WebAnno Developer Guide

The WebAnno Team

Version 3.0.1

Table of Contents

This document targets developers working on WebAnno.

Setup

Installation guide to develop WebAnno in Eclipse

This is a guide to setting up a developer environment in Eclipse for WebAnno using Max OS X. The procedure should be similar for other operation systems.

First, you need to follow some steps of the user [InstallationGuide installation guide]. You need to configure your MySQL-server for WebAnno. After that, jump right to the chapter WebAnno and follow all steps besides the first one until the end of the document.

We recommend you start from a Eclipse Classic distribution.

Eclipse plugins

- **Version Control:** Use Subclipse 1.8.x, see: `http://subclipse.tigris.org/update_1.8.x` *Please do not use Subversive. If you did not start with an Eclipse Classic, you may end up with Subversive and Subclipse installed, which can easily confuse you as well as Eclipse.*
- **Maven Integration:** m2e , is included in Eclipse Classic. Use **Help→Install New Software, select "--All available sites--" and choose Collaboration → m2e - Maven Integration for Eclipse**
- **Subclipse/Maven Integration:** Update site: `http://subclipse.tigris.org/m2eclipse/1.0/`
- **Apache UIMA tools:** Update site: `http://www.apache.org/dist/uima/eclipse-update-site/`
- You should check that Text file encoding is UTF-8 in "Preferences → General → Workspace" of your eclipse install.

WebAnno and tomcat installation=

Checkout out the svn repository `https://webanno.googlecode.com/svn/trunk` and **checkout the project as Maven project**: http://webanno.googlecode.com/svn/wiki/images/checkout_as_maven_project.png

Download Apache Tomcat from `http://tomcat.apache.org/` (we're using version 7). Then, you need to add the Tomcat server to your runtime configuration. Go to preferences and go to **Servers → Runtime environments**:

http://webanno.googlecode.com/svn/wiki/images/AddApacheTomcat.png

When prompted for an installation path, specify the folder where you extracted (or installed) Apache Tomcat v7 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

`-Dwebanno.home="/srv/webanno"` to specify the home directory for WebAnno:

http://webanno.googlecode.com/svn/wiki/images/ChangeRunConfiguration.png

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 v7 localhost** and click on **Add and remove...**:

http://webanno.googlecode.com/svn/wiki/images/AddAndRemoveServer.png

You should end up with:

http://webanno.googlecode.com/svn/wiki/images/AddAndRemoveServerFinal.png

WebAnno should now be configured to start with tomcat.

Troubleshooting

If you run into problems with the last step (Add and remove...) and get the error *There are no resources that can be added or removed from the server*, checkout if you have installed *m2eclipse-wtp*:

http://webanno.googlecode.com/svn/wiki/images/Problems%20with%20no%20resource%20available.png

and go to the project settings and check if these project facets are activated for the project. If you have the _ m2eclipse-wtp_ installed, it should be sufficient to right-click on the project and do a **Maven → Update project** to reconfigure the project and have m2e update these settings:

http://webanno.googlecode.com/svn/wiki/images/ProjectsFacets.png

CAS Doctor

The CAS Doctor is an essential tool while developing WebAnno. 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 WebAnno, 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 WebAnno 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.

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 WebAnno 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	*unset*	comma-separated list of repairs

Checks

All Feature Structures Indexed

ID

`AllFeatureStructuresIndexedCheck`

Related repairs

Remove Dangling Chain Links, Remove Dangling Relations

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

For example, older versions of WebAnno 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

Reattach Feature-Attached Span Annotations

Certain span layers in WebAnno 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 of WebAnno 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 in WebAnno and can cause undefined behavior.

Repairs

Reattach 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.

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, WebAnno normally also deletes the attached relations (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**

Column	Description
allowSTacking	Behavior
crossSentence	Behavior
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 WebAnno currently does not allow 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.

Column	Description
visible	Feature rendered - if set to false only shown in annotation editor
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)

Column	Value
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
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