INCEpTION Developer Guide

The INCEpTION Team

Version 0.5.1

Table of Contents

This document targets developers working on INCEpTION.

Setup

GIT

All sources files are stored using UNIX line endings. If you develop on Windows, you have to set the `core.autocrlf` configuration setting to `input` to avoid accidentally submitting Windows line endings to the repository. Using `input` is a good strategy in most cases, thus you should consider setting this as a global (add `--global`) or even as a system (`--system`) setting.

Configure git line ending treatment

```
C:\> git config --global core.autocrlf input
```

After changing this setting, best do a fresh clone and check-out of the project.

Setting up the for development in Eclipse

This is a guide to setting up a development environment using Eclipse on Mac OS X. The procedure should be similar for other operation systems.

First, you need to follow some steps of the user [InstallationGuide installation guide]. It is recommended to configure a MySQL-server.

We recommend you start from a **Eclipse IDE for Java Developers** package.

Use a JDK

On Linux or OS X, the following setting is not necessary. Having a full JDK installed on your system is generally sufficient. You can skip on to the next section.

On Windows, you need to edit the `eclipse.ini` file and directly before the `-vmargs` line, you have to add the following two lines. Mind to replace `C:/Program Files/Java/jdk1.8.0_144` with the actual location of the JDK on your system. Without this, Eclipse will complain that the `jdk.tools:jdk.tools` artifact would be missing.

Force Eclipse to run on a JDK

```
-vm
C:/Program Files/Java/jdk1.8.0_144/jre/bin/server/jvm.dll
```

Eclipse Plug-ins

- **Maven Integration:** m2e , already comes pre-installed with the Eclipse IDE for Java Developers. If you use another edition of Eclipse which does not have m2e pre-installed, go to **Help→Install New Software**, select "--All available sites--" and choose **Collaboration → m2e - Maven**

Integration for Eclipse

- **Apache UIMA tools:** Update site: http://www.apache.org/dist/uima/eclipse-update-site/
- **Eclipse Web Development Tooling:** go to **Help→Install New Software**, select "--All available sites--" and select the following plug-ins for installation from the section **Web, XML, Java EE and OSGi Enterprise Development**:
  - Eclipse Java Web Developer Tools
  - Eclipse Web Developer Tools
  - Eclipse XML Editors and Tools
  - JST Server Adapters
  - JST Server Adapters Extensions
  - JST Server UI
  - m2e-wtp - Maven Integration for WTP
  - WST Server Adapters

Eclipse Workspace Settings

- You should check that Text file encoding is UTF-8 in **Preferences → General → Workspace** of your Eclipse install.

Importing INCEpTION into the Workspace

Checkout out the INCEpTION git repository with your favorite git client. If you use the command-line client, use the command

```
$ git clone https://github.com/inception-project/inception.git
```

In Eclipse, go to **File → Import**, choose **Existing Maven projects**, and select the folder to which you have cloned INCEpTION. Eclipse should automatically detect all modules.

Eclipse Tomcat Integration

Download Apache Tomcat from http://tomcat.apache.org/ (we're using version 8.5). Then, you need to add the Tomcat server to your runtime configuration. Go to preferences and go to **Servers → Runtime environments**:

When prompted for an installation path, specify the folder where you extracted (or installed) Apache Tomcat v8.5 into.

Change the runtime configuration for the project. On the left side of the dialog, you should now be able to select Apache Tomcat. Change its VM arguments and include the definition `-Dinception.home="/srv/inception"` to specify the home directory for the application. Also add `-Dwicket.core.settings.general.configuration-type=development` to enable the development mode. This adds additional debugging features to the UI and disables UI caches.

Head to the servers pane. If you cannot locate it in your eclipse window, add it by going to **Window → Show View → Other...** and select **Servers**. Right click on **Tomcat v8.5 localhost** and click on **Add and remove...**:

INCEpTION should now be configured to start with Tomcat.

In the **Servers** view, double-click on the Tomcat instance you have configured. Activate the checkbox **Serve modules without publishing**. Go to the **Modules** tab, select the INCEpTION module and disable auto-reloading. After these changes, you will have to manually restart the Tomcat server in order for changes to Java class files to take effect. However, as a benefit, changes to HTML, CSS or JavaScript files take effect immediately and you just have to refresh the browser to see the changes.

Checkstyle

- Install **Checkstyle Eclipse plugin** from here: `http://eclipse-cs.sourceforge.net`
- Install the **Checkstyle configuration plugin for M2Eclipse** from here: `http://m2e-code-quality.github.com/m2e-code-quality/site/latest/`
- Select all INCEpTION projects, right click and do a **Maven → Update project**

Should the steps mentioned above not have been sufficient, close all the INCEpTION projects in Eclipse, then remove them form the workspace (not from the disk), delete any `.checkstyle` files in the INCEpTION modules, and then re-import them into Eclipse again using **Import→Existing Maven projects**. During the project import, the Checkstyle configuration plugin for M2Eclipse should properly set up the `.checkstyle` files and activate checkstyle.

CAS Doctor

The CAS Doctor is an essential development tool. When enabled, it checks the CAS for consistency when loading or saving a CAS. It can also automatically repair inconsistencies when configured to do so. This section gives an overview of the available checks and repairs.

It is safe to enable any checks. However, active checks may considerably slow down the application, in particular for large documents or for actions that work with many documents, e.g. curation or the calculation of agreement. Thus, checks should not be enabled on a production system unless the application behaves strangely and it is necessary to check the documents for consistency.

Enabling repairs should be done with great care as most repairs are performing destructive actions. Repairs should never be enabled on a production system. The repairs are executed in the order in which they are appear in the `debug.casDoctor.repairs` setting. This is important in particular when applying destructive repairs.

When documents are loaded, CAS Doctor first tries to apply any enabled repairs and afterwards applies enabled checks to ensure that the potentially repaired document is consistent.

Additionally, CAS Doctor applies enabled checks **before** saving a document. This ensures that a bug in the user interface introduces inconsistencies into the document on disk. I.e. the consistency of the persisted document is protected! Of course, it requires that relevant checks have been implemented and are actually enabled.

By default, CAS Doctor generates an exception when a check or repair fails. This ensures that inconsistencies are contained and do not propagate further. In some cases, e.g. when it is known that by its nature an inconsistency does not propagate and can be avoided by the user, it may be convenient to allow the user to continue working with the application while a repair is being developed. In such a case, CAS Doctor can be configured to be non-fatal. Mind that users can always continue to work on documents that are consistent. CAS Doctor only prevents loading inconsistent documents and saving inconsistent documents.

Configuration

Setting	Description	Default	Example
debug.casDoctor.fatal	If the extra checks trigger an exception	true	false
debug.casDoctor.checks	Extra checks to perform when a CAS is saved (also on load if any repairs are enabled)	*unset*	comma-separated list of checks
debug.casDoctor.repairs	Repairs to be performed when a CAS is loaded - order matters!	*unset*	comma-separated list of repairs

Setting	Description	Default	Example
debug.casDoctor.forceReleaseBehavior	Behave as like a release version even if it is a beta or snapshot version.	false	true

Checks

All Feature Structures Indexed

ID

`AllFeatureStructuresIndexedCheck`

Related repairs

Remove Dangling Chain Links, Remove Dangling Relations, Re-index Feature-Attached Span Annotations

This check verifies if all reachable feature structures in the CAS are also indexed. We do not currently use any un-indexed feature structures. If there are any un-indexed feature structures in the CAS, it is likely due to a bug in the application and can cause undefined behavior.

For example, older versions of INCEpTION had a bug that caused deleted spans still to be accessible through relations which had used the span as a source or target.

This check is very extensive and slow.

Feature-Attached Span Annotations Truly Attached

ID

`FeatureAttachedSpanAnnotationsTrulyAttachedCheck`

Related repairs

Re-attach Feature-Attached Span Annotations, Re-attach Feature-Attached Span Annotations And Delete Extras

Certain span layers are attached to another span layer through a feature reference from that second layer. For example, annotations in the POS layer must always be referenced from a Token annotation via the Token feature `pos`. This check ensures that annotations on layers such as the POS layer are properly referenced from the attaching layer (e.g. the Token layer).

Links Reachable Through Chains

ID

`LinksReachableThroughChainsCheck`

Related repairs

Remove Dangling Chain Links

Each chain in a chain layers consist of a **chain** and several **links**. The chain points to the first link

and each link points to the following link. If the CAS contains any links that are not reachable through a chain, then this is likely due to a bug.

No Zero-Size Tokens and Sentences

ID

`NoZeroSizeTokensAndSentencesCheck`

Related repairs

Remove Zero-Size Tokens and Sentences

Zero-sized tokens and sentences are not valid and can cause undefined behavior.

Relation Offsets Check

ID

`RelationOffsetsCheck`

Related repairs

Repair Relation Offsets

Checks that the offsets of relations match the target of the relation. This mirrors the DKPro Core convention that the offsets of a dependency relation must match the offsets of the dependent.

Repairs

Re-attach Feature-Attached Span Annotations

ID

`ReattachFeatureAttachedSpanAnnotationsRepair`

This repair action attempts to attach spans that should be attached to another span, but are not. E.g. it tries to set the `pos` feature of tokens to the POS annotation for that respective token. The action is not performed if there are multiple stacked annotations to choose from. Stacked attached annotations would be an indication of a bug because attached layers are not allowed to stack.

This is a safe repair action as it does not delete anything.

Re-attach Feature-Attached Span Annotations And Delete Extras

ID

`ReattachFeatureAttachedSpanAnnotationsAndDeleteExtrasRepair`

This is a destructive variant of Re-attach Feature-Attached Span Annotations. In addition to re-attaching unattached annotations, it also removes all extra candidates that cannot be attached. For example, if there are two unattached Lemma annotations at the position of a Token annotation, then one will be attached and the other will be deleted. Which one is attached and which one is deleted is undefined.

Re-index Feature-Attached Span Annotations

ID

`ReindexFeatureAttachedSpanAnnotationsRepair`

This repair locates annotations that are reachable via a attach feature but which are not actually indexed in the CAS. Such annotations are then added back to the CAS indexes.

This is a safe repair action as it does not delete anything.

Repair Relation Offsets

ID

`RelationOffsetsRepair`

Fixes that the offsets of relations match the target of the relation. This mirrors the DKPro Core convention that the offsets of a dependency relation must match the offsets of the dependent.

Remove Dangling Chain Links

ID

`RemoveDanglingChainLinksRepair`

This repair action removes all chain links that are not reachable through a chain.

Although this is a destructive repair action, it is likely a safe action in most cases. Users are not able see chain links that are not part of a chain in the user interface anyway.

Remove Dangling Relations

ID

`RemoveDanglingRelationsRepair`

This repair action removes all relations that point to unindexed spans.

Although this is a destructive repair action, it is likely a safe action in most cases. When deleting a span, normally any attached relations are also deleted (unless there is a bug). Dangling relations are not visible in the user interface.

Remove Zero-Size Tokens and Sentences

ID

`RemoveZeroSizeTokensAndSentencesRepair`

This is a destructive repair action and should be used with care. When tokens are removed, also any attached lemma, POS, or stem annotations are removed. However, no relations that attach to lemma, POS, or stem are removed, thus this action could theoretically leave dangling relations behind. Thus, the Remove Dangling Relations repair action should be configured **after** this repair action in the settings file.

Database Model

Projects

`project`

Documents

`source_document`

The original document uploaded by a user into a project. The document is preserved in its original format.

`annotation_document`

Annotations made by a particular user on a document. The annotation document is persisted separately from the original document. There is one annotation document per user per document. Within the tool, a CAS data structure is used to represent the annotation document.

Layers

`annotation_type`

Column	Description
id	
project	
name	UIMA type name
uiName	Layer name displayed in the UI
type	span/relation/chain
description	
builtIn	Built-in types are pre-defined via DKPro Core and cannot be deleted.
enabled	If the type can be used for annotation or not. Types cannot be deleted after creation because we need to retain the type definitions in order to load CASes which still contains the type, so this is a way to not allow editing/displaying of these types anymore.
readonly	If the annotations of this type can be created/edited.
attachType	**optional (span)**
attachFeature	**optional, forbidden if attachType is unset**
allowSTacking	Behavior
crossSentence	Behavior

Column	Description
linkedListBehavior	**chain** Behavior
lockToTokenOffset	**span** Behavior
multipleTokens	**span** Behavior

For historical reasons, the names in the database differ: `attachType` is called `annotation_type`, `attachFeature` is called `annotation_feature`.

Span layer

A span layer allows to create annotations over spans of text.

If `attachType` is set, then an annotation can only be created over the same span on which an annotation of the specified type also exists. For span layers, setting `attachFeature` is mandatory if a `attachType` is defined. The `attachFeature` indicates the feature on the annotation of the `attachType` layer which is to be set to the newly created annotation.

For example, the `Lemma` layer has the `attachType` set to `Token` and the `attachFeature` set to `lemma`. This means, that a new lemma annotation can only be created where a token already exists and that the `lemma` feature of the token will point to the newly created lemma annotation.

Deleting an annotation that has other annotations attached to it will also cause the attached annotations to be deleted.

This case is currently not implemented because it is currently not allowed to create spans that attach to other spans. The only span type for which this is relevant is the `Token` type which cannot be deleted.

Relation layer

A relation layer allows to draw arcs between span annotations. The `attachType` is mandatory for relation types and specifies which type of annotations arcs can be drawn between.

Arcs can only be drawn between annotations of the same layer. It is not possible to draw an arc between two spans of different layers.

Only a single relation layer can attach to any given span layer.

If the `annotation_feature` is set, then the arc is not drawn between annotations of the layer indicated by `annotation_type`, but between annotations of the type specified by the feature. E.g. for a dependency relation layer, `annotation_type` would be set to `Token` and `annotation_feature` to `pos`. The `Token` type has no visual representation in the UI. However, the `pos` feature points to a `POS` annotation, which is rendered and between which the dependency relation arcs are then drawn.

Deleting an annotation that is the endpoint of a relation will also delete the relation. In the case that `annotation_feature`, this is also the case if the annotation pointed to is deleted. E.g. if a POS annotation in the above example is deleted, then the attaching relation annotations are also deleted.

Chain layer

`annotation_feature`

Column	Description
id	
project	
name	UIMA feature name
uiName	Feature name displayed in the UI
description	
annotation_type	(foreign key) The type to which this feature belongs.
type	The type of feature. Must be a type from the CAS or a UIMA built-in type such as "uima.cas.String".
multi_value_mode	Used to control if a feature can have multiple values and how these are represented. "none", "array".
link_mode	If the feature is a link to another feature structure, this column indicates what kind of relation is used, e.g. "none", "simple", "withRole".
link_type_name	If a "multipleWithRole" type is used, then the an additional UIMA type must be created that bears a role feature and points to the target type.
link_type_role_feature_name	The name of the feature bearing the role.
link_type_target_feature_name	The name of the feature pointing to the target.
tag_set	**optional** The id of the tagset which is used for this layer. If this is null, the label can be freely set (text input field), otherwise only values from the tagset can be used as labels.
builtIn	Built-in features are pre-defined via DKPro Core and cannot deleted.
enabled	If the feature can be used for annotation or not. Features cannot be deleted after creation because we need to retain the type definitions in order to load CASes which still contains the type, so this is a way to not allow editing/displaying of these types anymore.
visible	Feature rendered - if set to false only shown in annotation editor

Column	Description
remember	Remember feature value - whether the annotation detail editor should carry values of this feature over when creating a new annotation of the same type. This can be useful when creating many annotations of the same type in a row.
hideUnconstraintFeature	Hides un-constraint feature - whether the feature should be showed if constraints rules are enabled and based on the evaluation of constraint rules on a feature.

Examples

Table 1. Part-of-speech tag feature in the DKPro Core POS layer

Column	Value
name	PosValue
uiName	Part of speech
description	Part-of-speech tag
annotation_type	→ de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos.POS (span)
type	uima.cas.String
link_mode	null
link_type_name	null
link_type_role_feature_name	null
link_type_target_feature_name	null
tag_set	→ STTS
builtIn	true

Table 2. Arguments feature in a custom semantic predicate-argument structure

Column	Value
name	args
uiName	Arguments
description	Semantic arguments
annotation_type	→ webanno.custom.SemanticPredicate (span)
type	webanno.custom.SemanticArgument (span)
link_mode	multipleWithRole
link_type_name	webanno.custom.SemanticArgumentLink
link_type_role_feature_name	role
link_type_target_feature_name	target

Column	Value
tag_set	null
builtIn	false

Tagsets

`tag_set` `tag`

Constraints

`constraints`

Column	Description
id	
project	
name	
description	
rules	

Permissions

`project_permissions` `authorities` `users`

System Properties

Setting	Description	Default	Example
wicket.configuration	Enable Wicket debug mode	deployment	development

Search core module

The search core module contains the basic methods that implement the search service and search functionalities of INCEpTION.

The SearchService and SearchServiceImpl classes define and implement the search service as a Spring component, allowing other modules of INCEpTION to create an index for a given project, and to perform queries over that index.

The indexes have two different aspects: the conceptual index, represented by the Index class, and the physical index, represented by a particular physical implementation of an index. This allows different search providers to be used by INCEpTION. Currently, the default search implementation uses Mtas (https://github.com/meertensinstituut/mtas), a Lucene / Solr based index engine that allows to annotate not only raw texts but also different linguistic annotations.

Every search provider is defined by its own index factory, with a general index registry to hold all the available search providers.

Mtas Index

The Mtas index is implemented in the **MtasDocumentIndex** and **MtasDocumentIndexFactory** classes. Furthermore, the **MtasUimaParser** class provides a parser to be used by Lucene when adding a new document to the index.

- MtasDocumentIndexFactory

The factory allows to build a new MtasDocumentIndex through the getNewIndex method, which is called by the search service.

- MtasDocumentIndex

This class holds the main functionalities of a Mtas index. Its methods are called by the search service and allow to create, open close and drop a Mtas index. It allows to add or delete a document from an index, as well as to perform queries on the index.

Each index is related to only one project, and every project can have only one index from a given search provider.

When adding a document to a Mtas index, the Lucene engine will use the class MtasUimaParser in order to find out which are the tokens and annotations to be indexed.

- MtasUimaParser

The parser is responsible for creating a new TokenCollection to be used by Lucene, whenever a new document is being indexed. The token collection consists of all the tokens and annotations found in the document, which are transformed into Mtas tokens in order to be added to the Lucene index. The parser scans the document CAS and goes through all its annotations, finding out which ones are related to the annotation layers in the document's project - those are the annotations to be indexed. Currently, the parser only indexes span type annotations.