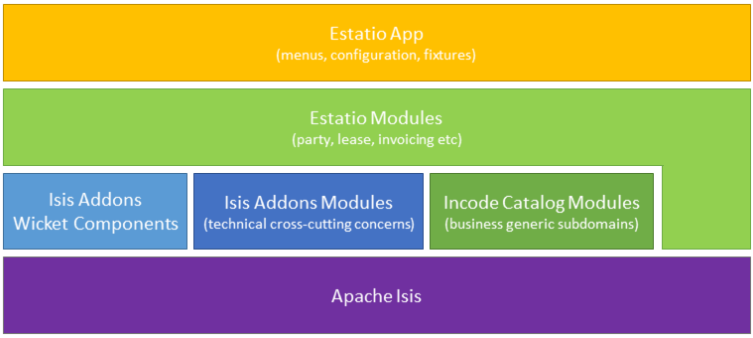
Incode Developers' Guide

This guide is intended for the developers of Estatio (and other applications).

Download PDF[1]

1. Architecture and Design

Incode applications (most notably Estatio) consists of multiple modules, deployed as a single WAR (a "modular monolith"). These modules are layered as follows:



At the bottom is Apache Isis[2] framework. The (Estatio) application and modules depend on the framework in that its domain objects are annotated with Apache Isis and DataNucleus annotations, and use domain services provided by Apache Isis (such as `RepositoryService` and `TitleService`).

On top of that are a number of modules from Isis Addons[3]. These address technical cross-cutting concerns such as security, auditing, publishing, freemarker mail-merge and so-on. The (Estatio) domain objects don't (tend to) have any dependencies on these modules (though there are one or two mixins to allow the audit trail to be viewed, for example). These modules are versioned independently (with a new version released for each major release of Apache Isis itself).

[1] https://github.com/incodehq/developers-guide/blob/master/pdf/developers-guide.pdf
[2] http://isis.apache.org
[3] http://www.isisaddons.org

At about the same level of abstraction are modules provided by the Incode Catalog[4]. These provide functionality for various generic business subdomains, such as documents, notes, aliases and classifications. This functionality is expected/hoped to be reusable across multiple different applications. Like the Isis Addons, these modules are versioned independently (with a new version released for each major release of Apache Isis itself).

Also part of the application are a number of Wicket UI components, also provided by Isis Addons[5]. Like the modules, the Estatio domain objects don't (tend to) have any dependencies on these Wicket UI components; the one exception is that objects that can be rendered on a map must implement the `Locatable` interface. Again, these components are versioned independently (with a new version released for each major release of Apache Isis itself).

Sitting on top of the generic business domain modules are the modules that contain the (Estatio) domain objects themselves, residing in Estatio (open source)[6] git repo. There is some integration between (Estatio) domain objects and these generic subdomains, for example to allow `Documents` to be attached to various domain objects (such as `Invoice` etc). To ensure modularity, these reside in separate maven modules, but are versioned together with all of them sharing a single parent `pom.xml`.

Finally there is the application itself. This defines the `AppManifest` (which specifies the modules make up the application) and various other configuration properties. It also defines the top-level menus which are presented to the end-user. These modules also versioned along with the Estatio domain objects mentioned above (they share the same parent `pom.xml`),

In the case of Estatio, the application layer is actually broken into two. In the Estatio (open source)[7] git repo there is the `estatio-app` module (defining the `EstatioAppManifest` and menu services etc), as well as the `estatio-webapp` module (which configures the various UI components and builds the WAR file). In addition, there is private (bitbucket) repo that contains a replacement webapp; this is used for the deployment of Estatio at Eurocommercial Properties (ECP). This

[4] http://catalog.incode.org
[5] http://www.isisaddons.org
[6] http://github.com/estatio/estatio
[7] http://github.com/estatio/estatio

brings in ECP-specific integration with CMS, General Ledger etc using Camel, ActiveMQ and other integration technologies.

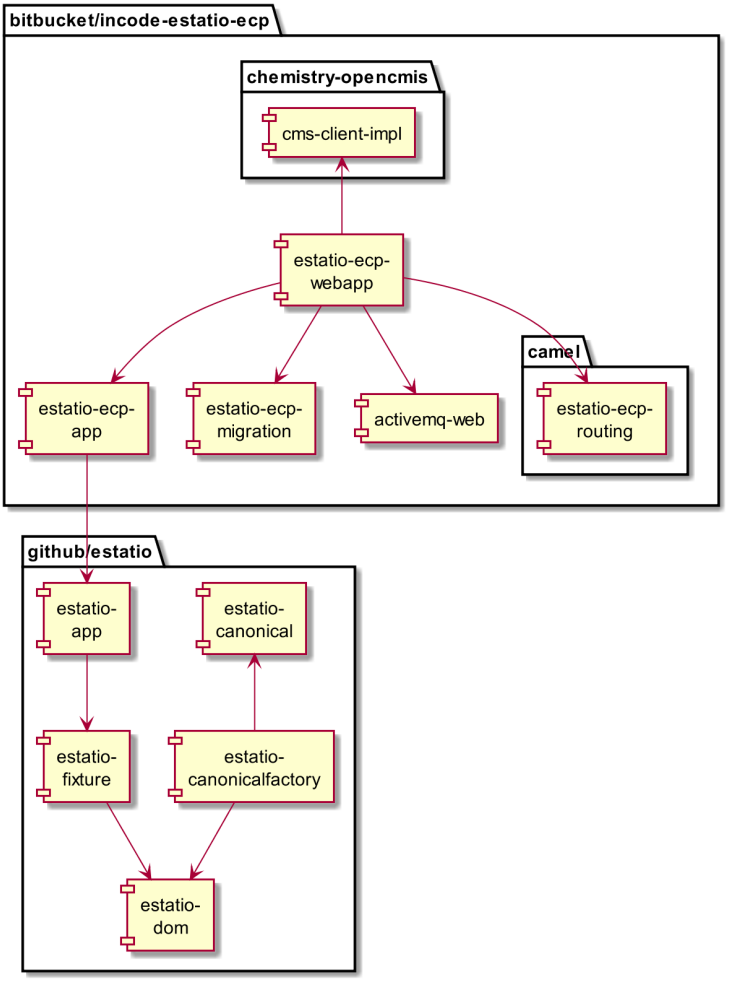
1.1. Estatio (ECP) Modules

(As noted above), Estatio (as deployed as ECP) consists of Estatio (open source)[8] along with a private repo (hosted in bitbucket) containing ECP-specific integration with CMS, General Ledger and stuff.

The modules that currently make up Estatio (ECP) are shown in this diagram[9], rendered from this PlantUML source (via asciidoctor-diagram):

[8] http://github.com/estatio/estatio

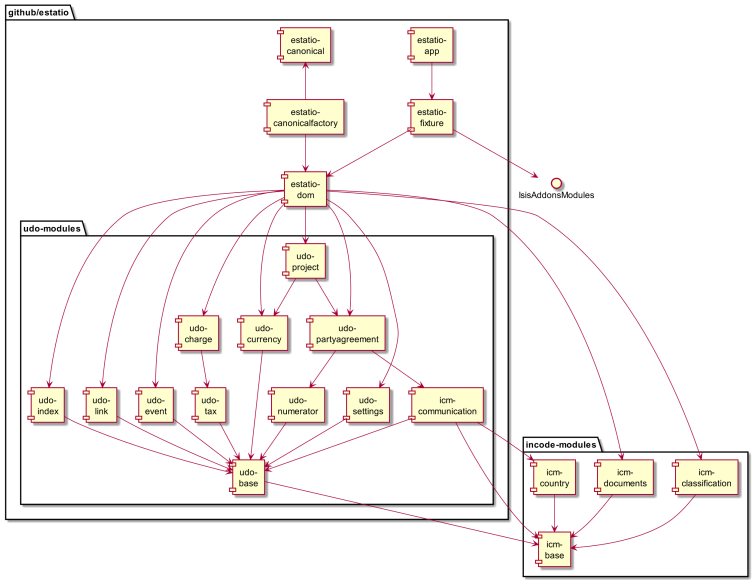
[9] https://raw.githubusercontent.com/incodehq/developers-guide/master/images/estatio-ecp.png



1.2. Estatio (Open Source) Modules

The modules that currently make up Estatio can be found in this diagram[10], rendered from the following PlantUML source (via asciidoctor-diagram):

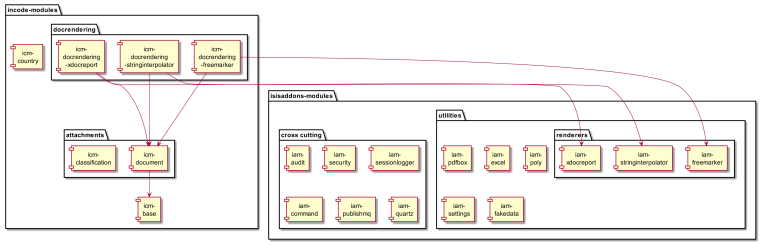
[10] https://raw.githubusercontent.com/incodehq/developers-guide/master/images/estatio-and-modules.png



Note that `icm-communication` currently resides in `github/estatio`, but the intention is to move it into `incode-modules`. The story (EST-866) is currently blocked because of refactoring that is required to generalize the concept of application tenancy paths (so that a given object might resolve to one or more application tenancies).

1.3. Incode Modules / IsisAddons

The modules that make up Incode Catalog and the Isis Addons can be found in this diagram[11], rendered from the following PlantUML source (via asciidoctor-diagram):



[11] https://raw.githubusercontent.com/incodehq/developers-guide/master/images/incode-and-isisaddons.png

Incode Modules not currently used in Estatio are:

- `icm-notes` (instead we have `udo-events`)
- `icm-alias` (still to be integrated)
- `icm-commchannel` (instead we have `icm-communications`).

Isis Addons packages not currently used in Estatio are:

- `iam-docx` (instead we use SQL Server Reporting Services)
- `iam-xdocreport` (instead we use SQL Server Reporting Services)
- `iam-publishing`(instead we use `iam-publishmq`)
- `iam-servletapi` (not required)
- `iam-tags` (instead we have the "lease.brands" package of `estatio-dom`)
- `iam-togglz`(not required)

2. Preserving the Architectural Integrity

aka "Cheese Moving" :-)

2.1. Module Dependencies (As-is vs To-be)

The UML component diagrams above represent the "as is" case, but this remains work-in-progress. Longer term "cheese moving" goals:

- move `icm-communications` formally out of github/estatio, and move to github/incodehq
    - currently blocked by application tenancy refactorings
- probably split `udo-partyagreement`, separate out `udo-party` and `udo-agreement` modules
- probably move `budgeting` package from `estatio-dom` to separate `udo-budgeting` module
- probably move `budgetassignment` package from `estatio-dom` to separate `udo-budgetassignment` module
- probably move `invoice` package from `estatio-dom` to separate `udo-invoice` module

- probably move "financial" packages from `estatio-dom` to separate `udo-xxx` module(s)

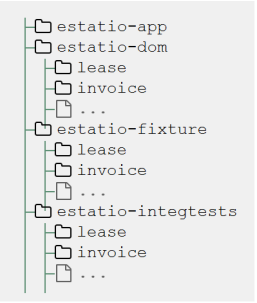
In terms of how this impacts database schemas, the approach we've gone for is:

- all Incode catalog and Isis Addons modules should be in their own schema
- all Estatio code should be in the `dbo` schema
    - this is mostly because we haven't yet found a way to make DataNucleus work with PK/FKs of entities in different schemas
    - note though that relationships between superclass/subclasses *can* be in different schemas (which is why the table-of-two-halves pattern as used by `incode-module-classification` and `incode-module-document` works ok)
- keep the code and the database DDL in sync
    - don't rely on "hacks" such as `.orm` files
    - the only exception is for modules (such as `icm-country` and `icm-communications`) that have already been refactored/moved out of estatio codebase; for these the `.orm` files should be considered a temporary measure
- use explicit (Apache Isis) `@DomainObject#objectType` and (DataNucleus) `@Discriminator` to ensure backward compatibility with persisted data

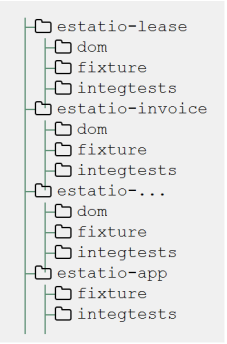
2.2. Keeping tests closer to code.

We also want to reorganize `dom` vs `fixture` vs `integtests`. Rather than have separate modules for each (resulting in all the integration tests lumped together), we instead want to group these by module so far as possible.

Thus, where today we have:



we instead want to evolve to:



where most of the integration tests reside with the module, but the `estatio-app` module contains any "left over" the fixtures and integration tests for the entire application.

2.3. Reducing Maven Boilerplate

Note also that the above refactorings could result in more boilerplate/repetition within the poms. That's because at the moment we have all the stuff relevant to integration tests in a single module, whereas having multiple integration test modules will obviously introduce more boilerplate.

There are a couple of third-party Maven plugins that aim to provide "mixins" or "tiles", opening up the idea of reusable snippets of POM files that can be stitched together:

- mixin-maven[12] plugin

[12] https://github.com/odavid/maven-plugins/tree/master/mixin-maven-plugin

- maven-tiles[13] plugin

Hopefully one of these might do the job.

3. Idioms and Patterns

Include:

- separate out menu from repository, with the menus in `estatio-app`
- use `@MemberOrder` to associate actions with properties or collections
- use `.layout.xml` for other layouts.
  - each layout should have a General/Application level/Metadata tab
- use mixins rather than contribution services
- follow conventional prefixes for action names, to automatically pick up CSS icons (hard-coded in `EstatioAppManifest`, search for "isis.reflector.facet.cssClassFa.patterns")
- don't have actions called "change" or "edit"; search instead for the deeper business rule
- use `XxxType.Meta` to gather together constants for datatypes (eg names, codes, descriptions etc).

(There are undoubtedly many more, just not yet documented…)

4. Development Environment

4.1. IDE

We use IntelliJ; see the Apache Isis docs[14].

4.2. git

Check out the Estatio Open Source[15] version and also the ECP private version (from bitbucket).

[13] https://github.com/repaint-io/maven-tiles
[14] http://isis.apache.org/guides/dg.html#_dg_ide_intellij
[15] http://github.com/estatio/estatio

4.3. repo and foreach scripts

Use this gist[16] to provide the `repo` and `foreach` bash functions, along with the _repos.txt[17] config file listing the location of all repositories.

You can then commands such as:

- `repo est`

to switch to the first directory with "est" in its filename (all other matches are also echo'd to the console), and you can run commands against all repos, eg:

- `foreach -g isis-module git fetch`
- `foreach -g isis-module git merge --ff-only`
- `foreach -g isis-module git commit -am \"EST-1234: corrects a problem in lots of places\"`

where the `-g` flag does a grep for matching repos of the argument ("isis-module" in the example above).

4.4. AsciiDoc documentation

We use Asciidoc for our documentation (such as it is). These reside in `adocs/documentation` directory, along with this README, of course. There is also some older non-Asciidoc documentation under `docs/` directory.

Recommended for editing Asciidoc[18], which is cross-platform. This provides side-by-side preview of the document (with sync'ed scrolling), and an outline view of the document.

Also, install GraphViz (to enable asciidiagram support, eg component diagrams above).

[16] https://gist.github.com/danhaywood/21b5b885433fd8bc440da3fab88c91cb
[17] https://gist.github.com/danhaywood/938f0f751f756b1cfd6a9751b8779407
[18] http://asciidocfx.com/

5. Development Practices

5.1. Kanban Boards

We maintain a (private) JIRA with two Kanban boards:

- Daily Stand-up, reviewed daily.
- Backlog, groomed weekly.

Periodically stories are moved from the "Backlog" board to the "Daily Standup" board.

5.2. Story lifecycle

(As its name suggests) the "Daily Stand-up" board is reviewed daily, and helps the team synchronize on work. For example, a story may need reviewing, in which case this can be flagged, or it may be blocked, awaiting input. The board itself defines the following columns:

- Next - longer-term stories (arrived from the Backlog)
- Current - work ready to put into play
- Blocked - awaiting input, eg from the business or external vendors
- In Progress - actively being worked on
- In Review - completed, waiting for review by some other team member
- Done - completed and merged into `master`, awaiting deployment.

We from right to left, looking to move stuff across the board.

If a new story is created in JIRA, then our workflow adds it to the "Current" column. This brings it to the attention of all for the next days stand-up (where it will either remain where it is, or perhaps be moved to the "Next" column or even the Backlog board if lower priority).

We aim to keep `master` deployable to production at all times. For all but the most trivial stories we use git branches and pull requests to allow stories to be reviewed by others in the team.

Overall then, the process is:

- identify the story, pull from "Current" column of "Daily Stand-up" board to "In progress", and assign to yourself.

> It's bad form to push work onto developers, rather developers pull work onto them.

- Identify what needs to be done.

  Generally this is an informal decision. As a quick checklist, consider:

  - which module will the change be made in
  - how will a feedback loop (typically be the business) be established
  - what unit tests are required
  - what integration tests are required
  - what DB migration scripts are required.
- Create and work in a new git branch
- If blocked on a story, and no-one is available to immediately assist, then move the story to "Blocked" and find some other work
- When done, push the branch and raise a PR. Move the story to "For Review". Ensure that any DB scripts that might need to be applied are clearly identified. Find someone to review the changes.
- With another developer, review the changes in the PR. If both happy, then merge the PR and push to `master`.

5.3. Style Guide

Commit message format

Use the format:

```
EST-xxxx: fixes the yada yada yada
```

that is, specifying the Jira issue number, and then a description of the change in the *present* tense.

For example:

- `EST-864: fixes bad reference to country-dom (was -SNAPSHOT, should be 1.13.0)`
- `EST-863 and EST-865: moves base and documentation module out to incode.`
- `EST-861: removes EstatioUserRole, with functionality moved to EstatioRole`

The idea of using present tense is that the commit history, when read back, can be read as: "this patch, if it was applied… "

Editing Asciidoc

Some guidance on writing Asciidoc (this list will likely grow in the future):

- Start each sentence in a paragraph on a new line.

  This makes it easy to spot too-long sentences, and sentences that are repetitive.
  It also makes it easy to apply pull requests to documentation.

5.4. Multi-module development

(As discussed above), Estatio consists of multiple modules:

- Isis Addons[19]' (technical) modules and wicket UI components each reside in their own git repo. These are versioned independently, generally tracking that of Apache Isis itself (eg `1.13.0`, `1.13.1`, `1.13.2` might all be releases running on top of Apache Isis `1.13.0`). They are re-released every time there is a new release of Apache Isis itself.
- Incode Catalog[20] (business) modules also each reside in their own git repo. These too are re-released every time there is a new release of Apache Isis itself.
- Domain objects specific to Estatio itself (`Party`, `Lease`, `Invoice` etc are in separate maven modules) within the Estatio (open source)[21] git repo. These

[19] http://www.isisaddons.org
[20] http://catalog.incode.org
[21] http://github.com/estatio/estatio

are split into different to eliminate cyclic dependencies between modules (to avoid the big ball of mud).

The Estatio application itself is also in maven modules within the Estatio (open source)[22] git repo.

For any given user story, we expect that changes should only need to be made to code in one module.

If we find that it isn't the case that any given user story only changes code in one module, then that is an indicator that the boundaries between the modules themselves maybe wrong, so should probably be reworked. In other words, we should ensure that the single responsibility principle[23] is followed: code that changes at the same rate should be grouped together.

In the case where a user story changes functionality that resides within only te Estatio domain modules (in this git repo), then there's not much to be said: just prototype and make the change, then productionize with unit- and integration-tests.

The slightly more complex case is a user story which changes functionality within an Isis Addons[24]' (technical) modules, or an Incode Catalog[25] (business) modules. Most of these modules have their own demo apps and integration tests, so *in theory* one could build out the new functionality just within that demo app. However, context is king, so what we recommend instead is that you do most of the work in the context of Estatio.

The steps are:

- import (into IntelliJ) the `-SNAPSHOT` version of the module to be changed
- update Estatio locally to reference that `-SNAPSHOT`.

[22] http://github.com/estatio/estatio
[23] https://en.wikipedia.org/wiki/Single_responsibility_principle
[24] http://www.isisaddons.org
[25] http://catalog.incode.org

- if necessary, do a manual reimport of all existing Estatio modules (to make sure that IntelliJ's classpaths are correctly resolving to the snapshot)
- You can then prototype and develop the changes.

When the feature is more or less there, then:

- switch back to the demo app and productionize the changes by adding in any unit- and integration tests for the functionality that has been prototyped
- push out an interim release of the module (details below)
- update Estatio to reference the interim release.
- push the changes for Estatio itself.

We insist on using interim releases because to ensure traceability.

The mechanics of creating an interim release are very simple, just call `interim-release.sh` script in the module's root directory. You will find that the `README` for each module explains how this is done; basically though it's just a matter of running a command such as:

```
sh interim-release.sh 1.13.0 origin
```

where:

- `1.13.0` is the base release (adjust as necessary)
- `origin` is the repo to push back to.

Estatio's CI server (on CloudBees) will then create a new timestamped build, eg `1.13.0.20161017-1231`; this is published to the Snapshot repository[26], eg for the Incode Catalog[27] or the Isis Addons modules[28].

5.5. Deploying to Dev or Test servers.

In the `estatio-ecp` repo the `deploy-tomcat8.sh` script can be used to deploy. See the `README` in the private ECP repository for more info.

[26] http://repository-estatio.forge.cloudbees.com/snapshot/
[27] http://repository-estatio.forge.cloudbees.com/snapshot/org/incode/module/
[28] http://repository-estatio.forge.cloudbees.com/snapshot/org/isisaddons/module/

Testing email

We use debugmail.io[29] for testing. This provides an SMTP service and a web UI that lets the emails received be inspected and optionally forwarded onto a "real" email account. See the `README` in the private ECP (bitbucket) repo for more info on how to run the application with the correct configuration properties to test this.

5.6. Releasing and Deploying to Production

Periodically the code in `master` will be deployed to production.

First it is released, then deployed.

To release the open source version, use eg:

```
repo est
sh release.sh -j EST-1234 -r 1.7.0 -s 1.8.0-SNAPSHOT
```

where:

- -j is the JIRA number
- -r is the release version
- -s is the next snapshot version

Then, for the `estatio-ecp` repo, use the same command, eg:

```
repo ecp
sh release.sh -j EST-1234 -r 1.7.0 -s 1.8.0-SNAPSHOT
```

6. Appendices

6.1. How to generate this guide as a PDF

This guide can be generated as a guide simply by loading into AsciidocFX and then saving as a PDF. Copy into the `pdf/` folder.

[29] https://debugmail.io

6.2. How to generate the AsciiDoc pages

For the AsciiDoc pages in `adocs/documentation`, just run:

```
cd adocs/documentation
mvn site
```

The `.html` will be generated in `target/site`. It should be possible to load the HTML straight from the directory. Alternatively, load from a webserver, eg:

```
python -m SimpleHTTPServer
```

and browse to http://localhost:8000.

The `mvn` script also generates docbook XML and PDF, but there are some caveats:

- the PDF currently does not include images at all.
- Using asciidoctor-fopub[30] the XML can be converted to PDF; however there are currently some issues with images being scaled correctly.

[30] https://github.com/asciidoctor/asciidoctor-fopub