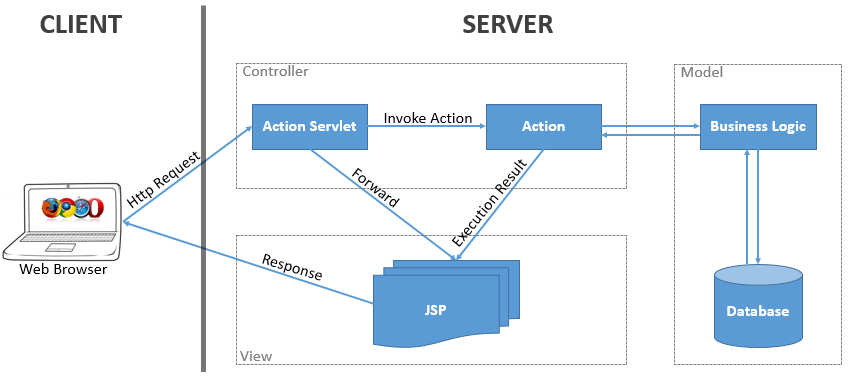
**OTO Documentation**

By Fengqiong Huang

11/07/2013

# Framework Description

OTO uses struts framework. The picture below show the architecture of struts framework.



More specifically, when a browser send a http request, it always go to action servlet configuration first. Then the configuration file will decide what kind of response the server will produce.

An http request is either a simple url (<http://biosemantics.arizona.edu/OTO/>) or url with an action name (e.g. <http://biosemantics.arizona.edu/OTO/validateLogin.do>). Notice, always attach “.do” with the action name in the request url.

Two configuration files are required to set up the request-and-response system. The first is web.xml in folder WEB-INF. In web.xml, it defines the struts configuration file name (struts-config.xml in OTO) and the default page of OTO, and other related configurations.

The second is struts-config.xml file, which is also in folder WEB-INF. This file defines how to map a certain action name (e.g. gotoLogin.do) to an action class.

Below is a simple url forward example. It means when the url is like */gotoDownload.do*, OTO will simply give back page */jsp/download.jsp*.

<action path=*"/gotoDownload"*

type=*"org.apache.struts.actions.ForwardAction"*

parameter=*"/jsp/download.jsp"* >

</action>

This simple url forward is similar to http get.

Another kind of action servlet configuration is like http post. For example the login action, the browser client needs to post the user’s email and password to the server, the server will validate the input and give proper response. The configuration is below:

<action path=*"/validateLogin"*

type=*"edu.arizona.sirls.action.UserLoginAction"*

name=*"loginForm"*

scope=*"request"*

input=*"/login.jsp"*

validate=*"true"*>

<forward name=*"reload"* path=*"/jsp/login.jsp"*/>

<forward name=*"home"* path=*"/jsp/index.jsp"*/>

</action>

It means when the post url is like “/validateLogin.do”, it will go into class “*edu.arizona.sirls.action.UserLoginAction*” and execute code there. In the UserLoginAction class, our server will query the database and validate user’s input. The server will then decide which page to give back to the client. Here the two “forward” are defined so that the server only need to decide forward name (*reload* or *home*) after validation.

All our action classes are in package *edu.arizona.sirls.action*.

The action classes may either return a .jsp page (e.g. **return** mapping.findForward(Forwardable.*RELOAD*); ) or return null with response message written in HttpServletResponse. Current usage in OTO is that when using Ajax to communicate with OTO server, we ususally use HttpServletResponse with reponse message in xml format. There will be examples in “Ajax in Javascript” section.

# Database Description

## Tables Description

On OTO server, we use MySQL to store all our data. Database name, user name and password is configured in *config.properties* file.

OTO holds multiple independent datasets. In database, we have global tables that hold general data to support the entire functionality of OTO, as well as a set of tables for each dataset. Table 1 shows all the global tables in OTO. Table 2 shows the table set for each dataset. Here we assume the dataset’s name is “datasetName”.

### Table 1: Global tables in OTO

|  |  |
| --- | --- |
| **Table Name** | **Description** |
| categories | Each dataset has its own categories table. But this table holds the default categories when creating a new dataset. When create a new dataset (e.g. upload data from CharaParser), the data in this table will be copied into the dataset’s categories table. |
| dataset\_owner | The owner of each dataset: who created this dataset. When upload data from CharaParser, who uploaded. |
| datasetprefix | Dataset list. |
| glossarytypes | Glossary type list. Do not change the records in this table since the id to name mapping is also staticly defined in the code. This id to name mapping is also used in OTOLite. |
| glossary\_dictionary | Globaly ID of triple <term, cateogry, glossary type> |
| users | Users of OTO. Three roles in OTO: U means normal user, S means super user, who have most previleges. A means administrators. Currently only Dr. Hong Cui is super user, who can manage system reserved datasets. |
| users\_log | Users operations on dataset, e.g. merge, delete, create |
| glossary\_versions |  |
| **Ontolgoy tables** |  |
|  |  |
|  |  |
|  |  |
|  |  |

### Table 2: Tables set for each dataset.

|  |  |
| --- | --- |
| **Table Name** | **Description** |
| datasetName\_categories | [table for group terms page] Categories in this dataset. |
| datasetName\_comments | [table for all pages] Comments by users |
| datasetName\_confirmed\_category | [table for group terms page] Reviewed and confirmed categorization by administrators or dataset owners. The records of this table must be synchronized with table datasetName\_user\_terms\_decisions. |
| datasetName\_confirmed\_orders | [table for orders page] Reviewed and confirmed ordering decisions by administrators or dataset owners |
| datasetName\_confirmed\_paths | [table for hierarchy page] Reviewed and confirmed hierarchical decisions by administrators or dataset owners |
| datasetName\_review\_history | [table for group terms page] Reviewed history in Group Terms page. When user review the categorization decisions, they may not change the existing decision, but we need to let the user know that he has looked at this term and agreed with the existing decision. So we consider terms they clicked, opened or dragged as reviewed and save their review history in this table. |
| datasetName\_sentence | [table for all pages] Context of terms, including all sentences in this dataset. |
| datasetName\_syns | [table for group terms page] Synonyms for finalized datasets. This table will be regenerated each time when a user finalized the dataset. |
| datasetName\_term\_category | [table for group terms page] Terms’ categories for finalized datasets. This table will be regenerated each time when a user finalize the dataset. |
| datasetName\_user\_orders\_decisions | [table for orders page] Users’ decisions on Order page |
| datasetName\_user\_tags\_decisions | [table for hierarchy page] Users’ decisions on Hierarchy page |
| datasetName\_user\_terms\_decisions | [table for group terms page] Users’ decisions on Group Terms page |
| datasetName\_web\_grouped\_terms | [table for group terms page] list of terms in Group Terms page |
| datasetName\_web\_orders | [table for orders page] List of orders in Orders page |
| datasetName\_web\_orders\_terms | [table for orders page] Terms in orders in Orders page |
| datasetName\_web\_tags | [table for hierarchy page] List of structures in Hierarchy page |

## Tables Schema Details

### Table **categories**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| category | varchar(50) | Category name shoud only be lower case and space is always replaced by underscore. Force these rules when user create their own category. |
| definition | varchar(1000) | The definition of this category. Force user to give definition when they create their own category. |

### Table **dataset\_owner**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| dataset | varchar(100) | Refer to **prefix** in **datasetprefix** table |
| ownerID | int(11) | Refer to **userid** in **users** table. System reserved datasets has ownerID = 1, which is OTO System. |

### Table **datasetprefix**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| prefix | varchar(40) | Dataset name. When the dataset is created through CharaParser upload, the prefix should contain a timestampt too, e.g. ant\_gloss\_20130517080844. The newly created datasets name should always be lower case and with no space in the name since the name will be used in naming the tables of this dataset. |
| time\_last\_accessed | timestamp | As the name. |
| tab\_general | varchar(1) | Inherited from previous developer. Not used anymore. |
| tab\_segm | varchar(1) | Inherited from previous developer. Not used anymore. |
| tab\_verf | varchar(1) | Inherited from previous developer. Not used anymore. |
| tab\_trans | varchar(1) | Inherited from previous developer. Not used anymore. |
| tab\_struct | varchar(1) | Inherited from previous developer. Not used anymore. |
| tab\_unknown | varchar(1) | Inherited from previous developer. Not used anymore. |
| tab\_finalm | varchar(1) | Inherited from previous developer. Not used anymore. |
| tab\_gloss | varchar(1) | Inherited from previous developer. Not used anymore. |
| grouptermsdownloadable | tinyint(1) | Whether the **group terms** page is finalized. Set to be true when a user finalizes **group terms** page of this dataset. Set to be false when a user reopen the **group terms** page of this dataset. |
| structurehierarchydownloadable | tinyint(1) | Whether the **hierarchy** page is finalized. Set to be true when a user finalizes **hierarchy** page of this dataset. Set to be false when a user reopen the **hierarchy** page of this dataset. |
| termorderdownloadable | tinyint(1) | Whether the **orders** page is finalized. Set to be true when a user finalizes **hierarchy** page of this dataset. Set to be false when a user reopen the **orders** page of this dataset. |
| note | text | Note of this dataset. Currently, this field stores the source of this dataset if this dataset is merged from other datasets. If this field is empty, it means this dataset is a single fresh one, not created by merging. Because we don’t have limitation on how many datasets can be merged together, this field must be **text** type. |
| glossaryType | int(11) | Refer to **glossTypeID** field in table **glossarytypes**. Specified when the dataset is created. |
| mergedInto | varchar(200) | This field stores |

### Table **glossarytypes**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| glossTypeID | int(11) | Currently we have 5 glossary types. As described before, the id and name mapping should not be changed once the record is inserted because the mapping is also been staticly defined in code. |
| glossaryName | varchar(100) | Glossary name. Should not be changed. |

### Table **glossary\_dictionary**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| termID | varchar(100) | UUID generated when inserting a record. A trigger called *glossary\_dict\_before\_insert\_uuid* was created to automatically generate UUID when inserting a reocrd in this table. The sql used to create the trigger is below:  **CREATE** TRIGGER glossary\_dict\_before\_insert\_uuid  BEFORE **INSERT** **ON** glossary\_dictionary  FOR EACH ROW  **SET** new.termID = uuid(); |
| term | varchar(100) | Term name |
| category | varchar(100) | Category name |
| glossaryType | int(11) | Glossary type |
| definition | text | Definition of the term in this category in this glossary type. |

### Table **users**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| userid | int(5) | User id |
| email | varchar(50) | User’s email |
| password | varchar(50) | User’s password. MD5 encrypted |
| firstname | varchar(50) | User’s first name |
| lastname | varchar(50) | User’s last name |
| affiliation | varchar(100) | User’s affiliation |
| status | varchar(1) | “Y” means activated. “N” means diactivated. This field will be set when an administrator of OTO approve or revoke a user in admin pages. |
| role | varchar(1) | Values can be “U” – normal users, “A” – administrators of OTO who have access to manage users and regular datasets, “S” – super user of OTO whoc have all administrator’s previleges plus access to manage system reserved systems. |
| bioportalUserId | varchar(20) | Required if the user wants to use to\_ontology page. Can be set in user’s account setting page. |
| bioportalApiKey | varchar(80) | Required if the user wants to use to\_ontology page. Can be set in user’s account setting page. |

### Table **users\_log**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| userid | int(11) | Refer to field **userid** in table **users**. |
| operation | varchar(100) | What did the user to the dataset |
| dataset | varchar(100) | Refer to field **prefix** in table **datasetprefix**. |
| operateTime | datetime | When did the action happen. |

### Table **glossary\_versions**

This table is for the old download. It was designed to do version control and management. Not used anymore since now our glossary files are uploaded to github and github will take care of the version control.

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| dataset | varchar(100) | Refer to prefix field in datasetprefix table. |
| glossaryType | int(11) | Refer to glossTypeID field in glossrytypes table. |
| filename | varchar(300) | Glossary file’s name |
| primaryVersion | int(11) | First version number. E.g. version 1.0, the value of this field is 1. |
| secondaryVersion | int(11) | Second version number. E.g. version 1.0, the value of this field is 0. |
| svnLink | varchar(300) | Reserved field. Was designed to store the download link of glossry file when uploaded to svn. |
| isLatest | tinyint(1) | Whether this glossary version is the latest version. |
| isForGlossaryDownload | tinyint(1) | Whether this glossary file is from system reserved dataset. System reserved dataset are the default glossary files for users to download with a given glossary type. |
| dateCreated | datetime | When is this record created. |
| mergedInto | varchar(100) | Track if the dataset of this glossary file has been merged into another dataset. Is null if null or empty. |

### Table datasetName**\_categories**

Categories in this dataset.

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| category | varchar(50) | Category name. Should be only lower case letters with underscore. |
| definition | varchar(1000) | Definition of this category. Categories in this table will all be listed in the group terms page. |

### Table **datasetName\_web\_grouped\_terms**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| groupId | int(11) unsigned | On group terms page, in the term list on the left side, terms are pre-grouped by groupId. If groupId = 0, do not group. List the terms one in each group. |
| term | varchar(50) | Term name. |
| cooccurTerm | varchar(50) | Inherited from previous developer. No longer used. |
| sourceDataset | varchar(100) | The sourceDataset of this term. Will be used when merging datasets and finalize dataset.  When merging dataset, each term’s sourceDataset will be set. If the same exists in more than one sourceDatasets, there will be mulitple records of that term in this table.  When finalize the dataset, sourceDataset will be filled to be the dataset name if this field is empty and the value will be included in the glossary file. |

### Table datasetName**\_user\_terms\_decisions**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| term | varchar(50) | Term name. |
| userid | int(10) | Who made this decision. Refer to userid field in users table. |
| decision | varchar(50) | Category name. Refer to category field in datasetName\_categories table. |
| decisiondate | date | When was this decision made. |
| isAdditional | tinyint(1) | If the term is decided to be a synonym, isAdditional = true. If the term is a main term, isAdditional = false. |
| relatedTerms | varchar(500) | If the term is a synonym, relatedTerms = synonym of ‘main term’. If the term is a main term, relatedTerms = a list of all its synonyms in this category. |
| isActive | tinyint(1) | A user can make a decision and then change his decision. isActive is designed to save the latest decision of this user. The latest decision represent the final decision of this term from this user and it will be used to decide if there is conflicting decisions regarding this term. |
| isLatest | tinyint(1) | All users can make decision on the term, this field is to store which decision is the latest. The idea behind is that because the latest decision maker can see all the previous decisions, this decision must be more trustworthy since he considered all the existing decisions. When a new user comes into this dataset, we are going to display (pre-populate) the latest decision to him and he can modify the decision to his satisfaction. |
| hasConflict | tinyint(1) | Whethe there is conflict regarding this term. Conflict here means different users have different active decisions on this term. This field must the same for all the records if the term is the same in this table. |
| hasSyn | tinyint(1) | To mark if the term has synonym with it. |
| decisionid | bigint(20) | Incrememtal table id. |
| groupid | int(11) | Refer to groupid field from datasetName\_web\_grouped\_terms table. Not useful anymore. |

### Table datasetName**\_review\_history**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| id | bigint(20) | Incrememtal table id. |
| userid | int(11) | Who reviewed the term. Refer to userid field in users table. |
| term | varchar(100) | Term name. |
| reviewTime | datetime | Review date. |

### Table datasetName**\_confirmed\_category**

This table mainteins synchronized record with table datasetName\_user\_terms\_decisions, which stores user’s categorization decision. When inserting a record in \_user\_terms\_decisions table, one (for simple categorization decision) or more than one (categorization decision with synonyms) records will be inserted to \_confirmed\_category table.

In \_user\_terms\_decisions table, all terms are stored as they displayed in the website (e.g. a copy red\_1 of term red is stored as red\_1), but in \_confirmed\_category table, we store the original term with its index parsed out and saved in termIndex field. Similarly for synonyms.

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| term | varchar(100) | Term name. If the term is a copied term (naming format: term name + unserscore + copy index), this field stores the original term. E.g. if the record is for term ‘red\_1’, the value in this field is ‘red’, not ‘red\_1’ |
| category | varchar(200) | category name. Refer to **category** field in **datasetName\_categories** table. |
| userid | int(11) | Who made the decision. Refer to userid field in users table. |
| confirmDate | datetime | When did the administrator approved this decision. This field will be set when an administrator accept the decision in admin page/ group terms/categories tab. |
| categoryApproved | tinyint(1) | Stores if the categorization decision is approved or not. This field will be set to be true (false) when an administrator accept (revoke) the decision in admin page/ group terms/categories tab. |
| synonymApproved | tinyint(1) | Stores if the synonym relation (the two terms in field term and field synonym are synonyms in this category) is approved or not. This field will be set to be true (false) when an administrator accept (revoke) the decision in admin page/ group terms/ synonyms tab. |
| synonym | varchar(100) | Synonym of the term. If the term has multiple synonyms, there will be multiple records of the term with one synonym in each record.  Similar to field ‘term’, this field also stores the original term, not the copied term with underscore and copy index attached to the term. |
| termIndex | int(11) | The copy index of the term. E.g. if term is ‘red\_1’, term field stores ‘red’ and this field stores 1. If the term is the original copy, the value of this fiedl is 0.  This field will be used in dataset merging function to generate term copies with the next copy index. |
| termWithIndex | varchar(100) | Combine term with termIndex. If the term of this decision is ‘red\_1’, this field stores ‘red\_1’. The value of this field is the same with the matching record in \_user\_terms\_decisions table. |
| synonymWithIndex | varchar(100) | Synonym with its copy index attached. The value of this field is the same with the matching record \_user\_terms\_decisions table. |
| isApprovedSynonym | tinyint(1) | If ‘term’ in this record has been approved to be a synonym. If the term has been approved to be a synonym, it should appear in the glossary download file.  This field will be set when an administrator accept (revoke) a synonym decision in admin page/ group terms/synonyms tab.  And this field will be used when generate datasetName\_term\_category table to filter out the terms that has been approved to be a synonym to another term. |

### Table datasetName**\_term\_category**

This table is used to generate glossary files. Each time when an administrator finalize the dataset, the table will be deleted and re-created and all content are re-generated. The glossary file is simply a .sql dump or .csv dump of this table.

The content of this table is generated from datasetName\_confirmed\_category table where categoryApproved = true and isApprovedSynonym = false.

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| term | varchar(100) | Term name. |
| category | varchar(100) | Category |
| hasSyn | tinyint(1) | If the term has synonym. If hasSyn = true, there must be at least one record in datasetName\_syns table. |
| sourceDataset | text | The source dataset of this term. For merged datasets, e.g. system reserved datasets, source datasets of a term could be more than one. That is why this field has type text.  This field is generated from sourceDataset field in table datasetName\_web\_grouped\_terms. |
| termID | varchar(100) | The UUID of this term in this category in this glossary type (glossary type of this dataset is stored in datasetprefix table.). Refer to termID field in glossary\_dictionary table. |

### Table datasetName**\_syns**

Similar to table datasetName\_term\_category, this table is also recreated and regenerated when an administrator finalize this dataset. The conent are generated from table datasetName\_confirmed\_category where isSynonymApproved = true. This table will be used to create the synonym part of the glossary download files.

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| term | varchar(100) | Main term’s name |
| synonym | varchar(100) | Synonym’s name |
| termID | varchar(100) | The termID of main term, not synonym. |

### Table **datasetName\_web\_tags**

This table is for hierarchy page. It stores the structures list displayed on the left column of hierarchy page. In OTO, the term on hierarchy page are called ‘tag’. The content of this table are generated from distinct tags (field tag) from datasetName\_sentence table.

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| tagID | bigint(20) | Incremental table id. |
| tagName | varchar(100) | Tag name. Refer to tag field in datasetName\_sentence table. |

### Table datasetName**\_user\_tags\_decisions**

This table stores the users’ decisions on hierarchy page.

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| ID | bigint(20) | Incrememtal table id. |
| tagID | bigint(20) | Refer to tagID field in datasetName\_web\_tags table. |
| name | varchar(100) | Term name. Refer to tagName field in datasetName\_web\_tags table. |
| tagPID | bigint(20) | Tag parent ID. Refer to ID field in datasetName\_user\_tags\_decisions table. |
| path | varchar(200) | Path of this tag with its parent Ids. |
| pathWithName | varchar(500) | Path of this tag with its parent names. |
| removed | tinyint(1) | Whether this tag should be removed from the lest structures list. When a user drag a structure onto the tree, he can keep the structure in the list by pressing ctrl button when dragging. If so, the term will display in gray color in the structures list to show that it has been dragged and saved as a node already. |
| isLeaf | tinyint(1) | If this node is a leaf node. |
| userid | bigint(20) | Who made this decision. Refer to userid field in users table. |
| decisionDate | datetime | When did the user make the decision. |
| hasConflict | tinyint(1) | Does the decision has conflict decisions by comparing all the paths of the same tag. |

### Table datasetName**\_confirmed\_paths**

This table is to store administrators’ acceptance of all the decisions in herarchy page.

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| term | varchar(100) | Term name. Refer to name field in datasetName\_user\_tags\_decision table. |
| path | varchar(200) | Refer to path field in datasetName\_user\_tags\_decision table. |
| pathWithName | varchar(500) | Refer to pathWithName field in datasetName\_user\_tags\_decision table. |
| accepted | tinyint(1) | Whether this decision is accepted by administrator or not. |
| userid | int(11) | The userid of the administrator: who accepted this decision. |
| confirmDate | datetime | When was this decision accepted by the administrator. |

### Table **datasetName\_web\_orders**

This table is for the orders page.

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| id | bigint(20) | Incrememtal table id |
| name | varchar(100) | Order name. If isBase = true, this field means the order category name. |
| isBase | tinyint(1) | Whether the order is the base order or a regular order under a order category. isBase = true means this record is the order category. |
| base | bigint(20) | If the order is a regular order under a base order, what is the base ID? Reger to id field in this table (datasetName\_web\_orders). |
| explanation | varchar(500) | Description of this order. Must be specified when a user creates a new order. |

### Table **datasetName\_web\_orders\_terms**

This table stores the terms in an order, either a base order or a regular order.

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| id | bigint(20) | Incrememtal table id. |
| orderID | bigint(20) | Refer to id field in datasetName\_web\_orders table. |
| name | varchar(100) | Term name. |
| isBase | tinyint(1) | We record the term orders by saving the position of each term in that order. The positioning system need a base point: which term’s position is 0. If isBase = true, the position of this term is 0. Terms before this base term have negative positions, and terms come after this base term have positive position values. |

### Table **datasetName\_user\_orders\_decisions**

This tables stores the users decisions in orders page.

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| userID | bigint(20) | Refer to usreid field in users table. |
| orderID | bigint(20) | Refer to id field in datasetName\_web\_orders table. |
| termName | varchar(100) | Refer to name field in datasetName\_web\_orders\_terms table. |
| decisionDate | datetime | When was this decision made. |
| distance | int(11) | the distance of this term to base term. |
| isTerm | tinyint(1) | Besides saving each term’s position in an order, we also save the entire order as a string (e.g. appressed->ascending->erect). We use this string to compare if there are conflicting decisions on this order. This string is saved in decision field. And when the record is about the order string, isTerm = false. |
| decision | varchar(2000) | As described above, this field is to store the entire order as a string. |
| isActive | tinyint(1) | If this decision is the most recent (active) one from this user. |
| isLatest | tinyint(1) | If this decision is the latest one from all users. |
| hasConflict | tinyint(1) | If this order has conflicting decisions. This field is set by comparing decision field of the same order. |
| isBase | tinyint(1) | When distance = 0, isBase = true, which means the term is the base term. |

### Table datasetName**\_confirmed\_orders**

This tables stores the accepted decisions by an administrator.

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| orderID | int(11) | Refer to id field in datasetName\_web\_orders table. |
| orderName | varchar(100) | Refer to name field in datasetName\_web\_orders table |
| term | varchar(100) | Refer to name field in datasetName\_web\_orders\_terms table. |
| distance | int(11) | Refer to distance field in datasetName\_user\_orders\_decisions table. |
| accepted | tinyint(1) | Will be set to be true (false) when an administrator accept (revoke) this decision on admin page/ orders page. |
| userid | int(11) | The userid of the administrator who accepted this decision. |
| confirmDate | datetime | When did the administrator accepted this decision. |

### Table datasetName**\_comments**

Record is inserted when user make a comment in the term’s detail popup window. Comments could be from group terms page, hierarchy page and orders page.

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| commentid | int(10) | Incremental ID |
| comments | varchar(1000) | Comments content. |
| term | varchar(50) | This comments is for which term if the comment is from group terms page. |
| userid | int(5) | Refer to field userid of table users. |
| commentDate | date | When was the comment made. |
| tagID | int(11) | Refer to field **tagID** in table **datasetName\_web\_tags**. Is not null when the comment is from hierarchy page. |
| orderID | int(11) | Refer to field **ID** in table **datasetName\_web\_orders**. Is not null when the comment is from orders page. |

### Table datasetName**\_sentence**

All conent of this table are uploaded by CharaParser. It contains all the sentences and will be used to generate structures in hierarchy page. It also provides sentence sources for term’s context part which exists for every page.

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| sentid | int(11) | Sentence id |
| source | varchar(500) | Source file name |
| sentence | varchar(2000) | Sentence content |
| originalsent | varchar(2000) | Original sentence content |
| lead | varchar(50) |  |
| status | varchar(20) |  |
| tag | varchar(150) |  |
| modifier | varchar(150) |  |
| charsegment | varchar(500) |  |
| checked | tinyint(1) |  |

## update Database

If you are going to update the database, no matter database schema or just data, always back up the database on the server first.

Test all the changes thoroughly on your local machine and also test the changes on the real-time database by copying the database on server to your local machine.

Keep record of the updates in file /src/dbUpdates.sql.

There are two ways to update the database. One is just run sql commands on the server. Second is to write a function to update the dabatase. This is very useful when you need to change each dataset with a series of sql commands. Examples can be found in class UpdateServerDatabaseAction. You can use url *biosemantics.arizona.edu/OTO/updateServerDatabase.do* to update the server. Make sure the update won’t be executed twice or it might cause problems.

# File Organization

|  |  |
| --- | --- |
| **Folder/File** | **Description** |
| /css | All the css files of OTO. |
| /docs | The old documentation about database schema. Not up to date. |
| /downloadFiles | This folder must be there. It stores all the glossary files ever generated from OTO. |
| /downloadFiles/readme.txt | This file must be there. It will be used to pack the .zip glossary file. |
| /images | All the images files of OTO. |
| /js | All the javascript files of OTO. |
| /jsp | All the .jsp files of OTO. |
| /src | Empty folder. Not used. |
| /swfs | All the video files for help page. |
| /WEB-INF | Contains the configuration files, source code, library .jar files and compiled java classes. |
| /WEB-INF/classes | Compiled java classes. |
| /WEB-INF/classes/config.properties | Server configuration files such as database, bioportal, github and etc. |
| /WEB-INF/classes/edu/arizona/sirls | The compiled java classes. |
| /WEB-INF/lib | External library files OTO uses. |
| /WEB-INF/src | Source code of java classes. |
| /work | Empty folder. Not used. |

# How to update Server

OTO is currently hosted on server biosemantics.arizona.edu, which also hosts OTOLite, fnasearch and GgServer and other websites. The server has both apache and tomcat. The configuration files related to OTO are below:

1. The tomcat configuration file: /var/lib/tomcat5/apache-tomcat-5.5.28/conf/server.xml file. Find the following line in this file.

<Context path="/OTO" docBase="/home/sirls/hongcui/WebFilesForTomcat/OTO" reloadable="true" crossContext="true"/>

You can see that the OTO context path is /home/sirls/hongcui/WebFilesForTomcat/OTO. So when you update the server, you should go to this folder and update the content.

When you only need to change .jsp or .js or .css or image files, you don’t need to restart tomcat or reload. Simply replace the files and they will take effect instantly.

But if you need to change any configuration of OTO of java classes of OTO, you need to reload the class files or restart tomcat if reload doesn’t work. No matter which way you’re going to use, do not do it at work time because there maybe some users using our service and either way will cause the loss of their unsaved work. We don’t want to get those complains.

* To reload OTO class files:
  + First, replace the files you need to change. To avoid any possible mistake, always back up the folder before you replace anything.
  + Then, go to <http://biosemantics.arizona.edu:8080/> -> Tomcat Manager (User: manager Pass: tomcat2x , the credentials of users are defined in /conf/tomcat-users.xml)
  + After you login, you may be able to start/stop/reload/undeploy OTO without interfering other websites.
* To restart tomcat server:
  + The location is tomcat on the server is /var/lib/tomcat5/apache-tomcat-5.5.28/. Go into the /bin folder and use sudo ./shutdown.sh or sudo ./startup.sh to shut down or re-start tomcat.
  + Sometimes, shutdown doesn’t work because some java process cannot be killed. In this case you can use the command sudo netstat –tlnp | grep 8080 to see which process in using port 8080. Then use sudo kill -9 processID to force kill the process. Then you should be able to shut down and re-start tomcat.

1. The apache configuration file: /etc/httpd/conf/httpd.conf. Find the following lines in this file. It will direct biosemantics.arizona.edu/OTO to the local url <http://localhost:8080/OTO> which is OTO in tomcat.

<Proxy /OTO>

Order deny,allow

Allow from all

</Proxy>

ProxyPass /OTO http://localhost:8080/OTO

ProxyPassReverse /OTO <http://localhost:8080/OTO>

# How to read the code

* Start with .jsp files

OTO is a web application project. So the interface is mainly in .jsp files.

The code in .jsp files are mixed with java code and html code. It is not the reccommended way to write .jsp pages, but it is very clear of the logic.

* Read javascript files

.jsp files may contain one or more javascript files. The links are defined in <head><script></script></head> in .jsp files.

* Get into the java code

As described before, .jsp files contains java code too. First, java classes are declared and imported into the .jsp file and then java code is written in <%%>.

# How to debug

1. Create a tomcat server in eclipse.
2. set the context path to direct to OTO files.
3. Debug run the tomcat server.
4. Go debug!

# How to add an action

1. Write the action class.

You can find examples in package edu.arizona.sirls.action. All classes in this package are action classes.

* 1. The class should extend ParserAction
  2. Parse out input
     1. Form input: Form has to be defined first and configured in struts-config.xml. E.g. UserLoginAction.java
     2. Xml input: use GeneralForm and get the input as string, then parse out the xml. E.g. SaveGroupAction.java
  3. Return
     1. Forward to another page (can be the same page): e.g. SaveGroupAction.java

Return mapping.findForward(Forwardable.*RELOAD*);

* + 1. Write response text. E.g. GetContextAction.java

1. Config the action in struts-config.xml. Simply follow existing examples.
2. Call the action
   1. Through http get: type in url/actionName.do
   2. Through http post: take a look at login form in login.jsp as an example.
   3. In javascript: take a look at getContext() function in context.js as an example.

# How is drag-and-drop implemented

In OTO, drag-and-drop is implemented with three javascript functions (example in categorize.js file):

* mouse\_down\_handler(e): make a clone of the widget that is dragged
* mouse\_move\_handler(e): set the position of dragged object
* mouse\_up\_handler(e): do dropping related stuff.

# How is context part implemented

All javascript functions in context.js file.

# Ajax in Javascript

Get the example in getContext() in context.js file.

# Major functions

## Save Data

Data sent in xml format. Example can be found in save\_categories() function in categorize.js file.

## Merge Datasets

Code can be found in function mergeDatasets in CharacterDBAccess.java class.

## Finalize Dataset

Code can be found in function finalizeDataset in CharacterDBAccess.java class.