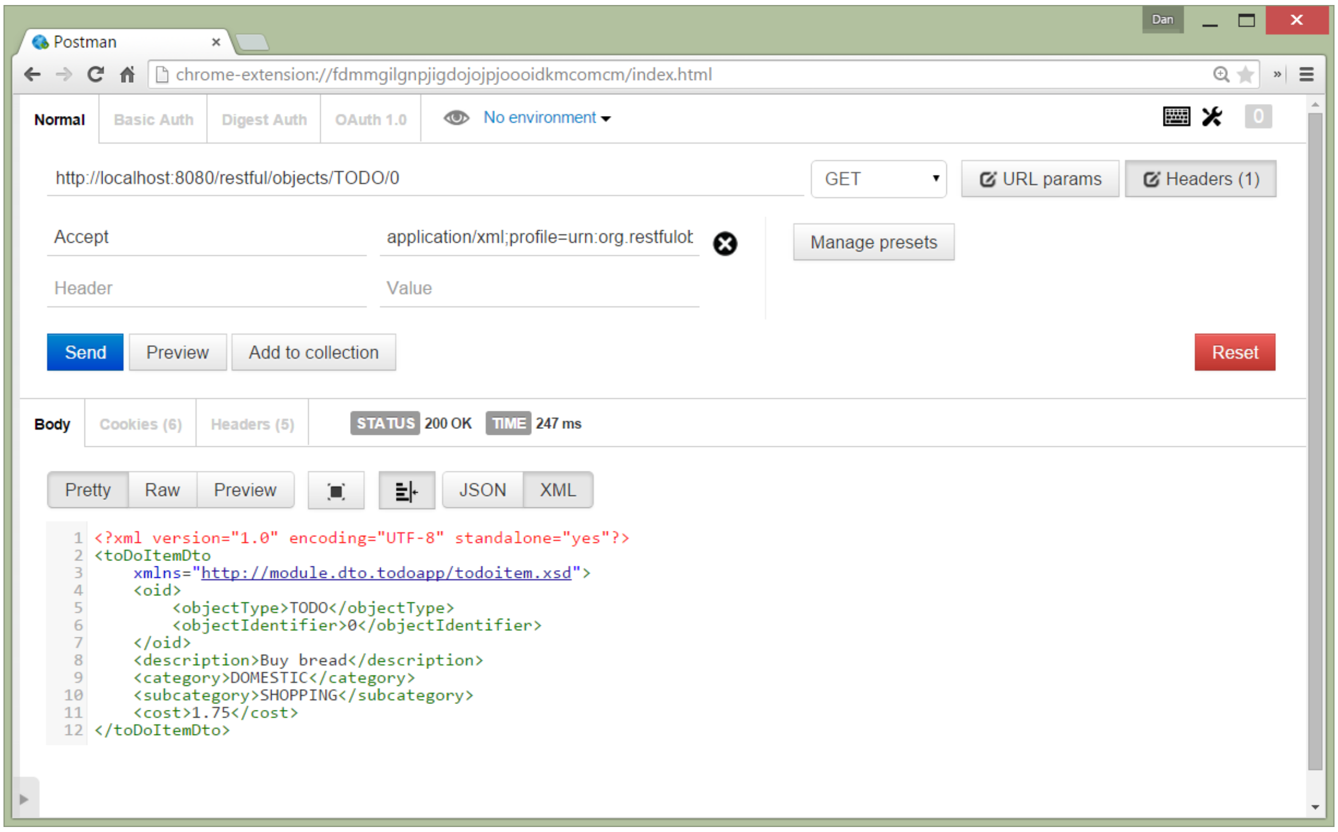
A REST client can therefore request a DTO representation of an entity by invoking

```
http://localhost:8080/restful/objects/TODO/0
```

with an `Accept` header of:

```
application/xml;profile=urn:org.restfulobjects:repr-types/object;x-ro-domain-
type=todoapp.dto.module.todoitem.ToDoItemDto
```

will result in an XML serialization of that class:



chrome-extension://fdmmgilgnpjigdojojpjoooidkmcomcm/index.html

http://localhost:8080/restful/objects/TODO/0

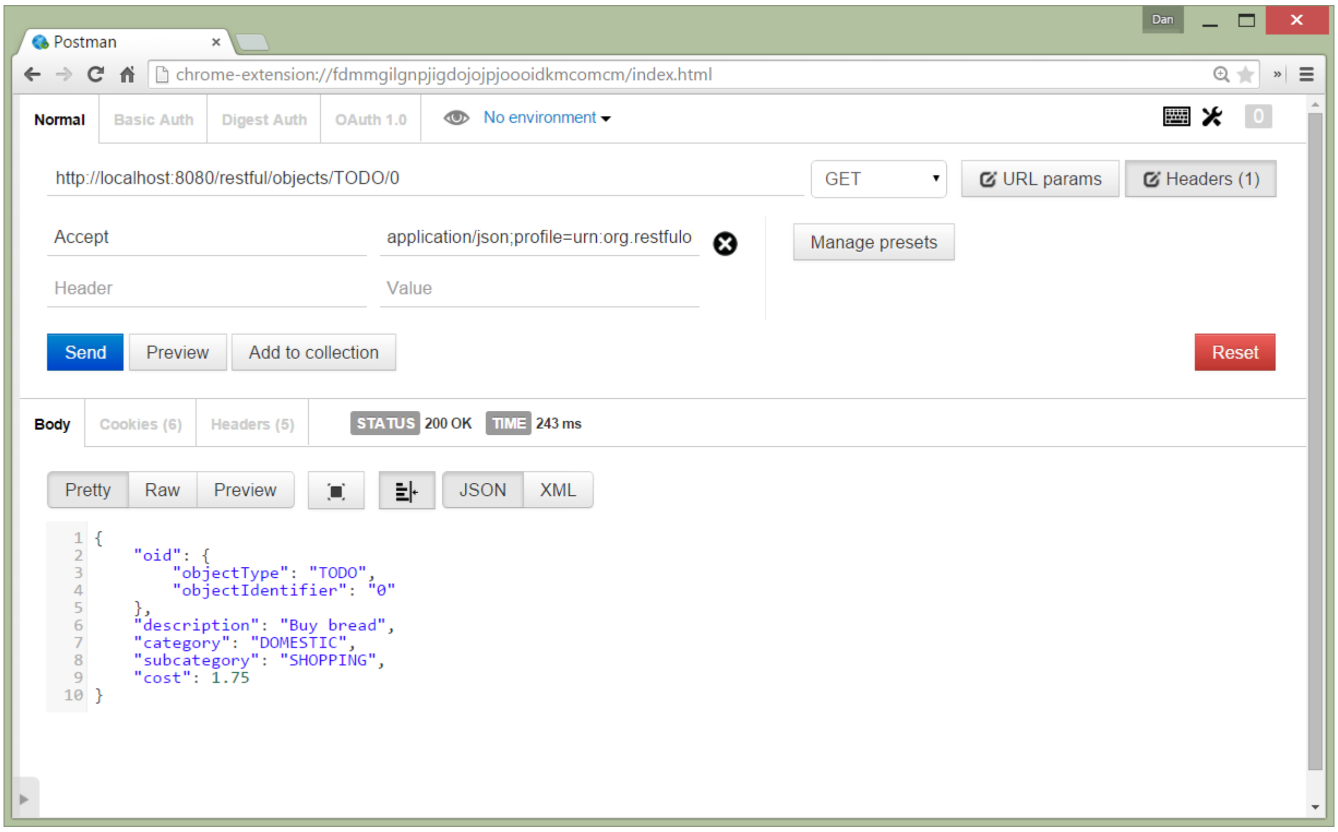
Header	Value
Accept	application/xml;profile=urn:org.restfulob

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<toDoItemDto
    xmlns="http://module.dto.todoapp/todoitem.xsd">
    <oid>
        <objectType>TODO</objectType>
        <objectIdentifier>0</objectIdentifier>
    </oid>
    <description>Buy bread</description>
    <category>DOMESTIC</category>
    <subcategory>SHOPPING</subcategory>
    <cost>1.75</cost>
</toDoItemDto>
```

while similarly hitting the same URL with an `Accept` header of:

```
application/json;profile=urn:org.restfulobjects:repr-types/object;x-ro-domain-
type=todoapp.dto.module.todoitem.ToDoItemDto
```

will result in the JSON serialization of that class:



http://localhost:8080/restful/objects/TODO/0

Accept: application/json;profile=urn:org.restfulo

```
{
    "oid": {
        "objectType": "TODO",
        "objectIdentifier": "0"
    },
    "description": "Buy bread",
    "category": "DOMESTIC",
    "subcategory": "SHOPPING",
    "cost": 1.75
}
```

2.1.4. Configuration

The default `ContentNegotiationServiceXRoDomainType` implementation provides a configuration property which controls whether a mapped domain object is pretty-printed (formatted, indented) or not:

```
isis.services.ContentNegotiationServiceXRoDomainType.prettyPrint=true
```

If the property is not set, then the default depends on the deployment type; production mode will disable pretty printing, while prototyping mode will enable it.

2.1.5. Registering the Services

Assuming that the `configuration-and-annotation` services installer is configured (implicit if using the `AppManifest` to bootstrap the app) then Apache Isis' default implementation of `ContentNegotiationService` service is automatically registered and injected (it is annotated with `@DomainService`) so no further configuration is required.

To use an alternative implementation, use `@DomainServiceLayout#menuOrder()` (as explained in the introduction to this guide).

2.1.6. Related Services

The default implementation of `ContentNegotiationService` delegates to `ContentMappingService` (if present) to convert domain entities into a stable form (eg DTO).

The `ContentNegotiationService` is itself called by the (default implementation of)

RepresentationService.

2.2. RepresentationService

The RepresentationService is the main plug-in point for the RestfulObjects viewer to generate representations.

The default implementation generates representations according to the Restful Objects spec v1.0. However, it also delegates to the ContentNegotiationService which provides a mechanism for altering representations according to the HTTP Accept header.

The principal motivation is to allow more flexible representations to be generated for REST clients that (perhaps through their use of a certain Javascript library, say) expect, or at least works best with, a certain style of representation.

In all there are three domain services that can influence the representations generated: this service, ContentNegotiationServiceand the ContentMappingService. The diagram below shows how these collaborate:

The RepresentationServiceForRestfulObjects is the default implementation of this service; likewise ContentNegotiationServiceXRoDomainType is the default implementation of the ContentNegotiationService. If you inspect the source code you'll see that the default implementation of this service's primary responsibility is to generate the default Restful Objects representations. Therefore, if you what you want to do is to generate a *different _representation then in many cases replacing either this service _or* the ContentNegotiationService will be equivalent (you'll notice that their SPIs are very similar).

2.2.1. SPI

The SPI defined by this service is:

```
public interface RepresentationService {
    @Programmatic
    Response objectRepresentation(                          ①
            Context rendererContext,
            ObjectAdapter objectAdapter);
    @Programmatic
    Response propertyDetails(                               ②
            Context rendererContext,
            ObjectAndProperty objectAndProperty,
            MemberReprMode memberReprMode);
    @Programmatic
    Response collectionDetails(                             ③
            Context rendererContext,
            ObjectAndCollection objectAndCollection,
            MemberReprMode memberReprMode);
    @Programmatic
    Response actionPrompt(                                  ④
            Context rendererContext,
            ObjectAndAction objectAndAction);
    @Programmatic
    Response actionResult(                                  ⑤
            Context rendererContext,
            ObjectAndActionInvocation objectAndActionInvocation,
            ActionResultReprRenderer.SelfLink selfLink);
    public static interface Context extends RendererContext {
        ObjectAdapterLinkTo getAdapterLinkTo();
    }
}
```

① representation of a single object, as per section 14.4 of the RO spec, v1.0

② representation of a single property of an object, as per section 16.4 of the RO spec v1.0

③ representation of a single collection of an object, as per section 17.5 of the RO spec v1.0

④ representation of a single action (prompt) of an object, as per section 18.2 of the RO spec v1.0

⑤ representation of the results of a single action invocation, as per section 19.5 of the RO spec v1.0

These methods provide:

- a `RendererContext` which provides access to request-specific context (eg HTTP headers), session-specific context (eg authentication) and global context (eg configuration settings)
- an object representing the information to be rendered

  eg `ObjectAdapter`, `ObjectAndProperty`, `ObjectAndCollection` etc
- for members, whether the representation is in read/write mode

  ie `MemberReprMode`

2.2.2. Implementation

As discussed in the introduction, the framework provides a default implementation, `o.a.i.v.ro.rendering.service.RepresentationServiceForRestfulObjects`. This delegates to `ContentNegotiationService` to generate an alternative representation; but if none is provided then it falls back on generating the representations as defined in the Restful Objects spec v1.0.

To use an alternative implementation, use `@DomainServiceLayout#menuOrder()` (as explained in the introduction to this guide).

2.2.3. Registering the Services

Assuming that the `configuration-and-annotation` services installer is configured (implicit if using the `AppManifest` to bootstrap the app) then Apache Isis' default implementation of `RepresentationService` service is automatically registered and injected (it is annotated with `@DomainService`) so no further configuration is required.

2.2.4. Related Services

The default implementation delegates to `ContentNegotiationService`, whose default implementation may delegate in turn to `ContentMappingService` (if present).

Chapter 3. Application Layer

These domain services are internal to the framework, controlling various aspects of the application layer.

The table below summarizes the application layer internal SPIs defined by Apache Isis. It also lists their corresponding implementation, either a default implementation provided by Apache Isis itself, or provided by one of the (non-ASF) Isis Addons modules.

Table 2. Internal Services

SPI	Maven Module Impl'n (g: a:)	Implementation	Notes
`o.a.i.c.m.s.command` `CommandDtoServiceInternal`	(`1.13.0-SNAPSHOT`) Creates memento of current action invocation, for use as a serializable XML reified command. The most notable usage of this is to allow the execution of the `Command` to be deferred to run in the background (via `@Action#commandExecuteIn()` or `@Property#commandExecuteIn()`).	`CommandDtoService-InternalServiceDefault` `o.a.i.c.r.s.command` `isis-core-runtime`	
`o.a.i.c.m.s.ixn` `InteractionDtoServiceInternal`	(`1.13.0-SNAPSHOT`) Creates DTO for the current execution of an action invocation or property edit, for use either as a reified command or for implementations of the `PublishingService`.	`CommandDtoService-InternalServiceDefault` `o.a.i.c.m.s.command` `isis-core-metamodel`	

Key:

- `o.a.i` is an abbreviation for `org.apache.isis`
- `o.ia.m` is an abbreviation for `org.isisaddons.module`
- `o.a.i.c.m.s` is an abbreviation for `org.apache.isis.core.metamodel.services`
- `o.a.i.c.r.s` is an abbreviation for `org.apache.isis.core.runtime.services`

3.1. `CommandDtoServiceInternal` (`1.13.0-SNAPSHOT`)

The `CommandDtoServiceInternal` (`1.13.0-SNAPSHOT`) is responsible for creating an memento of the current action invocation or property edit, to store in the `Command` object (from `CommandContext`). This memento is a JAXB DTO being an instance of the "cmd" schema, so can be reified so that its execution can be deferred until later, as a background command.

3.1.1. SPI & Implementation

The SPI of the service is:

```
public interface CommandDtoServiceInternal {
    @Deprecated
    ActionInvocationMemento asActionInvocationMemento(      ①
            Method m,
            Object domainObject, Object[] args);
    CommandDto asCommandDto(                                ②
            List<ObjectAdapter> targetAdapters,
            ObjectAction objectAction,
            ObjectAdapter[] argAdapters);
    CommandDto asCommandDto(                                ③
            final List<ObjectAdapter> targetAdapters,
            final OneToOneAssociation association,
            final ObjectAdapter valueAdapterOrNull);
    void addActionArgs(                                     ④
            final ObjectAction objectAction,
            final ActionDto actionDto,
            final ObjectAdapter[] argAdapters);
    void addPropertyValue(                                  ⑤
            final OneToOneAssociation property,
            final PropertyDto propertyDto,
            final ObjectAdapter valueAdapter);
}
```

① Note that this method (more precisely, `ActionInvocationMemento`) does *not* support mixins.

② Returns a JAXB DTO being an instance of the "cmd" schema (hence convertible to XML) that represents the *intention* to invoke an action on a target object (or possibly many targets, for bulk actions). If an action, it can also be either mixin action or a contributed action.

③ Returns a JAXB DTO that represents the intention to edit (set or clear) a property on a target (or possibly many targets, for symmetry with actions).

④ add the arguments of an action to an `ActionDto`. This is used when the command is actually executed (per `InteractionContext`) to populate the parameters of the equivalent `ActionInvocationDto`.

⑤ add the new value argument of a property to a `PropertyDto`. This is used when the command is actually executed (per `InteractionContext`) to set the the new value of the equivalent `PropertyEditDto`.

The SPI is implemented by `o.a.i.c.r.s.command.CommandDtoServiceInternalServiceDefault`.

3.1.2. Related Services

The design of this service is similar to that of `InteractionDtoServiceInternal`, used to create the `MemberExecutionDto` (from the "ixn" schema).

3.2. InteractionDtoServiceInternal (1.13.0-SNAPSHOT)

The InteractionDtoServiceInternal internal domain service (1.13.0-SNAPSHOT) is used by the framework to create and update DTOs representing member executions, ie the invocation of an action or the editing of a property. The DTO is in all cases a subclass of MemberExecutionDto, from the "ixn" schema, and subsequently accessible from the Interaction object (per the InteractionContext service).

3.2.1. SPI & Implementation

The SPI of the service is:

```
public interface InteractionDtoServiceInternal {
    ActionInvocationDto asActionInvocationDto(          ①
            ObjectAction objectAction,
            ObjectAdapter targetAdapter,
            List<ObjectAdapter> argumentAdapters);
    PropertyEditDto asPropertyEditDto(                  ②
            OneToOneAssociation property,
            ObjectAdapter targetAdapter,
            ObjectAdapter newValueAdapterIfAny);
    ActionInvocationDto updateResult(                   ③
            ActionInvocationDto actionInvocationDto,
            ObjectAction objectAction,
            Object resultPojo);
}
```

① called by the framework when invoking an action, to create a DTO capturing the details of the action invocation (target, arguments etc).

② called by the framework when editing a property, to create a DTO (for the "ixn" schema) capturing the details of the property edit (target, new value etc).

③ called by the framework to attach the result of an action invocation to the aforementioned DTO.

The service is implemented by o.a.i.core.runtime.services.ixn.InteractionDtoServiceInternalDefault.

3.2.2. Related Services

The design of this service is similar to that of CommandDtoServiceInternal, used to create the CommandDto (from the "cmd" schema).

Chapter 4. Persistence Layer internal SPI

These domain services are internal to the framework, controlling various aspects of the persistence layer.

The table below summarizes the persistence layer internal SPIs defined by Apache Isis. It also lists their corresponding implementation, either a default implementation provided by Apache Isis itself, or provided by one of the (non-ASF) Isis Addons modules.

Table 3. Internal Services

SPI	Maven Module Impl'n (g: a:)	Implementation	Notes
`o.a.i.c.r.s.auditing. AuditingServiceInternal`	(`1.13.0-SNAPSHOT`) Co-ordinates between `ChangedObjectsServiceInternal` and `AuditingService`.	concrete class.	
`o.a.i.c.r.s.changes. ChangedObjectsServiceInternal`	(`1.13.0-SNAPSHOT`) Request-scoped service holding objects enlisted into current transaction.	concrete class.	
`o.a.i.c.m.s.publishing. PublishingServiceInternal`	(`1.13.0-SNAPSHOT`) Co-ordinates between `ChangedObjectsServiceInternal` and `MetricsService` and the SPI services, `PublisherService` and (deprecated) `PublishingService`.	`PublishingService-InternalDefault` `o.a.i.c.r.publishing` `isis-core-runtime`	

Key:

- `o.a.i` is an abbreviation for `org.apache.isis`
- `o.ia.m` is an abbreviation for `org.isisaddons.module`
- `o.a.i.c.m.s` is an abbreviation for `org.apache.isis.core.metamodel.services`
- `o.a.i.c.r.s` is an abbreviation for `org.apache.isis.core.runtime.services`

4.1. `AuditingServiceInternal` (`1.13.0-SNAPSHOT`)

The (internal) `AuditingServiceInternal` domain service (`1.13.0-SNAPSHOT`) acts as an internal facade to any configured `AuditingService`. It is responsible for obtaining the details of all changes to domain objects within an interaction, and then to call the configured `AuditingService` to actually create audit entries of those changes.

The service is a no-op if there is no configured `AuditingService`.

4.1.1. SPI and Implementation

The SPI of the service is:

```
public class AuditingServiceInternal {
    public boolean canAudit();               ①
    public void audit();                     ②
}
```

① whether auditing is enabled; checks to see if any `AuditingService` has been configured.

② uses the `ChangedObjectsServiceInternal` to obtain details of the changed properties, then call the configured `AuditingService`.

The service implementation is `o.a.i.c.r.s.auditing.AuditingServiceInternal`.

4.1.2. Registering the Service

Assuming that the `configuration-and-annotation` services installer is configured (implicit if using the `AppManifest` to bootstrap the app) then Apache Isis' default implementation of `AuditingServiceInternal` class is automatically registered (it is annotated with `@DomainService`) so no further configuration is required.

4.1.3. Related Classes

The service delegates between the (internal) `ChangedObjectsServiceInternal` domain service to the configured `AuditingService`. If no such `AuditingService` is configured, this service is in effect a no-op.

The (internal) `PublishingServiceInternal` performs a similar function for the `PublisherService`, also collating details of the changed objects from `ChangedObjectsServiceInternal`.

4.2. `ChangedObjectsServiceInternal` (`1.13.0-SNAPSHOT`)

The `ChangedObjectsServiceInternal` class (`1.13.0-SNAPSHOT`) is an (internal) request-scoped domain service that is responsible for collecting the details of all changes to domain objects within an interaction. This is then used by various other (internal) domain services, notably `AuditingServiceInternal` and `PublishingServiceInternal`.

4.2.1. SPI and Implementation

The SPI of the service is:

```
@RequestScoped
public class ChangedObjectsServiceInternal {
    public void enlistCreated(final ObjectAdapter adapter);
①
    public void enlistUpdating(final ObjectAdapter adapter);
    public void enlistDeleting(final ObjectAdapter adapter);

    public boolean hasChangedAdapters();
②

    public Map<ObjectAdapter, PublishedObject.ChangeKind>
getChangeKindByEnlistedAdapter();     ③
    public int numberObjectsDirtied();
    public int numberObjectPropertiesModified();

    public Set<Map.Entry<AdapterAndProperty, PreAndPostValues>>
getChangedObjectProperties();   ④

    public void clearChangedObjectProperties();
⑤
}
```

① Enlists an object that has just been created, updated or deleted, capturing the pre-modification values of the properties.

② Used by the framework to determine whether to set the "persist hint" on the `Command` object (as per `CommandContext`).

③ Used by `PublishingServiceInternal` to obtain details of and counters of all objects changed within the transaction.

④ Used by `AuditingServiceInternal` to obtain all pairs of pre/post values of changed properties

⑤ Called by the framework to for clean up after auditing and publishing has completed.

For enlisted objects, if just created, then a dummy value `"[NEW]"` is used for the pre-modification value; if just deleted, then a dummy value `"[DELETED]"` is used for the post-modification value. The post-modification values of properties are captured when the transaction commits.

The service implementation is `o.a.i.c.r.s.changes.ChangedObjectsServiceInternal`.

4.2.2. Registering the Service

Assuming that the `configuration-and-annotation` services installer is configured (implicit if using the `AppManifest` to bootstrap the app) then Apache Isis' default implementation of `ChangedObjectsServiceInternal` class is automatically registered (it is annotated with `@DomainService`) so no further configuration is required.

4.2.3. Related Classes

Both the `AuditingServiceInternal` and `PublishingServiceInternal` (internal) domain services query this object.

4.3. PublishingServiceInternal (1.13.0-SNAPSHOT)

The (internal) PublishingServiceInternal domain service (1.13.0-SNAPSHOT) acts as an internal facade to any configured PublisherService or (deprecated in 1.13.0-SNAPSHOT) PublishingService domain services.

For published action invocations/ property edits, it provides an API for those member executions to call.

For published objects, it provides an API for the framework to call at the end of the interaction; it obtains details of the changed objects (from the ChangedObjectsServiceInternal) and filters them to just those objects that are to be published; these are then passed through to any configured PublisherService or PublishingService implementations.

4.3.1. SPI and Implementation

The SPI of the service is:

```
public class PublishingServiceInternal {
    void publishAction(
            Interaction.Execution execution,        ①
            ObjectAction objectAction,              ②
            IdentifiedHolder identifiedHolder,
            ObjectAdapter targetAdapter,
            List<ObjectAdapter> parameterAdapters,
            ObjectAdapter resultAdapter);
    void publishProperty(                           ③
            Interaction.Execution execution);
    void publishObjects();                          ④
}
```

① to publish an action invocation, as represented by the specified member Execution parameter and with the @Action#publishing() annotation attribute or equivalent, to any configured PublisherService. The Execution object will be an instance of ActionInvocation (see InteractionContext for details).

② the remaining parameters are to support the publishing of the action to any configured PublishingService services (deprecated in 1.13.0-SNAPSHOT).

③ to publish a property edit, as as represented by the specified member Execution parameter and with the @Property#publishing() annotation attribute or equivalent, to any configured PublisherService. The Execution object will be an instance of PropertyEdit (see InteractionContext for details).

④ to publish all changed objects that are to be published (with the @DomainObject#publishing() annotation attribute or equivalent).

The service implementation is o.a.i.c.m.s.publishing.PublishingServiceInternal.

4.3.2. Registering the Service

Assuming that the `configuration-and-annotation` services installer is configured (implicit if using the `AppManifest` to bootstrap the app) then Apache Isis' default implementation of `PublishingServiceInternal` class is automatically registered (it is annotated with `@DomainService`) so no further configuration is required.

4.3.3. Related Classes

The service delegates between the (internal) `ChangedObjectsServiceInternal` domain service to the configured `PublisherService` and `PublishingService`.

The (internal) `AuditingServiceInternal` performs a similar function for the `PublisherService`, also collating details of the changed objects from `ChangedObjectsServiceInternal`.