Chado Controller User Manual

V1.0.0

Preface

The Chado Controller is a middleware that wraps a Chado Database to improve its possibilities. It enables access restriction to Chado data, verifies annotator work and keeps track of modifications made to the database.

This user manual targets annotators that use Artemis genome editor but also people using GBrowse genome browser. It describes how the Chado Controller is integrated to these tools and how to take advantage of it.

Table of content

Basics

The Chado Controller is a middleware mainly embedded in a Chado Database. It is composed of 3 main parts:

1) the Access Restriction module;
2) the Annotation Inspector;
3) the Annotation History module;

Access Restriction Module

The Access Restriction module manages the access to any feature of the database (chromosome, contig, gene, mRNA, protein, regions,…).

Logging in and logging out

When using GBrowse, users may have to login in order to access to protected features. To log in, there should be a box on the top of GBrowse labelled 'User Account' with 2 fields: 'Login' and 'Password'. Simply enter your login and password and click on the button 'Login'. The page should be reloaded and a display the message 'Welcome <*your login*>!' if you logged in successfully.
Once logged in, you can change your password using the link 'Change password...' which displays two fields, one for the new password and a second one to type the new password again to ensure there are no mistyping. Then, the 'change password' button will proceed to the password update.
Be careful, after login, you have to have to update image in the GBrowse page to see all protected tracks (Frequently Asked Questions p. 13).
With Artemis, the user just has to log into the Chado database using the regular Artemis login box and the Access Restriction module will manage his/her access rights automatically. New created feature will be given the same access right of the object they are located on (ie. through Chado featureloc table).

When using front-end programs to PostgreSQL such as 'psql' command line program, you should manually initialize the Access Restriction module using the SQL command:
"SELECT init_access();"
If you don't do so, your first query using the feature table will be slower than expected as the Access Restriction module will proceed to a self-initialization and treat your query in a non-optimized way.

Annotation Inspector

The Annotation Inspector helps annotators to produce consistent annotations. It is based on controlled vocabularies (CV). CV owned by the Chado Controller are prefixed by 'CC_'. The CC cvterms are available through the CV tab of Artemis feature builder or at http://www.gnpannot.org/sites/gnpannot.org/files/chado_controller_1.6.0.tgz
The Annotation Inspector automates some annotation tasks so the annotator doesn't have to handle these anymore such as:

- add new feature to the manual annotation track (update feature_dbxref source as source and type are required by GBrowse);
- change the color feature property of annotated features according to their annotation status (colours are selected by the administrator at installation time);

- set the owner property of a gene to current user;
- add relationship entries between sub-elements of a transposable element.

The Annotation Inspector can also be called to check the consistency of an annotation. When integrated to Artemis, the Annotation Inspector is automatically called when the annotator uses the 'Commit' button. Then, it performs its checks and either just commits if everything was fine or displays a dialog box reporting encountered issues. When issues are detected, the annotator can commit his/her changes anyway or cancel to fix the problems. If the Annotation Inspector is told to commit while issues were reported, it will add properties to the annotated element indicating which issues were detected.

When using front-end programs to PostgreSQL such as 'psql' command line program, the Annotation Inspector can be called manually.
To check the whole database for annotation issues, you can use the SQL command:
"SELECT * FROM validate_annotations(0, FALSE);"
To start a new manual annotation and check it for issues do the following queries:
"SELECT start_new_transaction_group();"
Then, note the returned value which is the transaction group identifier that will be used later on.
"START TRANSACTION;"
Do your annotation SQL queries.
"SELECT, INSERT, UPDATE, DELETE,…"
Once done, to check your work, do:
"SELECT * FROM validate_annotations(<identifier>, FALSE);"
where "<identifier>" is the transaction group identifier you got earlier. That call will return 2 fields: a filed 'validation' which is set to 0 if no issue has been encountered and a field 'validation_message' containing encountered issues description. Then you have 3 choices:

- you can just ignore the Annotation Inspector results and commit:
  "COMMIT;"
- you may want to commit but keep track of what was wrong:
  "SELECT * FROM validate_annotations(<identifier>, **TRUE**);
  COMMIT;"
- or cancel all your changes:
  "ROLLBACK;".

To check an older annotation, you need administrator access right to get the annotation transaction identifier of the annotation. This identifier can be found in the '*_audit' tables (Annotation History p. 11). Typically, a query to retrieve such an identifier looks like this:
"SELECT transaction_group FROM <table>_audit WHERE <come condition to identify the annotation> GROUP BY transaction_group, transaction_date ORDER BY transaction_date DESC;"
For instance, to retrieve annotations made on a feature between September the 1st and September the 2nd:
"SELECT transaction_group FROM feature_audit WHERE '2011-09-01 00:00:00' < transaction_date AND transaction_date < '2011-09-02 23:59:59' GROUP BY transaction_group, transaction_date ORDER BY transaction_date DESC;"

The checks performed by a standard installation of the Annotation Inspector and its behavior when 'validate_annotations' is called with 'TRUE' as second argument are listed in Table 1.
Note: each one of the above functions can be called the same way 'validate_annotations' is.

Table 1. Annotation rules of the Chado Controller manual annotation tracking module.
Functions called by the validate_annotations function: auto-fill triggers, check and manage procedures.
*****admin account: includes both administrator and database loading accounts

Function	Annotation Inspector Rule	Default behavior
Auto-owner	Set owner to current user for new polypeptide or repeat region except when using admin account*	handled by triggers
	Set owner to current user on polypeptide or repeat region when a qualifier is added or modified except when using admin account*	handled by triggers
	Set owner to current user on polypeptide or repeat region when a gene element position or a CV term or a feature dbxref is added or modified except when using admin account*	handled by triggers
Auto-manual Curation	Add any modified features to manual curation track (feature_dbxref relationship)	handled by triggers
Auto-Color	Set color of related features of gene or repeat region to the color choosen at Chado Controller installation time when a gene related or a repeat region related feature is modified	handled by triggers
Auto-TE Relationship	Insert missing feature_relationship entries between repeat region related features when any of them is modified	handled by triggers

Function	Annotation Inspector Rule	Default behavior	Commit behavi
check_gene_structure	Check if the obsolete status of an element is consistent with its associated elements	for non-obsolete gene: - report missing non-obsolete mRNA - report missing non-obsolete polypeptide - report missing non-obsolete exon - make sure gene name is shared between gene elements - report last stop codon shared between at least 2 different non-obsolete genes for obsolete gene: - report non-obsolete mRNA - report non-obsolete polypeptide - report non-obsolete exon for deleted gene: - report non-deleted mRNA - report non-deleted polypeptide - report non-deleted exon for non-obsolete mRNA: - report missing non-obsolete gene - report missing non-obsolete polypeptide - report missing non-obsolete exon for obsolete mRNA: - report non-obsolete gene - report non-obsolete polypeptide - report non-obsolete exon for deleted mRNA: - report non-deleted gene - report non-deleted polypeptide - report non-deleted exon	add /redundant_gene q if a stop codon is share between at least 2 gen otherwise, remove /redundant_gene quali

Function	Default behavior	Commit behavior
check_start_stop_codons	report invalid start or stop codon	add "missing_start_codon" CV term for invalid start codon add "missing_stop_codon" CV term for invalid stop codon
check_sequence	report sequence length which are not a multiple of 3 add "peptide" CV term if polypeptide lenght is below 60bp report any stop codon found inside the coding sequence	add "not_3-multiple" CV term if sequence length is not a mu of 3 add "peptide" CV term if polypeptide lenght is below 60bp add "stop_in_frame" CV term if only one stop codon has be found inside the coding sequence add "multiple_stop_in_frame" CV term if more than a stop c has been found inside the coding sequence remove "stop_in_frame" and "multiple_stop_in_frame" if no codon has been found inside coding sequence
check_introns	report negative intron length report unrecognized intron donor site (non-GT) report unrecognized intron acceptor site (non-AG)	add "negative_intron_length" CV term when negative intron length detected add "missing_donor" CV term when an unrecognized intron donor site (non-GT) is found add "missing_acceptor" CV term when an unrecognized int acceptor site (non-AG) is found
manage_evidence	add "curated" CV term when a feature is modified	same behavior
manage_note	auto-set note qualifier content to something like: "name~ product~ gene~ completeness" for genes or "rpt_class~ rpt_order~ rpt_superfamily~ rpt_family~ name~ rpt_type~ completeness" and fill "/mobile_element" qualifier for repeat regions or "satellite~ name" and fill "/satellite" qualifier for satellites	same behavior
manage_transposable_element_gene	add "/transposable_element_gene" qualifier set to 1 for genes inside a repeat region or a transposon or if the gene has one of the detected keyword in its "product" qualifier or if the gene ha a detected IPR code as dbxref if the "/transposable_element_gene" has already been set, its value remains unchanged (even if it is set to 0)	same behavior
manage_mandatory_properties	report missing "product" qualifier report missing "/functional_completeness" qualifier report missing "/status" qualifier report missing "/evidence" qualifier report missing or not set "/inference" qualifier	same behavior

Function	Annotation Inspector Rule	Default behavior	Commit beha
manage_evidence_code_coherence	Report inconsistency between gene qualifiers and the selected evidence code	report missing or invalid "evidence_code" CV term	same behavior
	Check consistency of "evidence_code" set to "IC1" Similarity with a polypeptide whose function has been experimentally demonstrated in the studied organism OR in the same genus (product is the validated function of the cognate polypeptide)	report missing "product", "GO terms" CV terms or "Dbxref" PMID	
	Check consistency of "evidence_code" set to "IC2" or "IC2a" High similarity with a polypeptide of validated function (product is the validated function of the ortholog)	report missing "product", "GO terms" CV terms or "Dbxref" PMID	
	Check consistency of "evidence_code" set to "IC2b" High similarity with a polypeptide of known function (product is the known function of the ortholog)	report missing "product" or "GO terms" CV terms	
	Check consistency of "evidence_code" set to "IC3" Similarity with Swissprot/TrEMBL polypeptide or InterPro family (product is the putative function of the homolog)	report unwanted "gene" (synonym) CV term warn if "product" does not contain "putative" keyword report missing "GO terms"	
	Check consistency of "evidence_code" set to "IC4" Similarity with polypeptide of unknown function or interspecies EST (product is conserved hypothetical protein)	report unwanted "gene" (synonym) or "ec_number" CV terms warn if "product" does not contain "conserved hypothetical protein" keywords warn if there is not just one "GO terms" set to "molecular function"	
	Check consistency of "evidence_code" set to "IC5" No significant blast hit (product is hypothetical protein)	report unwanted "gene" (synonym) or "ec_number" CV terms warn if "product" does not contain "hypothetical protein" keywords warn if there is not just one "GO terms" set to "molecular function" warn if there are "Dbxref" PMID	
	Check consistency of "evidence_code" set to "IC6" No significant functionnal prediction, short coding sequence and/or low coding probability (product is doubtful protein)	report unwanted "gene" (synonym) or "ec_number" CV terms warn if "product" does not contain "hypothetical protein" keywords warn if there is not just one "GO terms" set to "molecular function" warn if there are "Dbxref" PMID	
	Check consistency of "evidence_code" set to "IC7" Very partial match and strong anomalies of the gene structure (product is remant gene symbol)	report unwanted "gene" (synonym) or "ec_number" CV terms warn if there is not just one "product" warn if the "product" contains "hypothetical" or "putative" keywords warn if there is not just one "GO terms" set to "molecular function" warn if there are "Dbxref" PMID	

Annotation History

The Annotation History module keeps track of every insertion, update or deletion made on Chado tables. Annotation history includes the login of the user who performed the changes and the date of the operation. Moreover, changes are grouped into transaction groups and their order is recorded. When a transaction group identifier is positive, it means changes were made without calling the function 'start_new_transaction_group()'. On the opposite, if the identifier is negative, it means that 'start_new_transaction_group()' has been called. So you can easily differentiate what was done with a Chado Controller compliant soft such as Artemis.

The history of a gene or a transposable element can be accessed from GBrowse using the script:
http://<your GBrowse site>/cgi-bin/ gbrowse_history/<your Chado instance>?name=<your gene>
Where "<your GBrowse site>" is the name of the server hosting your GBrowse, "<your Chado instance>" is the name of the Chado instance you use and "<your gene>" is the name of the gene of interest.
Note: you can also get the URL of the history page of a feature by replacing "gbrowse_details" with "gbrowse_history" if you have the GBrowse details page URL.
Example:
http://gnpannot.cirad.fr/cgi-bin/gbrowse_history/musa?name=MaC088K20_g300
Login: guest, password: guest

GBrowse history page can display 2 kinds of reports: one for gene (or polypeptides) features and one for other kinds of features.
For genes, only the history of the following properties will be displayed (while the history of other properties is also in database):

- Feature fields (feature table): locus_tag (name), length (seqlen)
- Feature properties (featureprop table): owner, note, inference, annotator_comment
- Feature controlled vocabulary terms (feature_cvtem table): product, functional completeness, gene, EC_number
- Feature database cross-references (feature_dbxref table): PMID

For other features, displayed properties are: owner, note, comment, annotator_comment, inference, length, Functional Completeness, Evidence Code, Gene, locus_tag and PMID.

Note: only the available properties will be displayed.

The annotation history page displays group of transactions in colored blocks with the date and the author of the changes for each group of transaction. Group of transaction are stored by date, the most recent being the first block displayed. Properties that have been changed during a transaction group are displayed in bold. When multiple changes occur on a same property, a plus sign ([+]) is displayed to show all the changes made.

Currently, it is not possible to restore an old annotation. If you want to do so, you will have to copy the old annotation and paste it on your annotation editor.

Troubleshooting

1) I can’t log in!
Make sure you use the appropriate login and password and “Caps Lock” on your keyboard is not turned on. If you are really sure your password is correct, see with your administrator. He/she can have access to log files that could provide additional information on the source of the problem.

2) I can login with GBrowse but not with Artemis!
The Chado Controller can not use PostgreSQL account passwords to authenticate users. Therefore, it is possible that the password of the Chado account is desynchronized with the password of the PostgreSQL account. If you change your password using the GBrowse interface, it might help to resynchronize your passwords. If the problem remains, see with your administrator.

3) I can’t access to the tracks I’m supposed to!
First, make sure you are logged in using the appropriate account. Then, make sure your administrator granted you the appropriate rights on the tracks.

Frequently Asked Questions

1) How do I know if I'm logged in using a specific login?
On GBrowse, the Access Restriction module adds a small area (box) entitled "User Account". In that area, if you're logged in, there will be a button starting with "Logout" followed by the name of the user account you are currently using.

2) Why do I have to reload GBrowse page after login to see all protected tracks?
Your administrator may hide tracks to anonymous users using GBrowse config. Unfortunately, for technical reasons, when you log in, GBrowse needs to process the config to know how to authenticate you. At the time it reads the config, you are not authenticated yet, and the tracks to be hidden to anonymous users remain hidden to you. Then the page loads and you become authenticated. You have to reload the page in order to let GBrowse reload its config and display the hidden tracks.

3) Some checks made by the Annotation Inspector are not relevant. Can I disable them?
To disable some checks made by the Annotation Inspector, you have to ask your administrator to do so. The Annotation Inspector calls functions that can be disabled by the administrator through the table 'annotation_inspector_procedures'.
Note that checks can not be disabled for a specific user: any change will be applied to all users.

Contacts

valentin.guignon@cirad.fr
stephanie.sidibe-bocs@cirad.fr

Glossary

Annotation History: it is a module of the Chado Controller package which records every modification made on data. It is composed of PostgreSQL scripts embedded in the database and some parts of interface in GBrowse to display the annotation history. The annotation history module is based on a modified version of the Chado Audit module to extend its initial possibilities.

Annotation Inspector: it is a module of the Chado Controller which automates some annotation tasks and can be used to check the consistency of annotations. It is composed of PostgreSQL scripts embedded in the database and some parts of interface in Artemis to display inspector messages. It is based on controlled vocabularies (CV).

Access Restriction: it is a module of the Chado Controller which enables access control to features of a Chado Database. It is composed of PostgreSQL scripts embedded in the database and some parts of interface in GBrowse or Artemis for initialisation of the module or user login.

Chado: Chado is a relational database schema that underlies many GMOD installations. It is capable of representing many of the general classes of data frequently encountered in modern biology such as sequence, sequence comparisons, phenotypes, genotypes, ontologies, publications, and phylogeny.

Chado Controller: it is a middleware between a Chado database and user interfaces that use it. It is composed of 3 main modules: Access Restriction module, Annotation Inspector module and Annotation History module.

Controlled Vocabulary: a controlled vocabulary is a list of terms grouped under a vocabulary name. It helps the annotator to find the allowed terms and prevents the creation of duplicate terms often due to typo (*e.g.* product, gene symbol, EC number, functional completeness, structural completeness, status, evidence, evidence code).

PostgreSQL: it is the relational database management system that handles Chado databases.