The University of Kansas

INFORMATION & TELECOMMUNICATION TECHNOLOGY CENTER
The University of Kansas

Technical Report

Bioinformatics Computational Journal: User Guide

Victor Frost, Terry Clark, Susan Gauch,
Gerald Lushington, Gary Minden, Ed Komp,
Adam Hoch, David Johnson, Lance Feagan,
Alexander Garrett, Justin Rohrer, Heather
Amthauer, Andrew Ozor

ITTC-FY2008-TR-38270-04

November 2007

Project Sponsor:
Edgewood Chemical Biological Center
Aberdeen Proving Ground, Maryland;
Dr. Kevin O'Connell
kevin.oconnell1@us.army.mil
U.S Army Contract W911SR-04-C-0087

Appendix A

Bioinformatics Computational Journal: User Guide

Investigators

Victor Frost
Terry Clark
Susan Gauch
Gerald Lushington
Gary Minden

Staff Researchers

Ed Komp
Adam Hock
David Johnson

Student Researchers

Lance Feagan
Alexander Garrett
Justin Rohrer
Heather Amthauer
Andrew Ozor

ITTC-FY2006-38270-A1
US Army Contract
W911SR-04-C-0087

Date: December 2006

Information and Telecommunications Technology Center
University of Kansas
2335 Irving Hill Dr.
Lawrence, Kansas 66045
Phone: (785) 864-4833   FAX:(785) 864-7789
e-mail: frost@eecs.ku.edu
http://www.ittc.ku.edu/

Project Sponsor
Edgewood Chemical Biological Center
Aberdeen Proving Ground, Maryland
Dr. Kevin O'Connell
kevin.oconnell1@us.army.mil

Table of Contents

Table of Figures

1 The Computational Journal Environment Overview

1.1 Background

The principal function of the Computational Journal (CJ) is to help users effectively perform computational experiments (with a particular focus on bioinformatics applications). *Computational experiment* is simply the term we use for running one more computer programs on a data set. We use *experiment* to emphasize the correspondence between this action and a biologist performing an experiment in a traditional wet lab. Likewise the CJ contains the record of these computational experiments, analogous to the record that the biologist maintains of his/her wet lab experiments in his/her lab notebook.

The CJ environment supports an arbitrary number of *journals*, which provide an organizational structure for the storage of entries. Journals are hierarchical, that is a journal can be stored in another journal, analogous to "directories" or "folders" in computer interfaces.

An *entry* (laboratory journal "entry") is the basic unit of storage in the CJ environment. It represents a unit of information or data in a specific format. Every entry is characterized by the format of the data it stores, that is, its *content type*. The size of an entry may vary widely, depending on its content type, from a short textual entry, to an entire chromosome. The CJ environment uses the *content type* to determine how an entry can be used and manipulated in the environment. For example, the CJ automatically determines the kind of editor to use for an entry based on the entry's content type. When searching for entries to use as an input to an experiment, the search is limited to entries with a content type compatible with that of the experiment to be run. The CJ environment supports the definition of new *content types* to support new application programs and new research domains.

When an entry is created, a set of metadata is immediately collected and stored with the entry. This metadata includes information such as: title, owner, access rights, creation time and content type. Therefore, all entries contain a certain amount of common information that can be used to help locate and/or organize the potentially large number of entries.

1.2 Entry Categories

Although the number of content types is large and extensible, every entry can be identified with one of four major functional categories, which are described below.

1.2.1 Imported Data

The CJ environment is self-contained. Any data that is used as input to an experiment must be stored in a CJ entry. Therefore data from external sources and/or wet lab experiments must be imported into the CJ environment.

1.2.2 Experiment Definition

The definition and execution of computational experiments are core tasks for the CJ environment. The CJ provides graphical block diagram editors and includes a number of pre-defined content types that support these activities. A thorough description of a *workflow* and *experiment definition* will be presented in later sections.

1.2.3 Experiment Output

Each output generated by executing a computational experiment is stored in a new entry. The content type of each output entry is determined by the experiment that was run. For example, if a `BLAST` search was performed, then the output of the experiment will be a `BLAST_Report`.

Output entries can also be exported. This operation makes a copy of the content of the entry on user's local computer. This allows users to communicate results to other people who are not CJ users (e.g. a collaborator at another university) or exchange data with the wet-lab. It is also useful if the user wants to perform further processing of the data by resources or programs not currently available in the CJ.

1.2.4 Annotations

Annotations are additional entries created directly by the user that can be attached to entries of the other three functional categories. All annotations are searchable. Typically, annotations are notes and comments that help explain and/or highlight key features of the entry to which it is attached. A variety of content types are frequently used for annotations, including text, images and urls. In addition, entries can be searched indirectly by searching the content of annotations attached to entries.

1.3 Entry Modification

Most CJ entries are read-only objects. They contain data that should not be changed, for example the output of laboratory experiment (or of a computational experiment performed through the CJ). The underlying philosophy of the CJ is to record events (experiments and the results they generate) as they occur. Experiments frequently are repeated with different inputs and refined over the course of time. These changes are treated as new events to be recorded – not as modifications of an existing object.

Every entry carries a *committed* attribute. If this attribute is `True` ("the entry is committed"), it is read-only and may never be modified by anyone in the future. Many entries are automatically marked *committed* upon creation, such as when data is imported from an external source. This attribute plays a fundamental role in the CJ environment. It promotes sharing and collaboration among CJ users, since it provides assurance that the entry will never be deleted or modified regardless of the owner of the entry. If the inputs to an experiment are committed, then the user is assured that the dependencies of an experiment will remain available so that the experiment can be reviewed and/or re-executed in the future.

Even for entries not yet committed, modification is carefully controlled in the CJ environment. All modification is restricted solely to the original creator of the entry. Until the entry is committed, the owner is able to save and modify the entry as often as he/she chooses. However, an entry that remains in an editable state cannot be used as a dependency of another entry. For example, you cannot run an experiment based on a workflow that is in an editable state -- because the content and/or parameter settings in the workflow might be changed after the experiment is run, and losing the details of the experiment definition. Therefore, you must use the *Commit* operation to set the committed attribute for an entry, before you can use it in other contexts. Note, however, that once you have set the committed attribute, that you will not be able to modify the entry.

For committed entries a browser (an editor with all operations that modify the data removed/disabled) are provided to view the data. Likewise, if you open an entry owned by another user, the entry contents will be opened in a browser, since you can never modify an entry owned by another user.

1.4 Entry Dependencies

The CJ environment recognizes and maintains dependency information among entries as they are created. Some examples of dependencies include:

1. Each experiment output is dependent on the experiment definition.
2. An experiment definition depends on the workflow it executes, and each of the data sources attached to the workflow inputs.
3. A workflow definition depends on the resource and workflow definitions used in its implementation.

This dependency information provides two major services:

1. Assurance of a complete provenance record for generated data.
   The CJ does not allow the removal of an entry if other entry(ies) depend upon it. This facilitates the sharing and reuse of data, since a CJ user incurs no risk of the loss of the data when he/she uses the results generated by another user as input to a new experiment.
2. Convenient access to related data / information regardless of naming conventions, author, etc. Embedding this information directly in the CJ infrastructure, relieves the user of many of the responsibilities of maintaining documentation of experiment dependencies, generating and following naming conventions for files to locate related data, etc. This facilitates collaboration within large teams and among teams since this

dependency information is available through a common interface for all entries in the CJ environment. The *dependent* (and corresponding *dependency*) *sub-viewer* in the *navigation view*, provide immediate access to the recursive requirements for (dependents of) the selected entry in the *navigation view*.

1.4.1 Navigation by Dependency

The CJ environment provides navigation tools based directly on the dependency information that is automatically collected and maintained among entries. The SubNavigator, *Dependent Manager* , provides a tree-based view of all entries which depend directly, or indirectly, on the entry selected in the primary *Navigator* view. (See Section 3 for details about the *Navigator* view.) At the top level appear all entries that directly depend on the selected entry. Beneath each of these entries will appear the entries that directly depend on this entry, and so on.

This sub-navigator provides a very focused view of entries closely related to an entry of interest. It is able to collect these entries regardless of the owner and/or containing journal. This reduces the user burden for carefully organizing the layout of files in the file system to keep related data in the same or related directory. It also eliminates the tendency to make copies of data, in order to keep it close to related experiments when the data is used as an input to multiple experiments. The SubNavigator, *Dependency Manager*, provides a similar view of the entries on which the selected entry depends.

1.5 Workflows

A workflow is a graphical representation of a *computational experiment*. This section describes workflows and their components.

1.5.1 Resource Definition

In the CJ, a *resource* is the graphical representation of a core computer program. This set of core programs is extensible. Section 2.4.1 provides details of how to enter a new *resource* into the CJ environment via an XML description. Figure 1 displays an instance of the BLASTn resource.

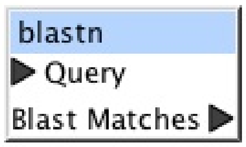


Figure 1. BLASTn resource

Each resource has three principal components:

1. Input Port(s) represents a data input to the program, for example the query sequence supplied to the BLASTn program.

Input ports are represented by a small triangle pointing into the block and a label, identifying the specific input. Some programs accept optional inputs, which are not required for the program to execute properly. These ports appear shaded rather than a solid color.
Help information supplies additional information about the type of data required and expected content.

2. Output Port(s) represents data generated by the program, for example, the blast report containing the results of a BLASTn run.
   Output ports are represented by a label, identifying the specific output, and a small triangle pointing out of the block,
3. Parameter(s) are arguments to the program that control its execution behavior. For example, the expectation value supplied to BLASTn affect the uniqueness of the alignments. By setting the expectation value to 1, we want to get one chance hit with this score to our query using this particular database.
   Parameters do not have a graphical representation. They appear as properties of the resource in the *Properties* View when a resource instance is selected in the *Workflow Editor*.

1.5.2 Workflow Definition

Generally, however, a user wants to define a more complex experiment than simply executing a single core program. A data source can be used as an input to one resource (program) and the output of this resource used as input to another resource, and so on.

Another aspect of an experiment that a user may want to control is the number of variables. In addition to the data source input(s), each parameter represents another variable in the experiment. In order to meaningfully compare the results of two experiments, one may want to ensure that various resource parameter values are the same for both experiments.

The *Workflow Editor* allows the user to conveniently control both of these aspects of more complex experiments: which resources appear in the experiment and how they are connected; and which parameter values are constant and which remain as experiment variables.

The *Workflow Editor* is defined in detail in section 2.4. Here we highlight its role in helping users define computational experiments. *Workflows* closely resemble resources described in the previous paragraphs. A resource represents a basic computer program. The *Workflow Editor* essentially allows the user to construct more complex programs by graphically combining simple programs (resources and other workflows). A workflow has the same three components as a resource:

1. Input Port(s).
2. Output Port(s).
3. Parameter(s)

Plus a fourth component:

4. A connected set of resources defining the functionality of the workflow.

The user defines the functionality of a workflow by placing instances of resources (and/or other previously defined workflows) and drawing connections from output ports to input ports.

The *Workflow Editor* also provides actions to add input and output ports to a workflow. These correspond to the inputs (outputs) of basic resources, the external interface of the workflow being defined. The user draws connections from these input (output) ports to appropriate input (output) of the resources defining the functionality of this workflow.

Finally, to complete the workflow definition, the user must assign values to the parameters of each of the resources in the workflow. For each parameter the user has two choices, either:

1. assign a specific value for the parameter. In this case, the parameter value for use (execution) of this workflow is guaranteed to be this value. This becomes a constant for this workflow.
2. *Promote* the parameter to the workflow interface. This creates a parameter for the workflow definition with the same name, type and value restrictions as the original parameter. This parameter becomes a variable for the workflow being defined. Each time the user executes an experiment based on this workflow, he/she must provide a value for this parameter; and when the experiment is run, the user supplied value will be used as the value of the parameter in the resource from which it was promoted.

Figure 2 provides an example of a simple workflow definition.

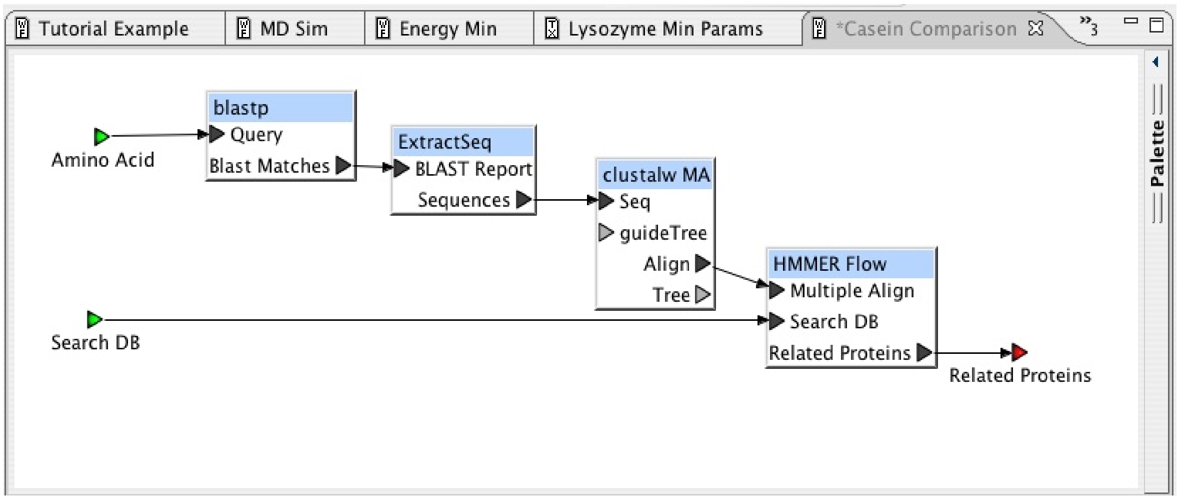


Figure 2. Workflow definition.

1.6 Experiments

1.6.1 Experiment Definition

In order to run an experiment, you implicitly create a new entry containing an experiment definition. In addition to specifying the workflow to execute, you must:

1. Assign a data source for each input port of the workflow.
   To do this, you select an entry containing the data to supply for this input.
2. Allocate a new entry for each output of the workflow.
   To do this, you simply supply a unique name for the entry. The system will automatically generate the correct values for ownership, data type and remaining fields. When the experiment completes, the output data generated for the corresponding port will be stored in this entry's content.
3. Assign a value to each parameter of the workflow.

The experiment definition stores this information and automatically generates links between the various components of the definition, maintaining critical provenance information. Whenever you select an entry representing an experiment output, you can retrieve the experiment definition from which it was generated. This allows you to view all the parameter settings for the experiment and the data sources supplied to the experiment. If a data source was itself an output of an earlier experiment, you can likewise trace back its provenance.

At the completion of an experiment definition, it is automatically executed on the computer cluster.

1.6.2 Experiment Execution

An experiment definition can be executed only once. If you want to execute a workflow again, say with different input and/or parameter values, then you will create a new experiment definition. Then you will be able to easily compare the results from the two different runs; and in the future easily be able to determine the differences between the experiments by tracing the output provenances.

If you decide that the output(s) of an experiment are no longer useful, then you can delete the corresponding experiment definition. Whenever you delete an experiment definition, it will also delete all output entry(s) generated by that experiment (after warning you of this effect).

The CJ handles all the details of running the computational experiment on the computer cluster for the user. You can monitor the progress of the experiment using the *Queue Manager* view, that typically appears in the lower right corner of the CJ window. This view provides a hierarchical view of all jobs running on the cluster. See section 5.6.2 for details about interacting with this view. In addition, there is the menu command, *Check Experiment Progress*. This command reports if the experiment is still running, or if it has completed successfully, or with errors.

1.6.3 Experiment Error Object

Most experiments will complete successfully, since many consistency and validation checks are performed as you define the workflow and the experiment definition. "Success" in this context means that all of the programs included in the workflow definition were able to execute without generating errors or crashing.

If an experiment terminates with errors, an *error object* is generated for the experiment that contains details of the individual steps comprising the experiment.

Internally, a data dependency graph is generated for the programs (resources) appearing in the workflow. A separate job is created and submitted to the batch queue system on the cluster for each of these programs. These jobs are scheduled by data dependency, so that a job begins only after the job(s) that generate its input(s) have completed successfully.

The *error object* contains sub-entries for each program executed, including the command script used to start the execution and the data written to stderr and stdout during execution. In addition, it contains data from intermediate results. Intermediate results are the data generated on the output(s) of resources that are used only internally as the input to another resource. With this additional information, the user will be able to understand why the experiment failed.

2 Example

2.1 Introduction

In this chapter, we will define and execute a simple computational experiment. This example provides a basic introduction to the CJ environment's capabilities. Menu items that do not pertain to the example are not discussed in this chapter, but will be covered in a later chapter.

From the introduction, we learned that journals are an organizational device. The content that can be contained in journals includes other journals, input information, output information, annotations, and experimental design data (experimental design data are characterized by workflows). We will be addressing each of these forms of data in this example.

The experiment we will be implementing in this example is trying to identify a protein that may be a potential target for drug therapy in treating malaria. A block diagram of the experiment we are going to replicate can be seen in Figure 3. We are first going to find similar proteins to the human casein kinase enzyme using `BLAST`. We are then going to create a global alignment using the sequences of the similar proteins that were retrieved by `BLAST` using `CLUSTALW`. Then, using the *hmmbuild* tool of `HMMER`, we are going to build a HMM model based on this multiple alignment. After we have this model, we will calibrate it. We will then run it against the proteins in the *Plasmodium falciparum* genomic database (PlasmoDB) to identify proteins that may be similar to this model using the *hmmsearch* tool of `HMMER`. We are interested in the divergence between *P. falciparium* casein kinase and its vertebrate hosts. If there is enough divergence, then specific inhibition of this enzyme offers itself as a potential drug therapy for malaria.

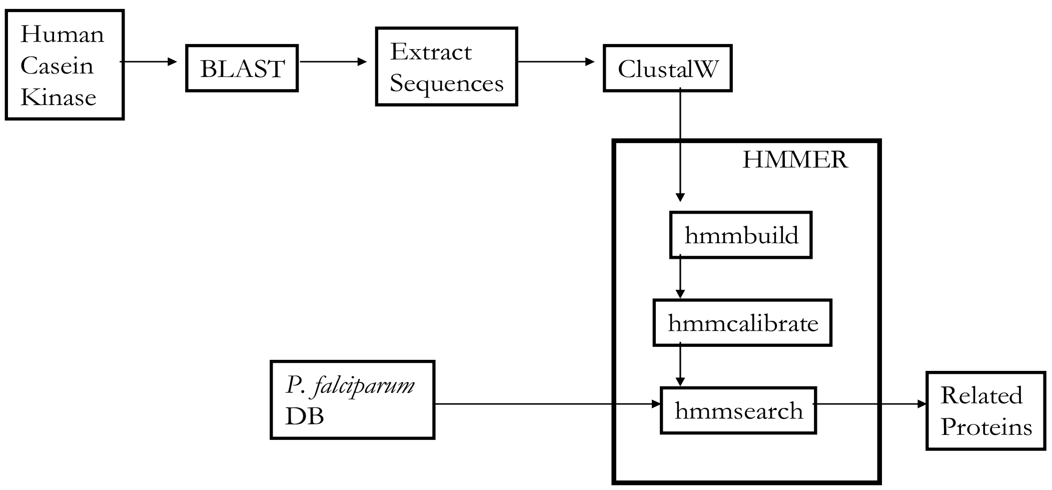


Figure 3. Flow chart of the experiment.

To implement this example, we will:

1. Create a new journal
2. Create entries that define the experiment
3. Create an annotation
4. Execute the experiment

2.2 Creating a new journal

The creation of a journal occurs in the *Navigator Shell* of the CJ environment. In the *Navigator Shell*, there is an icon that resembles a book that will create a new journal. See Figure 4 to see the icon.

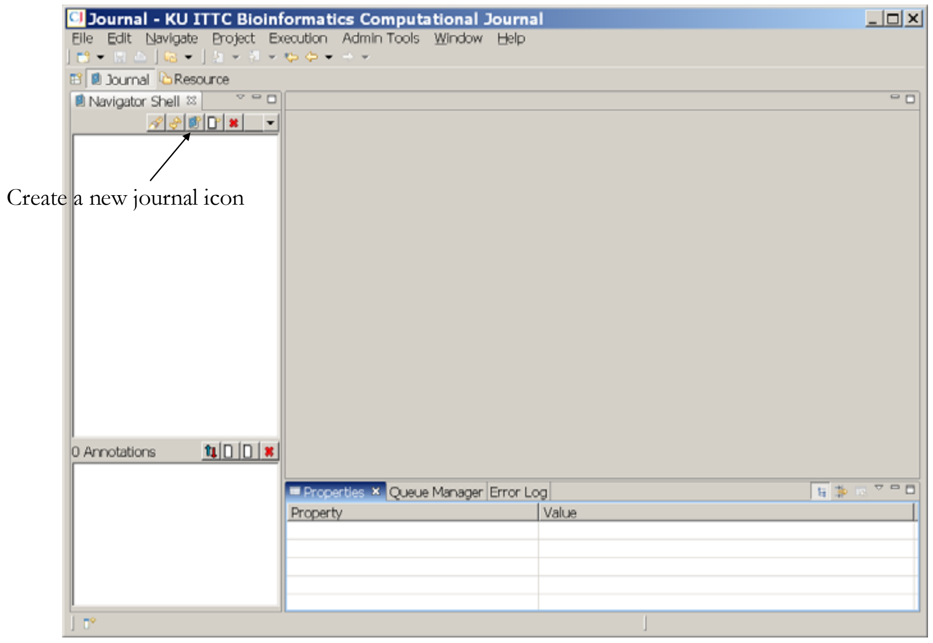
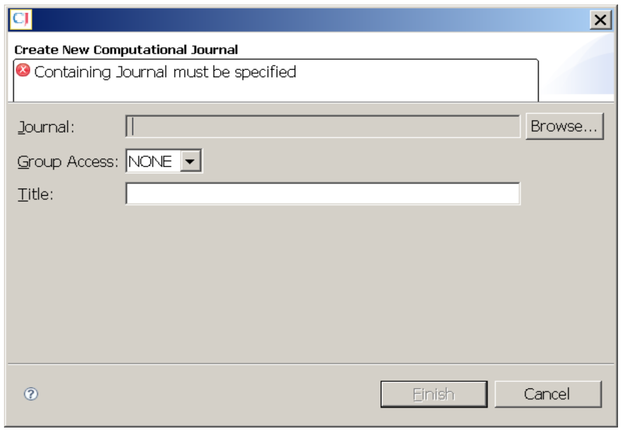

Journal - KU ITTC Bioinformatics Computational Journal
File Edit Navigate Project Execution Admin Tools Window Help
Journal Resource
Navigator Shell
Create a new journal icon
0 Annotations
Properties Queue Manager Error Log
Property Value

Figure 4. *Create new journal* icon in the *Navigator Shell*.

With your mouse, click on the icon to *Create a new journal*. When you do this, a dialogue box will appear (See Figure 5). The first thing we will have to specify is in what journal we want this new journal to be placed. The hierarchal structure of the CJ environment allows for journals to be stored in other journals. To see a list of journals that you may place this new journal in, click on the *Browse…* button.


Create New Computational Journal

Containing Journal must be specified

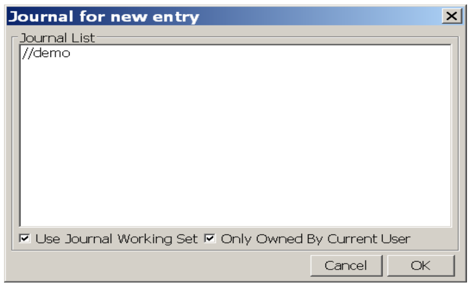
Journal:

Group Access: NONE

Title:

Figure 5. Create New Computational Journal dialog box.

Another dialogue box will appear (See Figure 6). There are two filtering options within the dialogue box that can affect the names of the journals options that will appear. By default, both options will be selected. When "Use Journal Working Set" is selected, only journals that appear in the user's working set (defined in the CJ preferences) will appear in the list. When "Only Owned by Current User" is selected, only journals that the user has created will appear in the list. We will place this new journal in our root journal (For each user account, a root journal entitled username is automatically created for the user.) In this example, our root journal is called *demo,* which is our username for this example. Click on the *demo* journal, and then click the *OK* button.


Journal for new entry

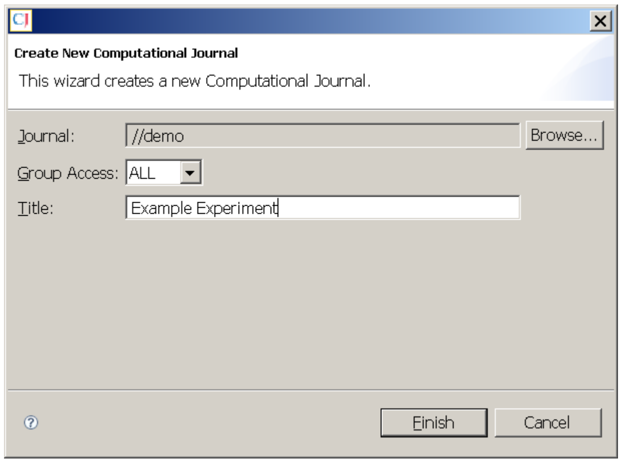
Journal List

//demo

☑ Use Journal Working Set ☑ Only Owned By Current User

Figure 6. Dialog box showing the list of journals.

Next, the *Group Access* will have to be specified in the *Create New Computational Journal* dialogue box. We will specify that the *Group Access* will be *All.* The last thing that needs to be specified is the title of this new journal. The title of this journal will be *Example Experiment.* Figure 7 will show how the dialogue box will appear with this information.


Create New Computational Journal

This wizard creates a new Computational Journal.

Journal: //demo

Group Access: ALL

Title: Example Experiment

Figure 7. Completed *Create New Journal* dialog box

To create the journal, click on the *Finish* button. The new journal, *Example Experiment*, will now appear in our root journal, *demo*, in the *Navigator Shell* (See Figure 8).

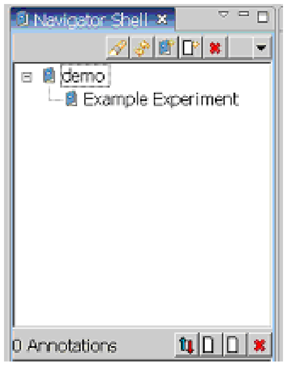


Figure 8. Navigator Shell showing the new journal just created.

2.3 Creating a new entry

Now that we have a journal, we can add entries to it. For this example, we will be making different types of entries. We will make workflow entries to implement our experiment and a FASTA entry to use as the input when we execute our experiment.

We are first going to implement the HMMER portion of the experiment that is mapped out in the block diagram in Figure 3. We will need to create an *Entry* to represent this portion of the experiment.

To create a new entry, click on the icon to *Create a new entry* in the *Navigator Shell* (See Figure 9).

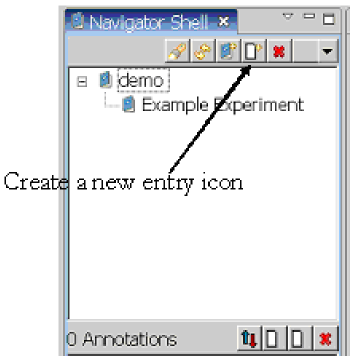
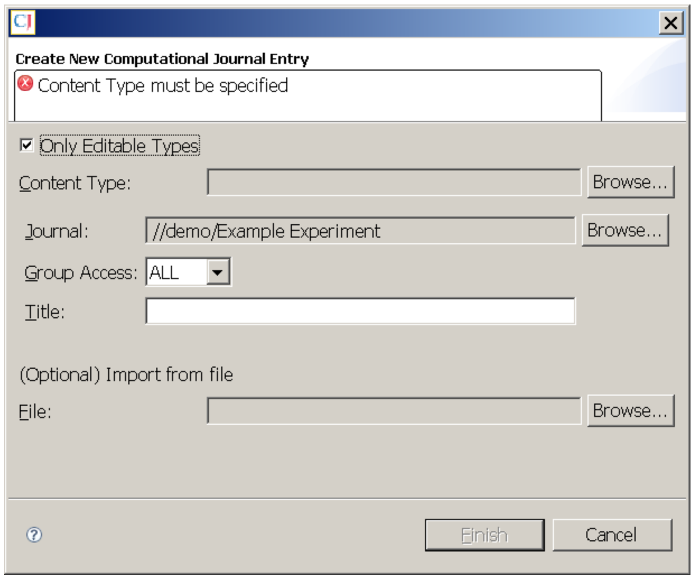


Figure 9. *Create a new entry* icon.

A dialogue box will then appear (See Figure 10).


Create New Computational Journal Entry

Content Type must be specified

☑ Only Editable Types

Content Type:

Journal: //demo/Example Experiment

Group Access: ALL

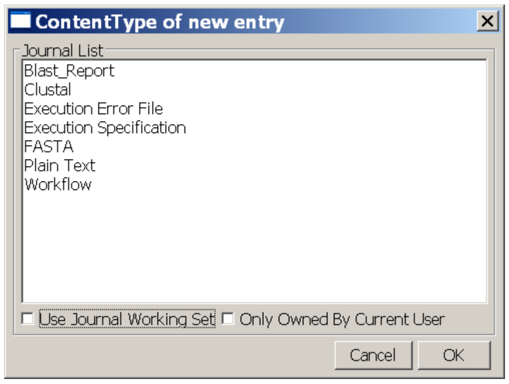
Title:

(Optional) Import from file

File:

Figure 10. Dialog box for *Create New Computational Journal Entry*.

We must specify the *Content Type* of the entry. To select a content type, click on the *Browse…* button. Another dialogue box will appear (See Figure 11).


ContentType of new entry

Journal List

Blast_Report
Clustal
Execution Error File
Execution Specification
FASTA
Plain Text
Workflow

☐ Use Journal Working Set ☐ Only Owned By Current User

Figure 11. Dialog box showing the list of Content Types.

In this example, the first entry we are going to make is a workflow, so we need to click on *Workflow*, and then click on the *OK* button. By default, the journal that we have active will be selected as the journal in which this entry will be placed. We are going to use this default setting in this example.

Again, the *Group Access* must be specified. In this case, it will determine who will be able to read the content of this *Entry*. We are going to set the *Group Access* to *All*. If *None* was selected, then only the owner of this entry could see the content of this entry. Next, we have to give this *Workflow* a title. For this entry, we will entitle it *HMMER Flow*. To create the *Entry*, we need to click the *Finish* button.

In the *Navigator Shell,* we should see our new journal and our new entry within the journal . Click on the new *HMMER Flow* entry to make it active in the *Editor Area*. Figure 12 shows us what we should be seeing.

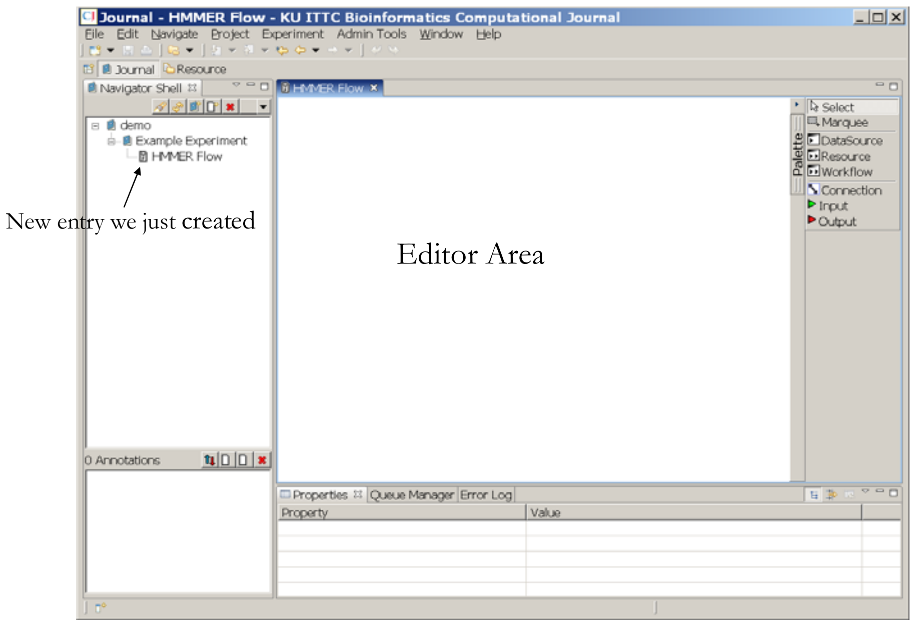


Figure 12. Shows the new entry, *HMMER Flow*, and the *Editor Area*.

2.4 Using the Workflow Editor

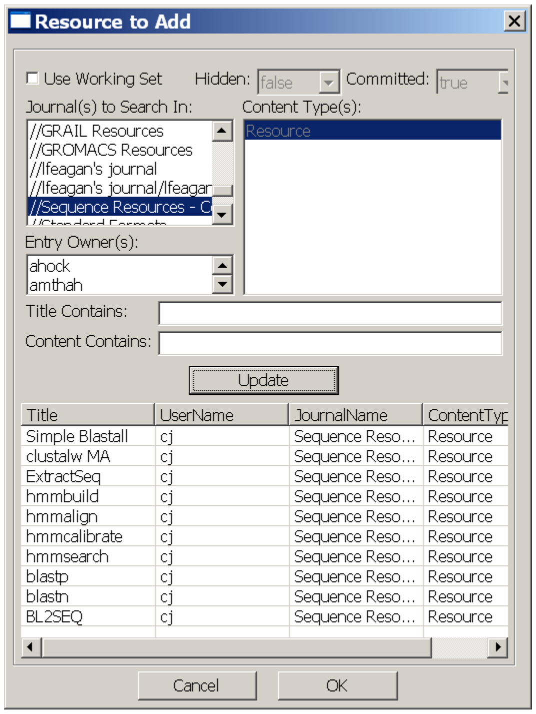
We will create the graphical representation of this experiment in the *Workflow Editor*. The *Workflow Editor* has a *Palette* of options that we can use to build our experiment.

2.4.1 Adding Resources to the Workflow

Before we start adding resources to this workflow, we need to consider our experimental design. We need to consider the resources we are planning on using and what these resources require as input and what output these resources produce. With this in mind, we can map out how these different resources can be connected.

For example, from the flow chart in Figure 3, we see that there is a block called *Extract Sequences*. This block is there because the output of the `BLAST` is a report. This is not an input format that is compatible for `CLUSTALW`. `CLUSTALW` accepts FASTA data as input, so we need to have a resource that can extract the sequences from the `BLAST` report, and retrieve their FASTA sequences to present them to `CLUSTALW`. We need to keep in mind what we can use as input and what output is produced by these resources.

In the *HMMER Flow*, we are going to add three different `HMMER` resources. To add resources, we need to click on *Resource* option in the *Palette*. We then need to move the cursor into the *Editor Area* to have the *Resource to Add* dialogue box appear. For this example, the resources that can be viewed in the *Working Set* will most likely be different from what a new user will see. This is due to the fact that in the CJ environment, resources can be added easily. It is likely that resources will be added by the time many new users read this manual. We want to use HMMER resources. These resources are located in the *//Sequence Resources - Core* Journal. We need to click on the *//Sequence Resources – Core* journal, and then we need to click on the *Update* button to view the resources that are contained in this journal. Figure 13 shows what the dialogue should look like.


Resource to Add

☐ Use Working Set Hidden: false Committed: true

Journal(s) to Search In:
//GRAIL Resources
//GROMACS Resources
//lfeagan's journal
//lfeagan's journal/lfeagar
//Sequence Resources - C

Content Type(s):
Resource

Entry Owner(s):
ahock
amthah

Title Contains:

Content Contains:

Update

Title	UserName	JournalName	ContentTyp
Simple Blastall	cj	Sequence Reso...	Resource
clustalw MA	cj	Sequence Reso...	Resource
ExtractSeq	cj	Sequence Reso...	Resource
hmmbuild	cj	Sequence Reso...	Resource
hmmalign	cj	Sequence Reso...	Resource
hmmcalibrate	cj	Sequence Reso...	Resource
hmmsearch	cj	Sequence Reso...	Resource
blastp	cj	Sequence Reso...	Resource
blastn	cj	Sequence Reso...	Resource
BL2SEQ	cj	Sequence Reso...	Resource

Cancel OK

Figure 13. *Resource to Add* Dialog box.

The first resource we are going to add is the *hmmbuild* resource. To do this, we will click on the *hmmbuild* and then click on the *OK* button. Now, place the cursor over the *Editor Area* and click on the location where we want the resource placed.

The next HMMER resource we will be adding is the *hmmcalibrate* resource. When we click on the *Resource* option in the *Palette*, the dialogue box will show the resources in the //*Sequence Resources – Core* journal. We need to click on the *hmmcalibrate* and click on the *OK* button. We then place the *hmmcalibrate* resource into the *Editor Area*. This same procedure is repeated for the *hmmsearch* resource. Figure 14 shows what the *Editor Area* should look like after the three resources have been added.

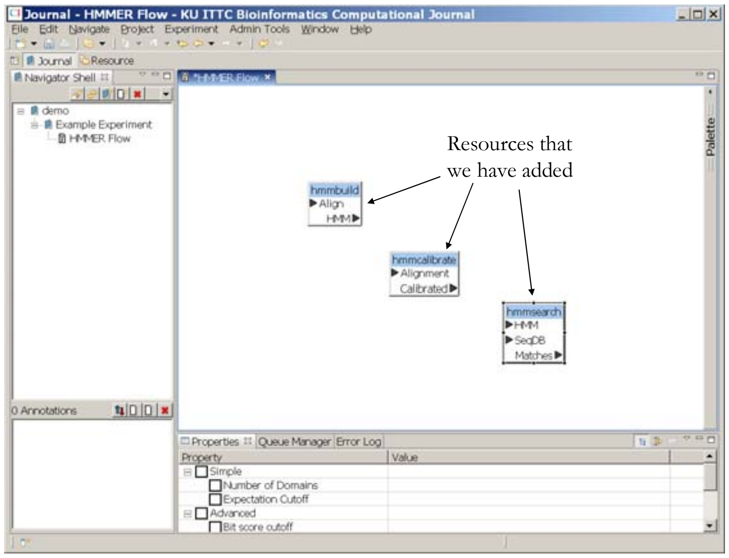
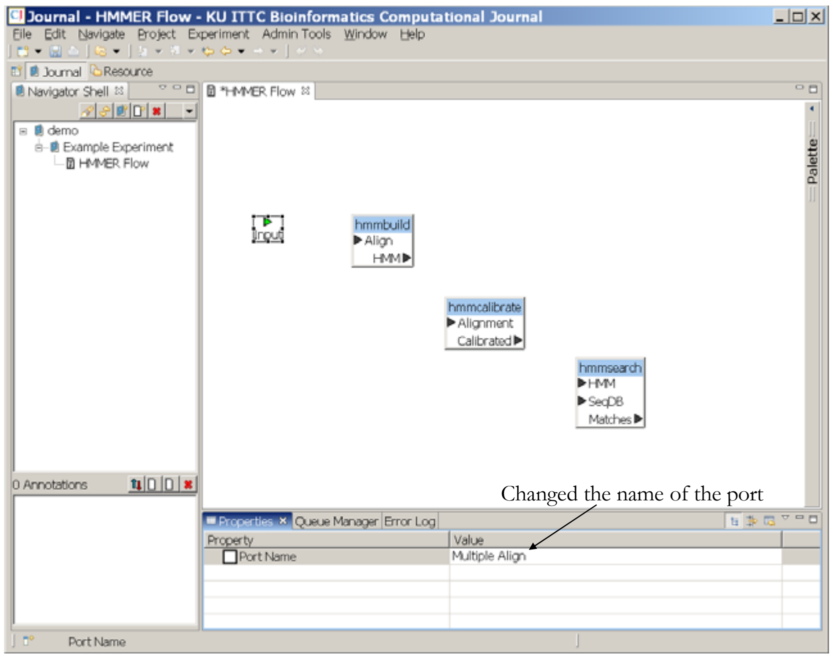


Figure 14. *Resources* added to the *HMMER Flow*.

2.4.2 Defining Workflow Inputs and Outputs

We now have to handle the inputs and outputs of these resources. The inputs to the resources appear on the left of the resource blocks. The outputs appear on the right of the resource blocks. The inputs and outputs that are aligned with the solid black arrows represent things that are required. The inputs and outputs that are aligned with the gray arrows are optional. In this example, we see that the *hmmbuild* tool needs the input of a multiple alignment. To specify this, we will click on the *Input* option in the *Palette* and then click on the area of the *Editor Area* where we want the input located. In this example, we are going to put the *Input* in the left-hand side of the *Editor Area*. With this *Input* block active, if we look in the *Properties* perspective (below the *Editor Area*), we will see the property *Port Name* with the value *Input* (See Figure 15). To give this port a more meaningful name, change the value from *Input* to *Multiple Align*.

Now, we need to connect this port, *Multiple Align,* to the appropriate port in the workflow. To do this, we need to click on the *Connection* option in the *Palette*. We then click on the *Multiple Align*'s green arrow then we click on the black arrow of the *Align* input port of the *hmmbuild* block. An arrow will appear connecting the input to the resource. Figure 16 shows what should appear in the *Editor Area* when the *Multiple Align* has been connected.



Property	Value
Port Name	Multiple Align

Figure 15. *Input* port added to the *HMMER Flow* and name change.

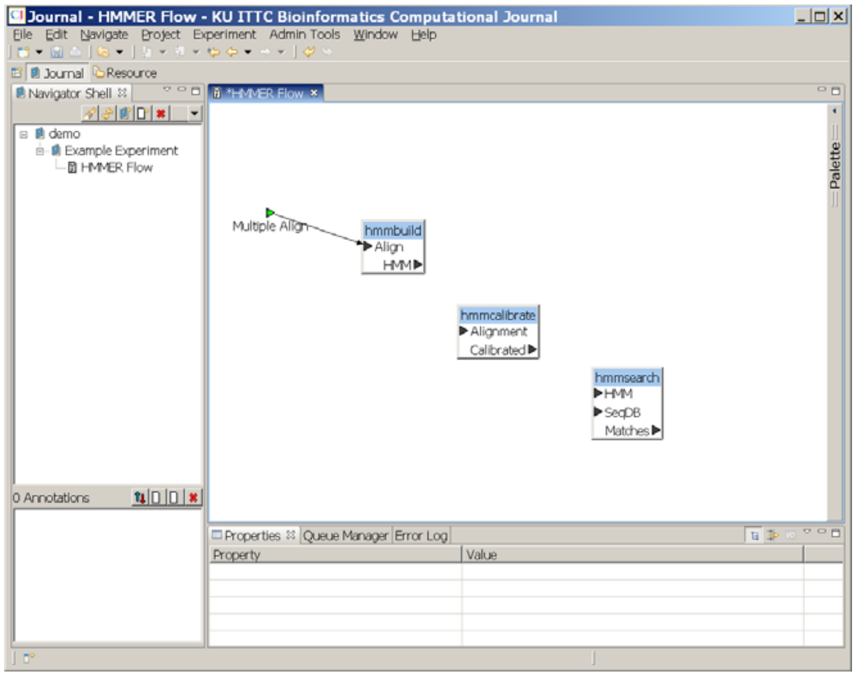


Figure 16. *HMMER Flow* with *Multiple Align* connected to *hmmbuild*'s *Align* input port.

Two of the resources have additional input options. Some of these inputs happen to be outputs of the other resources in this workflow. The *hmmcalibrate* resource has *Alignment* as an input port, this input comes from the *hmmbuild* resource's *HMM* output port. These ports need to be connected. To do this, click on the *Connection* option in the *Palette* and click on the

HMM output arrow of the *hmmbuild* resource, and then click on the *Alignment* input arrow of the *hmmcalibrate* resource.

Next we need to address the inputs for hmmsearch resource. The *HMM* input port needs to be connected to the *hmmcalibrate* resource's *Calibrated* output port. The *SeqDB* input port needs a protein database in FASTA format. This will require us to add another *Input* block to this *Workflow*. We will entitle this *Input* as *Search DB*. We then need to connect the *Search DB* to the *SeqDB* input port of the *hmmsearch* resource.

The next thing we will address is the output port of the *hmmsearch* resource, *Matches*. To do this, we click on the *Output* option in the *Palette*, and place the *Output* block in the *Editor Area* near the *hmmsearch* resource. To give the *Output* block a more descriptive and meaningful name, we will change the port name of *Output* to *Related Proteins* in the *Properties* perspective. When all of the inputs and outputs have been connected, the workflow will look like Figure 17.

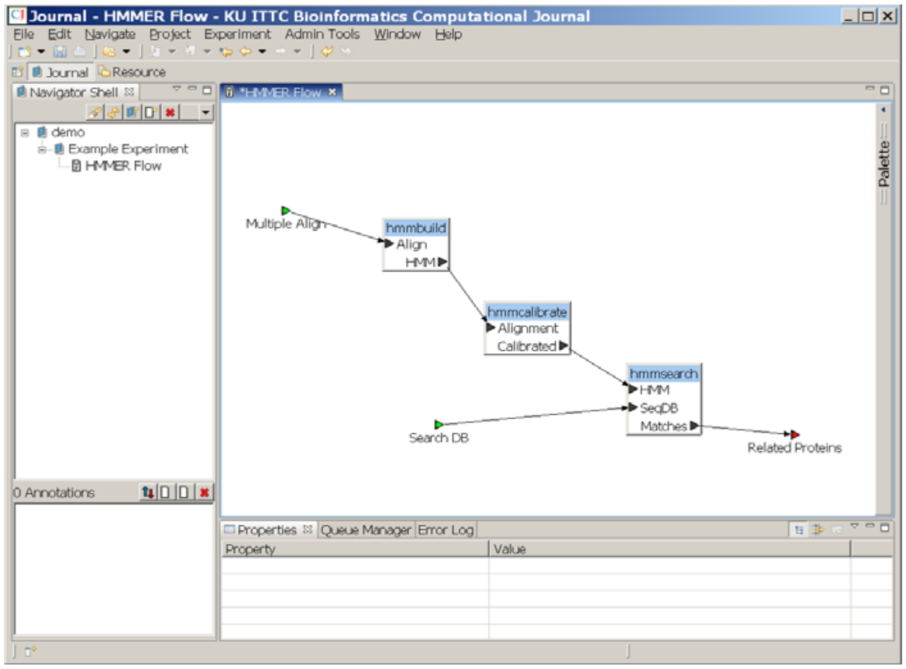


Figure 17. *HMMER Flow* with all the Resources, Inputs and Outputs connected.

2.5 Setting Properties

The *Properties* perspective is where we can set the properties of the blocks contained in the workflow (*HMMER Flow*) that we just created. To set the properties of the blocks in the *HMMER Flow*, we need to click on the block whose properties we want to set.

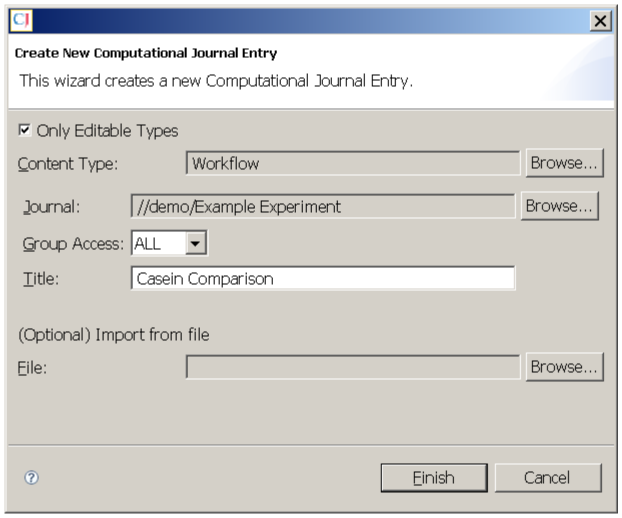
If we click on the resource blocks, the properties that we can set will appear. In this example, we are using the default values, so we do not need to change any of the values. In the next workflow that we create, we will change a value. We are now ready to save and then commit this workflow to the *CJ* environment.

2.6 Saving the Entry

To save this workflow, we need to go to the *File Menu* on the Menu bar of the workbench window. We select the *Save* option in the *File Menu*. This saves the workflow. We next need to commit the workflow. To do this, we go to the *File Menu* and select *Commit*. In Section 1.3 there is a detailed explanation of the Commit process. It should be stressed that once a workflow is committed, it cannot be altered/edited. This makes the *Commit* option different than the *Save* option. Items that are just saved can be edited multiple times and resaved. Once the *HMMER Flow* has been committed, it is ready to be used in the *CJ* environment.

2.7 Adding Workflows to other Workflows

We are now ready to implement the rest of our experiment that is mapped out in Figure 3. We will need to create another *Entry* of the *Workflow Content Type* within our *Example Experiment* journal entitled *Casein Comparison*. Figure 18 shows what the dialogue box should look like with these settings.



Create New Computational Journal Entry

This wizard creates a new Computational Journal Entry.

☑ Only Editable Types

Content Type: Workflow

Journal: //demo/Example Experiment

Group Access: ALL

Title: Casein Comparison

(Optional) Import from file

File:

Figure 18. Completed *Create New Computational Journal Entry* dialogue box.

The resources we will be adding to this workflow are *blastp* and *ExtractSeq* (Extract Sequences from BLAST report). To add these resources, we need to click on the *Resource* option in the *Palette*. These resources are in the *//Sequence Resources – Core* journal. We first need to click on this journal, then we click the *Update* button. We then need to click on *blastp* and then click the *OK* button. We then place this resource in the *Editor Area*. We repeat this process for the *ExtractSeq* resource and the *clustalw MA* resource.

The next thing we need to add is the *HMMER Flow* that we have created in the previous sections. To do this we click on the *Workflow* option in the *Palette*. To find this workflow, we need to look in our *Example Experiment* journal that we created. This is the journal that contains *HMMER Flow*. To look in this journal, we click on *Example Experiment* and then click the *Update* button. We then select the *HMMER Flow* and click the *OK* button. We then place the *HMMER Flow* in the *Editor Area*.

Again we will have to provide the proper inputs and outputs for the workflow. The resources and the inputs and outputs must then be connected. Figure 19 shows what the *Casein Comparison* workflow should look like after all the inputs and outputs have been added, and the connections have been made.

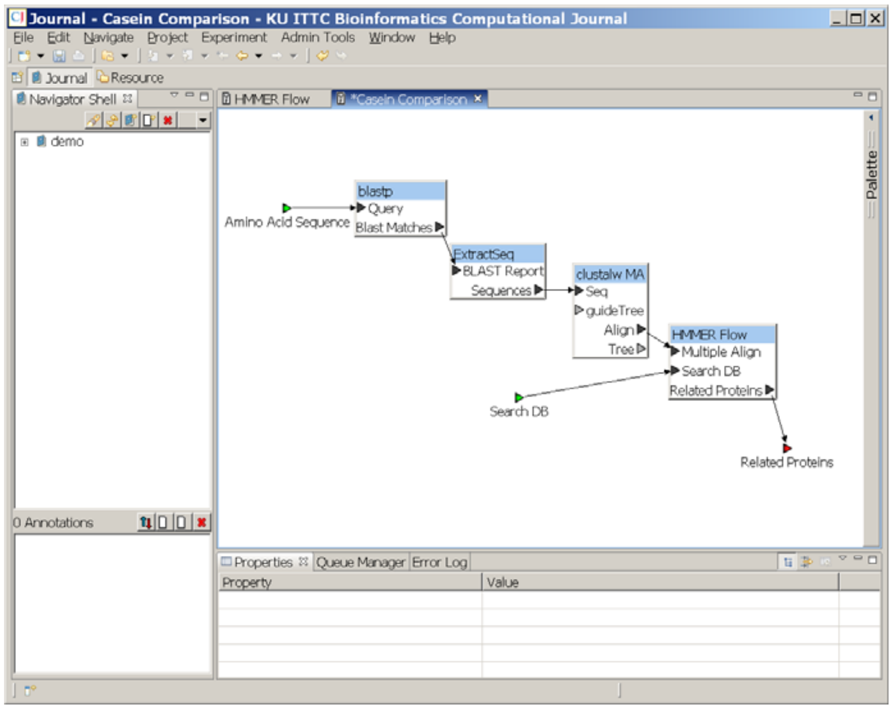
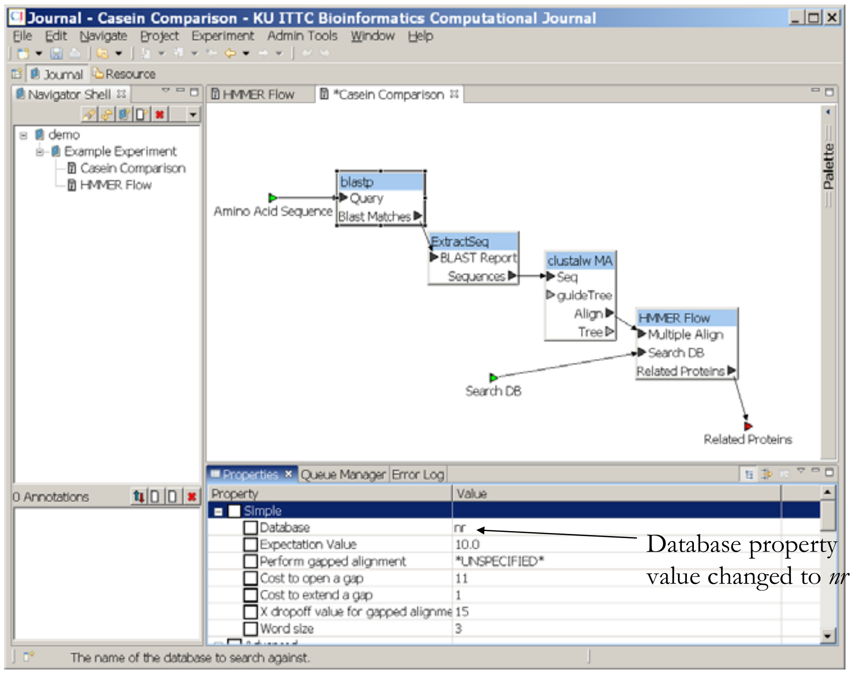


Figure 19. *Casein Comparison Workflow* with all the *Resources, Inputs* and *Outputs* connected.

We need to set the properties of the resources in this experiment. We want to change some of the default property settings for the *blastp* resource. To do this, we need to click on the *blastp* resource. In the *Properties* perspective we need to find the property called *Database*. In the value area for this property, we need to type *nr*, and for the *Expectation Value*, we need to set it to 0.001.

From Figure 20, we can see that there are check boxes by the names of each *Property*. If we check these boxes, we are allowing users to specify the values of these properties during the execution of this workflow. By not checking the boxes of properties, we make the values of the properties fixed to some specific value that we provide or to the default value. This means that every time we execute this workflow, these values cannot be altered.



Property	Value
Simple	
Database	nr
Expectation Value	10.0
Perform gapped alignment	*UNSPECIFIED*
Cost to open a gap	11
Cost to extend a gap	1
X dropoff value for gapped alignme	15
Word size	3

The name of the database to search against.

Figure 20. Setting *Database* parameter value.

We are using the default settings for the *ExtractSeq* resource and the *ClustalW* resource, so we do not need to change the values of any of these properties. The workflow is now completed. This workflow needs to be saved then committed.

2.8 Creating a FASTA Entry

From the design of this experiment, we can see that we will need FASTA inputs for the *blastp* resource. These inputs must be within the CJ environment for the CJ resources to have access to them. This means we need to make entries to represent these inputs.

To keep these forms of data better organized, we encourage users to make separate *Journals* to store different forms of data. For this example, a journal has been created entitled *ProteinSequences* to store this *Entry* that we are going to create. Before we start creating a new *Entry*, make sure that the *Journal* we want the *Entry* to be placed is active.

We will go through the same process of creating an *Entry* that we performed when creating a *Workflow*, but in this instance, the *Content Type* will be FASTA. We will entitle this *Entry* as *human casein kinase 1*. Then, we will click the *Finish* button. In the *Editor Area*, a blank *Entry* will appear. The FASTA report for this protein can be found at NCBI. The report can be copied and pasted into the blank *Entry*. It should be noted that the FASTA data does not have to be copied and pasted; it can be typed in by the user, or imported. To import files, we would use the *Import from file* option. This option selects files from the user's personal machine.

2.9 Adding an Annotation

Annotations are very valuable. They provide a means to attach additional descriptive material to entries. Common uses of annotations include: providing a detailed description of an experiment; entering the hypotheses of an experiment; highlighting the key results in an experiment output; and linking the results of one experiment to the results of another experiment. It is in the *Annotations* that we can provide the detailed description of our experiments and remind ourselves of the purpose of the experiment. *Annotations* allow us to explain what the experiment is looking for; it provides an area where we can place our hypotheses for our experiments.

Annotations can make our experiments more meaningful to other CJ journal users. We can relate to other users why we set up our experiments in certain ways and explain the logic of our experimental design and its progression.

It should also be noted that *Annotations* can be searched based on keywords. This makes it important for the language in the *Annotations* to be descriptive.

To create an *Annotation*, we will use the Sub-Navigator entitled *Annotations* that is located below the *Navigator Shell*. In this example, we are going to add an *Annotation* to the workflow we created called *Casein Comparison*. We need to select the *Casein Comparison* workflow by clicking on it. Now, we need to click on the *Create new Entry to annotate selected Entry* icon (See Figure 21.

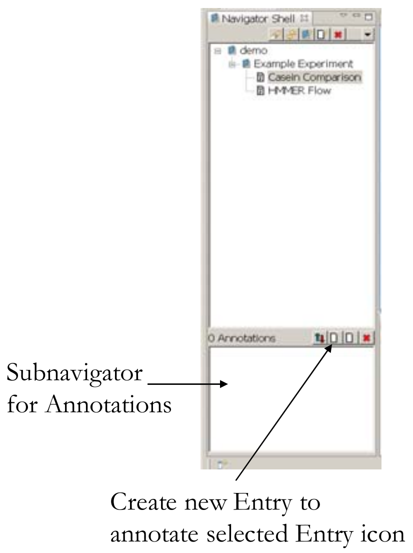
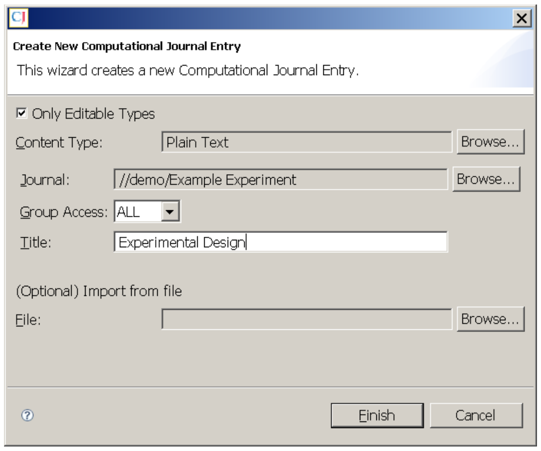


Figure 21. *Subnavigator* for *Annotations* and the icon to *Create a new Entry to annotate selected Entry*.

A dialogue box will appear. Since *Annotations* are a type of *Entry*, we will go through similar steps that we have previously gone through. We need to specify the *Content Type*. For this *Annotation*, we will select *Plain Text*. We will entitle this as *Experimental Design*. The completed dialogue box will look like Figure 22.


Create New Computational Journal Entry

This wizard creates a new Computational Journal Entry.

☑ Only Editable Types

Content Type: Plain Text

Journal: //demo/Example Experiment

Group Access: ALL

Title: Experimental Design

(Optional) Import from file

File:

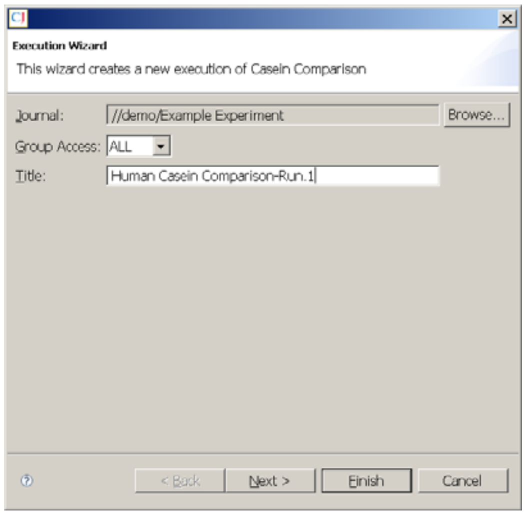
Figure 22. Completed dialog box for *Annotation*.

Once we click on the *Finish* button, in the *Editor Area*, a blank plain text document will appear. We can now type our description for this experiment. There is no spell checking capability in this editor, so be careful when typing. Once we are done typing this *Annotation*, we need to save it and commit it.

If we click on other entries that we have created in this example, we will not see the *Annotation* we created in the subnavigator. Only when *Casein Comparison* is selected will we be able to see the *Experimental Design Annotation* in the subnavigator.

2.10 Define Experiment from Workflow

Now that we have created our workflows, we can define our specific experiment from our *Casein Comparison Workflow*. To do this, we need to have the *Casein Comparison Workflow* selected. Next we need to go to the *Experiment* menu bar option. From the drop down menu, we need to select the *Define Experiment from Workflow* option. An *Execution Wizard* dialogue box will appear (See Figure 23).


Execution Wizard

This wizard creates a new execution of Casein Comparison

Journal: //demo/Example Experiment

Group Access: ALL

Title: Human Casein Comparison-Run.1

Figure 23. Execution Wizard for defining an experiment from a workflow.

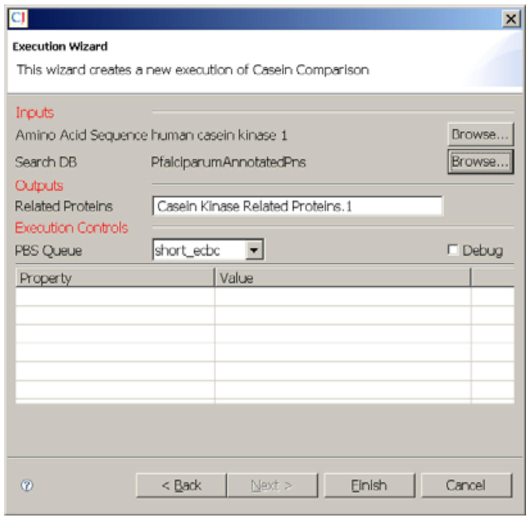
This dialogue box will allow us to change the name of this experiment. We will call this *Human Casein Comparison-Run.1*. We will then click on the *Next* button.

Another dialogue box will appear. In this dialogue box we need to provide the inputs for the experiment. We first will click the *Browse…* button by the *Amino Acid Sequence* input area. The *Amino Acid Sequence* we will use can be found in the Journal entitled *Protein Sequences*. The input data is entitled *human casein kinase 1*. We will select this *Entry*. Next, we need to select the input for *Search DB*. The input for this can be found in the *Amino Acid Sequences* journal. The input we will select is entitled *PfalciparumAnnotatedPns.*

A new entry is implicitly created for the output of this experiment. This dialog allows the user to assign the name for this entry. The dialog is initialized with a default name consisting of the associated port name and a digit. For this experiment we choose to provide a more descriptive title for the entry: *Casein Kinase Related Proteins.1*.

Next we need to define the *Execution Controls*. First, we need to specify the PBS (Portable Batch queue System) Queue. All jobs executed on the cluster are controlled by a queuing system, PBS, to facilitate sharing of the resources. Many different queues are defined on the system that control priority of the jobs submitted to them and the range of resources available to those jobs. Before an experiment is run, the project supervisor and/or the system administrator needs to be consulted about which queues can be used and the restrictions they impose on jobs submitted. It is the user's responsibility to choose an appropriate queue for the defined experiment. If inadequate resources are available in the selected queue, for example, maximum execution duration, the experiment will not be able to complete successfully.

We then need to decide if we should use the *Debug* option. If the *Debug* option is selected, an *Error Object* will be generated for the experiment. Even if it the experiment completes successfully, an *Error Object* will be generated. The additional information collected in the *Error Object* can help a user understand why an experiment does not generate the expected results even though it completes successfully. We will select this option. The completed dialogue box will look like Figure 24. We now need to click the *Finish* button.



Execution Wizard

This wizard creates a new execution of Casein Comparison

Inputs

Amino Acid Sequence human casein kinase 1

Search DB PfalciparumAnnotatedPns

Outputs

Related Proteins Casein Kinase Related Proteins.1

Execution Controls

PBS Queue short_ecbc

☐ Debug

Property	Value

Figure 24. Execution Wizard dialog box

After we click the *Finish* button, we will see a graphical representation of the experiment in the Editor Area (See Figure 25).

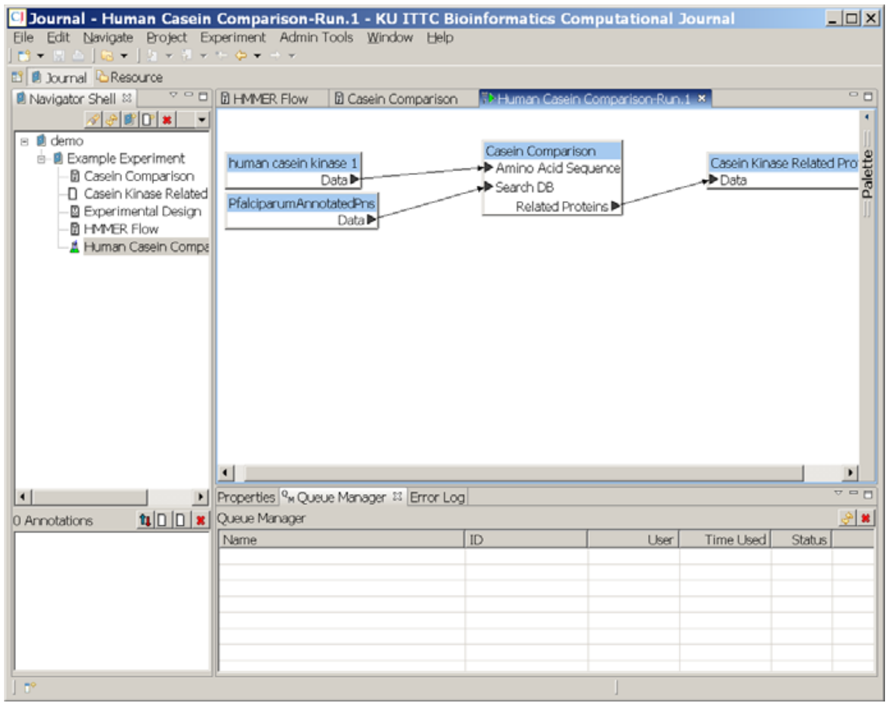
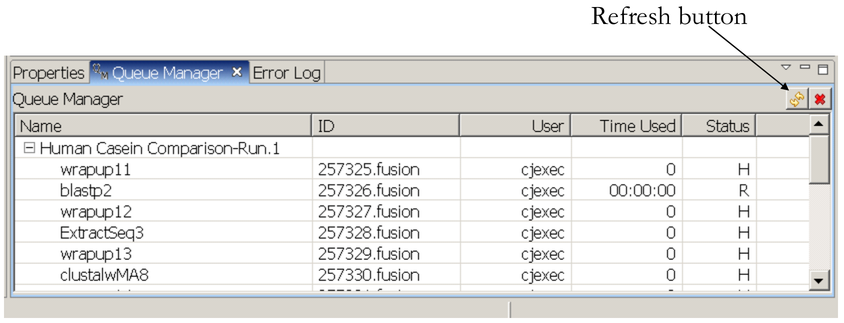


Figure 25. The graphical representation of the experiment.

We can track the progress of the experiment through the *Queue Manager*. By clicking the *Refresh* button in the *Queue Manager*'s tool bar, we can see the progression of the experiment (See Figure 26).

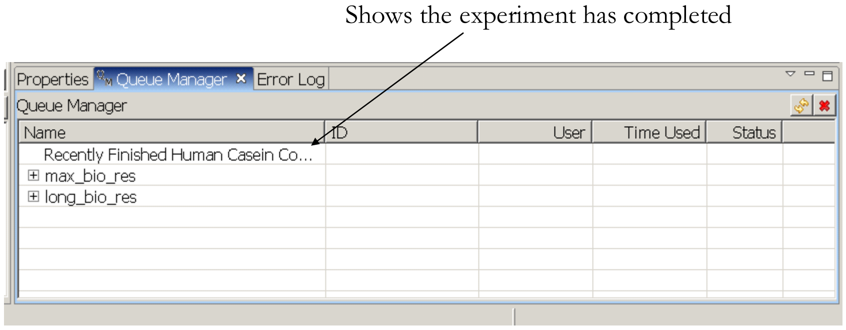


Name	ID	User	Time Used	Status
⊟ Human Casein Comparison-Run.1				
wrapup11	257325.fusion	cjexec	0	H
blastp2	257326.fusion	cjexec	00:00:00	R
wrapup12	257327.fusion	cjexec	0	H
ExtractSeq3	257328.fusion	cjexec	0	H
wrapup13	257329.fusion	cjexec	0	H
clustalwMA8	257330.fusion	cjexec	0	H

Figure 26. *Queue Manager* perspective and the *Refresh* button.

There is another option that allows us to track the experiment's progress. Selecting the *Check Experiment Progress* action from the *Experiment* Menu Bargenerates a dialogue box informing the user of the experiment's progress.

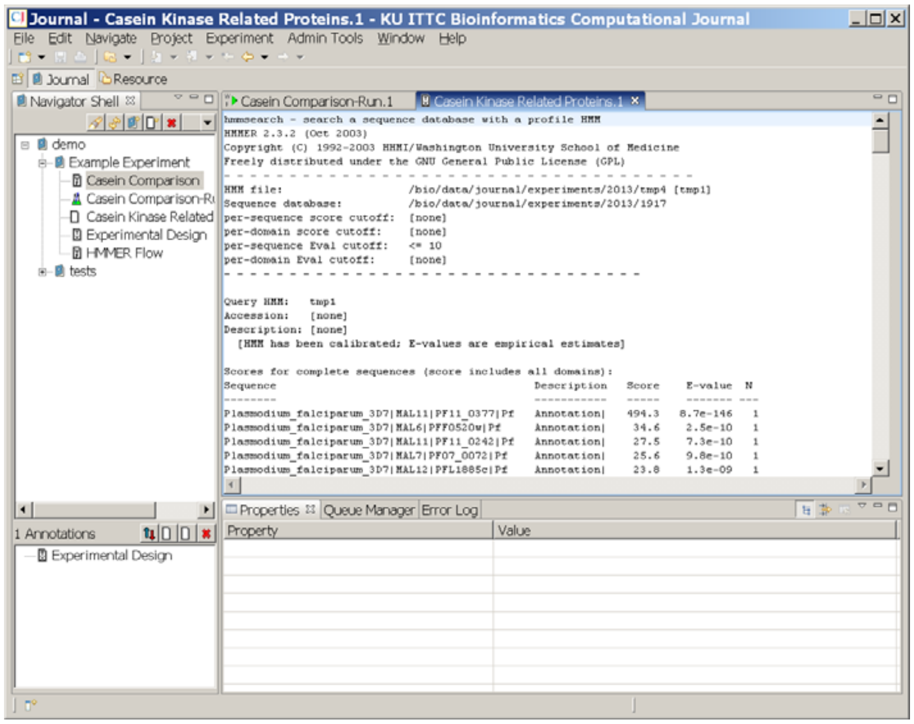
When the experiment has completed, we can see that a message in the *Queue Manager* will report that the experiment has finished (See Figure 27). If we exit the CJ environment with an experiment that has recently completed (or still running), when we return the *Queue Manager* will show us show us the "Recently Finished …" message if the experiment has finished. This message will only appear once. The next *Refresh* will not say anything about the completed experiment(s).



Name	ID	User	Time Used	Status
Recently Finished Human Casein Co…				
⊞ max_bio_res				
⊞ long_bio_res				

Figure 27. Shows that the experiment has finished execution.

The experimental output is automatically committed as soon as the experiment completes. This feature allows us to use these output(s) as datasource(s) for a new experiment. The results of the experiment can be accessed by either clicking on the *Casein Kinase Related Proteins.1* data block in *Casein Comparison-Run.1*, or by clicking on the *Casein Kinase Related Proteins.1* in the *Navigator Shell*. The results will appear in the *Editor Area* (See Figure 28).



```
hmmsearch - search a sequence database with a profile HMM
HMMER 2.3.2 (Oct 2003)
Copyright (C) 1992-2003 HHMI/Washington University School of Medicine
Freely distributed under the GNU General Public License (GPL)
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
HMM file:                 /bio/data/journal/experiments/2013/tmp4 [tmp1]
Sequence database:        /bio/data/journal/experiments/2013/1917
per-sequence score cutoff:  [none]
per-domain score cutoff:    [none]
per-sequence Eval cutoff:   <= 10
per-domain Eval cutoff:     [none]
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -


Query HMM:   tmp1
Accession:   [none]
Description: [none]
  [HMM has been calibrated; E-values are empirical estimates]

Scores for complete sequences (score includes all domains):
Sequence                                      Description   Score    E-value  N
--------                                      -----------   -----    ------- ---
Plasmodium_falciparum_3D7|MAL11|PF11_0377|Pf   Annotation|   494.3   8.7e-146   1
Plasmodium_falciparum_3D7|MAL6|PFF0520w|Pf     Annotation|    34.6    2.5e-10   1
Plasmodium_falciparum_3D7|MAL11|PF11_0242|Pf   Annotation|    27.5    7.3e-10   1
Plasmodium_falciparum_3D7|MAL7|PF07_0072|Pf    Annotation|    25.6    9.8e-10   1
Plasmodium_falciparum_3D7|MAL12|PFL1885c|Pf    Annotation|    23.8    1.3e-09   1
```

Figure 28. Shows the results of the experimental run.

Further analysis needs to be performed to see if there is enough divergence between the *P. falciparium* casein kinase and its vertebrate hosts.

3 Navigation

The CJ environment provides a variety of *Navigators*, tools that provide a hierarchical view of a collection of *Entries*. Each *Navigator* provides a virtual organization of the *Entries* based on one or more relations among the *Entries*. This allows the user to see the data organized to match a specific activity by selecting the *Navigator* most appropriate for the task. In addition, each *Navigator* provides filters that will restrict the number of *Entries* displayed.

Navigators typically do not include all *Entries* in the CJ environment. Each *navigator* attempts to restrict the number of *Entries* presented making it easier to find the *Entries* of interest for the user's current activities. If the user does not find an *Entry* that he/she is looking for in the *Navigator* Shell, the user may want to change the *Navigator* being used and/or the filters it applies. Alternatively, the user may use the search tool to query across the CJ environment. These alternatives are described in detail in the following sections.

Entries in the CJ environment are organized in *Journals* that can be viewed from the *Navigator Shell* view. *Journals* are organization devices that contain other *Journals* and *Entries*. The *Journal* representation allows the users to organize their computational experiments and related data. We can see this organizational structure in the *Navigator Shell* view. This view presents *Entries* in the default user assigned organization. Each user is allocated a top level *Journal*, and then typically defines his/her *Journals* beneath this local root; so this view provides an organization that reflects *Entry* ownership.

3.1 Entry Actions

Most operations on *Entries* are invoked from the *Navigator* view. In this section, operations on entries will be discussed.

3.1.1 Open an Entry

To edit/view the contents of an *Entry*, simply double click on the *Entry* in the *Navigator*. The CJ will implicitly locate an editor/browser appropriate for the *Entry* content, based on its content type; and open it in the *Editor Area*. Modification of *Entries* is carefully constrained in the CJ environment. The user can edit an *Entry* only if:

1. The user is the owner of the *Entry*; AND
2. The *Entry* has not been *committed*. An *Entry* that has been committed can no longer be modified.

If the user is not allowed to edit the *Entry* he/she selects to open (for either of the reasons above), the content will be presented with a browser, an editor with reduced capabilities that does not permit modification of the contents. If the user wants to make modifications to the object, he/she can select the browser's *Save As* action, to create a new *Entry* containing a copy of the data in the original *Entry*.

3.1.2 Icon Menu Bar Options

Several actions are available from the icon menu bar at the top of the *Navigator* view.

3.1.2.1 Search

The *Search* functionality allows us to search for *Entries* by specifying various search criteria. We can limit the search by specifying the *Journal*(s) in which to look, by the owner(s) of the *Entry*, by the content type of the *Entry*, by key words contained in the title, by keywords contained in the content of the *Entry*, by limiting the search to the *Journals* in the user's *Working Set*, and/or by if the *Entry* has been committed or not.

Through the *Search* tool, users with accounts can search through all the *Journals* that they are allowed access and view other users' data. This permits the sharing of experimental designs and results. Due to the read-only capabilities of this search process, the original data cannot be tampered with. If a user wants to alter any of the data (i.e. adjust test parameters), he/she must create a new *Entry* containing a copy of the original *Entry* content (using the *Save As* action).
An example of a Search follows. First, select the *Search* icon in the *Navigator Shell* (See Figure 29).

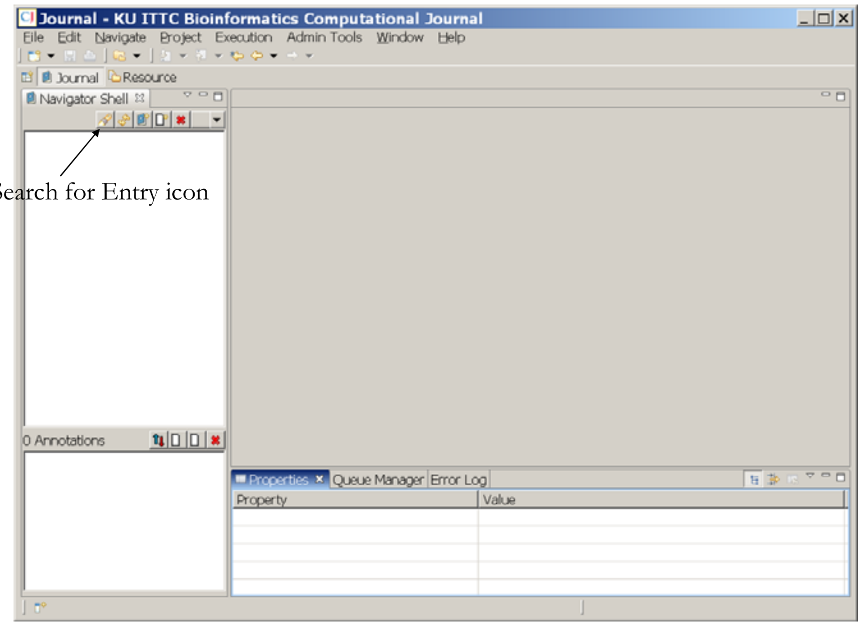
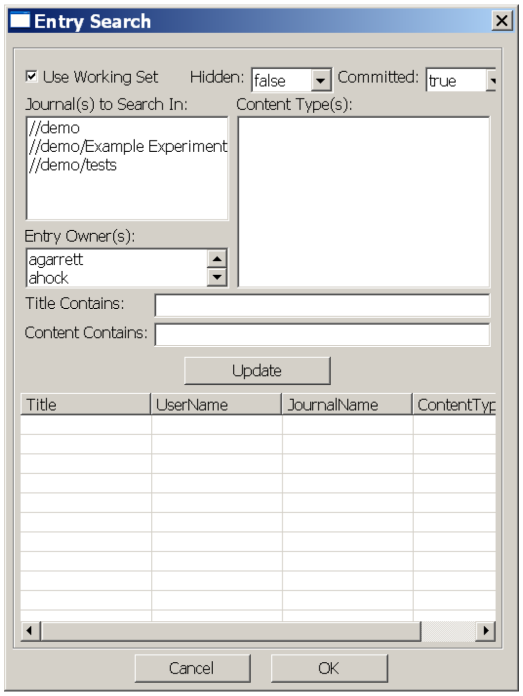

Journal - KU ITTC Bioinformatics Computational Journal
File Edit Navigate Project Execution Admin Tools Window Help
Journal
Resource
Navigator Shell
Search for Entry icon
0 Annotations
Properties
Queue Manager
Error Log
Property
Value

Figure 29. *Search* icon in the *Navigator Shell*

After clicking on the icon, a dialog box will appear. In the dialog box, the user needs to specify the search criteria he/she wants to use. In this example, a search for *Entries* that are of the *Content Type* FASTA will be performed. Figure 30 shows what the *Entry Search* dialog box looks like.



☑ Use Working Set Hidden: false Committed: true

Journal(s) to Search In:
//demo
//demo/Example Experiment
//demo/tests

Content Type(s):

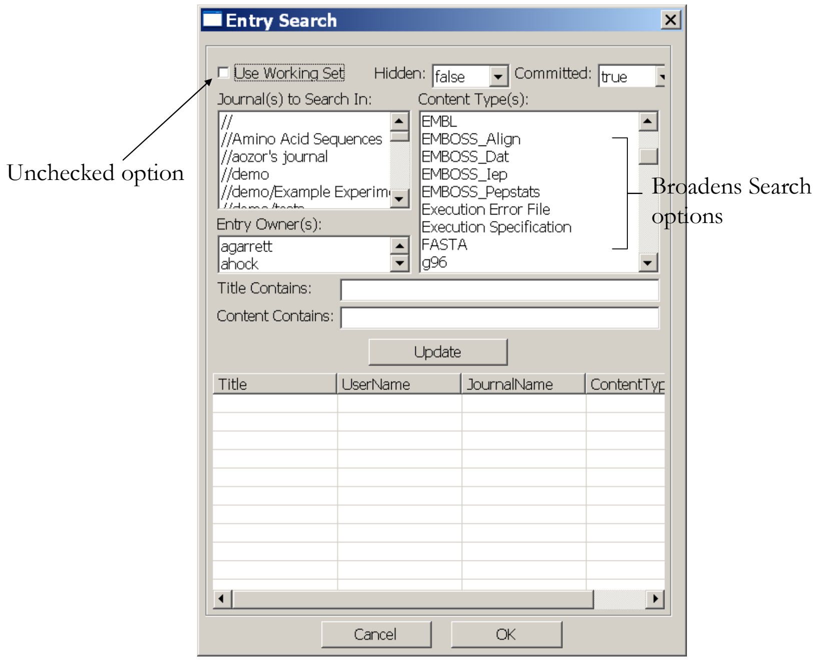
Entry Owner(s):
agarrett
ahock

Title Contains:

Content Contains:

Figure 30. *Entry Search* dialog box.

From this dialog box, the user will be by default constrained to only searching for *Entries* that are in their *Working Set*. To broaden the *Search*, to allow *Entries* outside the user's *Working Set*, the user has to uncheck the *Use Working Set* option. Figure 31 shows how the *Content Type(s)* option has been expanded.



☐ Use Working Set Hidden: false Committed: true

Journal(s) to Search In:
//
//Amino Acid Sequences
//aozor's journal
//demo
//demo/Example Experim
//demo/tests

Content Type(s):
EMBL
EMBOSS_Align
EMBOSS_Dat
EMBOSS_Iep
EMBOSS_Pepstats
Execution Error File
Execution Specification
FASTA
g96

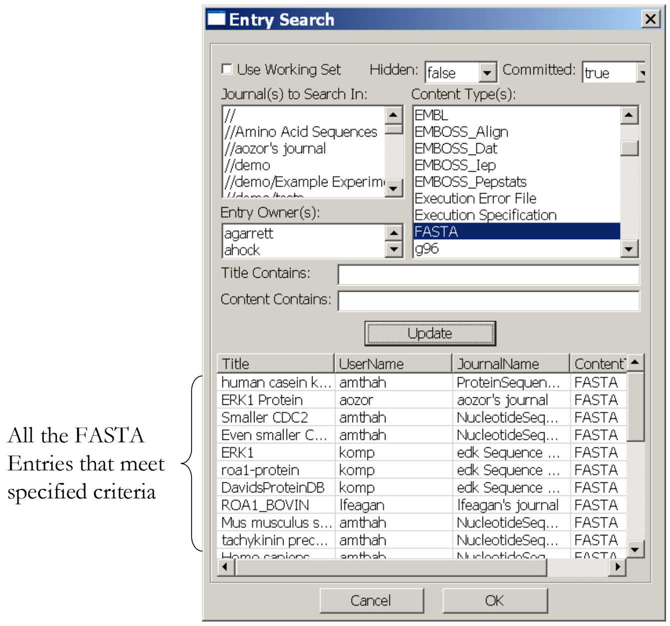
Entry Owner(s):
agarrett
ahock

Title Contains:

Content Contains:

Figure 31. Expanded *Content Type(s)* option

To search for FASTA *Entries*, the user selects the FASTA option then clicks the *Update* button. Figure 32 shows an example of the Search dialog box after Update has been clicked.



Entry Search

☐ Use Working Set Hidden: false Committed: true

Journal(s) to Search In:
//
//Amino Acid Sequences
//aozor's journal
//demo
//demo/Example Experim

Content Type(s):
EMBL
EMBOSS_Align
EMBOSS_Dat
EMBOSS_Iep
EMBOSS_Pepstats
Execution Error File
Execution Specification
FASTA
g96

Entry Owner(s):
agarrett
ahock

Title Contains:

Content Contains:

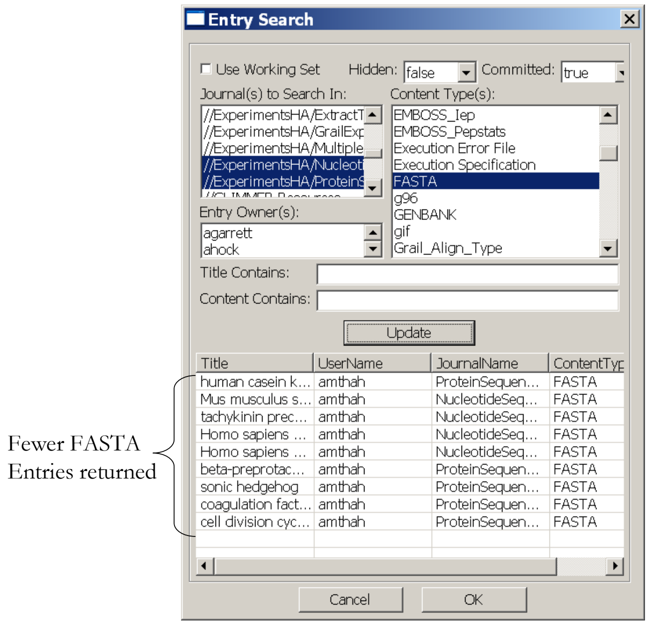
Update

Title	UserName	JournalName	Content
human casein k...	amthah	ProteinSequen...	FASTA
ERK1 Protein	aozor	aozor's journal	FASTA
Smaller CDC2	amthah	NucleotideSeq...	FASTA
Even smaller C...	amthah	NucleotideSeq...	FASTA
ERK1	komp	edk Sequence ...	FASTA
roa1-protein	komp	edk Sequence ...	FASTA
DavidsProteinDB	komp	edk Sequence ...	FASTA
ROA1_BOVIN	lfeagan	lfeagan's journal	FASTA
Mus musculus s...	amthah	NucleotideSeq...	FASTA
tachykinin prec...	amthah	NucleotideSeq...	FASTA

Cancel OK

Figure 32. Search dialog box after Update has been clicked.

Several FASTA *Entries* appear for the user to select. If too many options appear, the user can further restrict the search space by specifying further Search criteria. For example, the user can limit the *Journals* to Search in. The user can select any (or all) of the *Journals*. To select more than one of the *Journals* to search, the user has to use the Shift-Click option (i.e. hold the shift button down and use the mouse to click/select the *Journals* of interest). In this example, the *Journals* of interest are entitled *ExperimentsHA/Nucleotide Sequences* and *ExperimentsHA/Protein Sequences* (Note: these *Journals* are located in the CJ environment for the purpose of testing. They may not persist after the release of this manual). To perform the *Search* with these additional restrictions, the user has to click the *Update* button (Figure 33).



Entry Search

☐ Use Working Set Hidden: false Committed: true

Journal(s) to Search In:
//ExperimentsHA/ExtractT
//ExperimentsHA/GrailExp
//ExperimentsHA/Multiple
//ExperimentsHA/Nucleot
//ExperimentsHA/ProteinS

Content Type(s):
EMBOSS_Iep
EMBOSS_Pepstats
Execution Error File
Execution Specification
FASTA
g96
GENBANK
gif
Grail_Align_Type

Entry Owner(s):
agarrett
ahock

Title Contains:

Content Contains:

Update

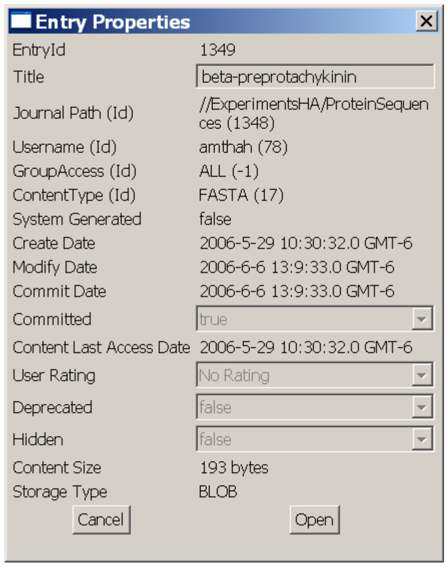
Title	UserName	JournalName	ContentTyp
human casein k...	amthah	ProteinSequen...	FASTA
Mus musculus s...	amthah	NucleotideSeq...	FASTA
tachykinin prec...	amthah	NucleotideSeq...	FASTA
Homo sapiens ...	amthah	NucleotideSeq...	FASTA
Homo sapiens ...	amthah	NucleotideSeq...	FASTA
beta-preprotac...	amthah	ProteinSequen...	FASTA
sonic hedgehog	amthah	ProteinSequen...	FASTA
coagulation fact...	amthah	ProteinSequen...	FASTA
cell division cyc...	amthah	ProteinSequen...	FASTA

Cancel OK

Figure 33. Search with additional restrictions.

The user can further restrict the search space by specifying more criteria, or the user can loosen restrictions by removing certain search criteria. When the user finds the *Entry* of interest, he/she needs to click on the *Entry*.

In this example, the *beta-preprotachykinin Entry* will be selected. Once the *Entry* has been selected, another dialog box that shows the *Entry's* properties will appear (Figure 34). To view the selected *Entry* in the *Editor Area*, click the *Open* button.



Entry Properties

EntryId	1349
Title	beta-preprotachykinin
Journal Path (Id)	//ExperimentsHA/ProteinSequences (1348)
Username (Id)	amthah (78)
GroupAccess (Id)	ALL (-1)
ContentType (Id)	FASTA (17)
System Generated	false
Create Date	2006-5-29 10:30:32.0 GMT-6
Modify Date	2006-6-6 13:9:33.0 GMT-6
Commit Date	2006-6-6 13:9:33.0 GMT-6
Committed	true
Content Last Access Date	2006-5-29 10:30:32.0 GMT-6
User Rating	No Rating
Deprecated	false
Hidden	false
Content Size	193 bytes
Storage Type	BLOB

Figure 34. *Entry Properties* dialog box for the selected *Entry*.

Variations of this search dialog are used throughout the CJ environment. Whenever the user is asked to select an *Entry* for a specific function, a search dialog is provided to help the user locate the *Entry*. For example, when defining an experiment, the user must provide an *Entry* for each experimental input. Instead of requiring the user to enter a unique identifier from memory, the user makes the selection via a search dialog. When the user input must satisfy particular constraints because of the context of is its usage (for example, an experiment input must be *committed*, and must have a content type compatible with the experiment's input type), the search dialog will constrain the search criteria to satisfy those requirements.

3.1.2.2 Refresh

The *Refresh from Persistent Store* will update the *Navigator Shell* view to reflect the current shared state of the CJ environment. This action may be required to see new *Entries* created by other users after one has started his/her session. Using the *Refresh* option will not reflect a change made by another user (for example, editing a workflow definition) if the user has already accessed the *Entry* in his/her local environment. To see such changes, the user must exit his/her session and restart. It is seldom necessary to do this, however, because typically the user only accesses an *Entry* owned by another user after it has been committed, and therefore cannot be changed even by the owner. See Figure 35 to see the *Refresh* icon.

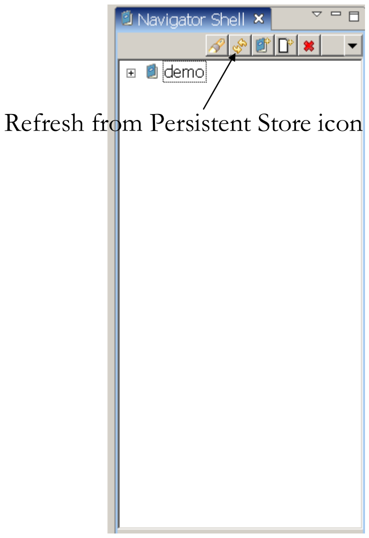


Figure 35. *Refresh from Persistent Store* icon.

3.1.2.3 Create *Journal* /Create *Entry*

The creation of a new *Journal* and the creation of a new *Entry* are covered in Sections 2.2 and 2.3 of this manual.

3.1.2.4 Delete Selection

To delete one or more *Entries*, select it (them) in the *Navigator* view, and then click on the *Delete Selection* icon (See Figure 36).

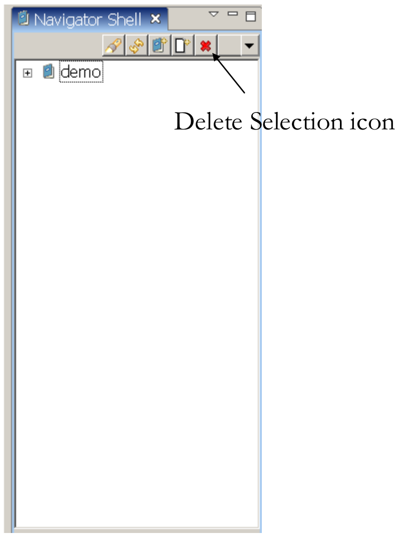


Figure 36. *Delete Selection* icon

To ensure the persistence of data in the collaborative CJ environment, delete operations are carefully regulated:

1. Only the owner of an *Entry* can delete it; AND
2. An *Entry* cannot be deleted if any other *Entries* "depend" on it.

If the *Entry* selected for deletion has dependencies AND the user owns all of the dependencies, then the user will get a warning that is similar to this:

> If you delete this entry, all of its dependencies (which are listed in the message) will also be deleted, do you want to continue?

For example, if the user deletes an experimental run, not only will the run be deleted, the experimental output(s) generated by the run will also be deleted. This is a convenience for the user, so the user does not have to work his/her way down to the tips of the dependency graph, delete these *Entries*, move back a level and repeat.

If a user tries to delete an *Entry*, and the user does not own all the dependents, then the *Entry* will not be deleted and the user will get the following error message:

> This entry cannot be deleted, since it has one or more dependents you cannot delete. The *Dependency Navigator* may be helpful to investigate these dependencies.

This protects shared data from being lost. For example, if a user uses data generated by someone else, the user does not have to worry that the data was deleted to save disk space.

A list of dependencies follows:

1. Experimental outputs depend on the experiment.
2. An Experiment depends on data source(s) it uses.
3. An Experiment depends on the workflow(s) it executes.
4. A Workflow depends on workflow/resource instances it contains
5. A Workflow depends on any data source(s) it contains
6. Every *Entry* depends on the *Journal* in which it is created(so users cannot delete a *Journal* if it contains any *Entries*).

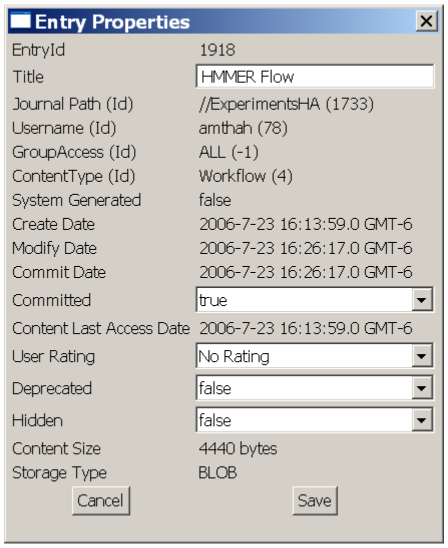
The infrastructure of the CJ environment automatically maintains these dependencies. It also supplies the *Dependency Sub-Navigator* to present these dependencies to the user.

3.1.3 Context Menu Options

Several additional actions are provided on the context menu that appears when the user clicks the ring (rightmost) mouse button after selecting one more *Entries.*

3.1.3.1 Show Entry Properties

To view the properties of an *Entry*, select the *Entry* of interest and the click on the ring button and select the *Show Entry Properties* option. A dialog box will appear presenting all the properties maintained for each *Entry* in the CJ. Figure 37 shows an example of this dialog.


Entry Properties

EntryId	1918
Title	HMMER Flow
Journal Path (Id)	//ExperimentsHA (1733)
Username (Id)	amthah (78)
GroupAccess (Id)	ALL (-1)
ContentType (Id)	Workflow (4)
System Generated	false
Create Date	2006-7-23 16:13:59.0 GMT-6
Modify Date	2006-7-23 16:26:17.0 GMT-6
Commit Date	2006-7-23 16:26:17.0 GMT-6
Committed	true
Content Last Access Date	2006-7-23 16:13:59.0 GMT-6
User Rating	No Rating
Deprecated	false
Hidden	false
Content Size	4440 bytes
Storage Type	BLOB

Figure 37. *Entry Properties* dialog box.

If the user owns this *Entry*, he/she can use this dialog to modify many of these properties. To permanently store the changes select the *Save* button at the bottom of the dialog. The *Save* button will not appear in the dialog, if the user is not allowed to modify the *Entry*.

Several field values include both a name and an id. The id is provided primarily as a debugging aid. Certain error messages may contain only an id (because the associated name could not be retrieved at the time the error is encountered). This additional information in the *Entry Properties* dialog can sometimes be useful to better understand such messages.

3.1.3.2 Delete

Deletion of selected *Entries* is covered in Section 3.1.2.4 of this manual.

3.1.3.3 Access Control

The group access attribute of an *Entry* controls which users in addition to the owner have access to the *Entry*. A *group* is a name associated with a collection of CJ users. The system administrator creates groups and can add and remove users from existing groups. A CJ user can be a member of an arbitrary number of groups.

To change the group access attribute of a selected *Entry(ies),* use the *Access Control* action. This will present a dialog containing all group names to which the user belongs as shown in Figure 38.

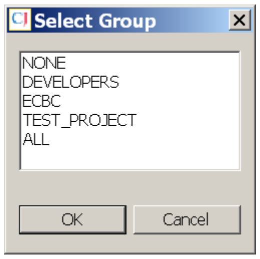


Figure 38. *Access Control* dialog box.

The *group* of choice can be selected then the *OK* button must be clicked to save the new setting.

3.1.3.4 Export

This action allows the user to create a copy of an *Entry's* content on his/her personal machine. When *Export* is selected, the user then must select the location and filename on his/her personal machine in which to place the *Entry*, then click the *Save* button.

This action generates a copy of the data stored in the CJ environment. The original data remains, unaltered in the CJ. If the user wants to share data that he/she has generated in the CJ with other colleagues who do not have access to the CJ environment, he/she can use this *export* tool.

Any changes that the user makes to the file on his/her local machine will have no effect on the data stored in the CJ. If the user wants to use the locally modified data in the CJ environment, the user must import the modified file back into the CJ environment. Importing a file always creates a new CJ *Entry* (the import action is provided through the create *New Entry* action, described in Section 2.3 of this manual).

3.2 View Options for Navigator Shell

Additional controls for the *Navigator* are provided by its pull-down menu. This menu is represented with the solid-black downward pointing triangle to the right of the toolbar, below the tab.

3.2.1 Navigator Filters

The user can select any number of filters to be applied to the *Entries* before display. An *Entry* will appear only if satisfies the test for every selected filter. In addition, if an *Entry* is not displayed because of a filter, then no *Entries* that would appear below this *Entry* in the hierarchical display will appear (even if they do satisfy all filters). So, if a *Journal* does not satisfy one of the filters, none of the *Entries* in that *Journal* will appear in the *Navigator*, until that filter is de-selected.

The filters that can be applied are:

1. Show only *Entries* in the user's *Working Set.*
   This filter applies to directly to *Journals*. The *Working Set* is a collection of *Journals*, chosen by the user, which contains the *Entries* of interest to the user for his/her current activities. The user can add/remove *Journals* from his/her *Working Set* to reflect his/her current needs. If this filter is selected, only *Journals* that are in the user's current *Working Set* will appear in the *Navigator* – and as a result, the *Navigator* will present ONLY *Entries* that appear in these *Journals.*
2. Show only *Entries* that the user has created.
   With this filter active, only *Entries* that the user has created will appear.
3. Show only committed *Entries.*
   With this filter active, only *committed Entries*, those that can no longer be modified will appear.

3.2.2 Navigator Sorters

Sorters allow the user to control the order *Entries* appear in the *Navigator.* The selected sort algorithm is applied to all *Entries* in the same hierarchical level.

The sorters available are:

1. Sort alphabetically
2. Sort by creation time
3. Sort by commit time.
   If this sorter is used, all uncommitted *Entries* will appear before any committed *Entry.*

3.3 Sub-Navigators

The area directly below the *Navigator* is reserved for a *Sub-navigator.* The *Sub-navigator* provides a specialized view of *Entries* related to the *Entry* selected in the main *Navigator* view. A variety of *Sub-navigators* are provided, each one displaying a different set of *Entries* based on a specific relationship among the *Entries.* Each of these will be described in detail in the following subsections.

The user selects the active *Sub-navigator* from the pull-down menu located by the *Navigator Shell* tab (See Figure 39). To open an *Entry* appearing in the *Sub-navigator*, simply double click on the *Entry*, as in the main *Navigator.*

Only a limited number of actions can be initiated from *Sub-navigators.* Specific *Sub-navigators* may include additional actions on their menu icon bar.

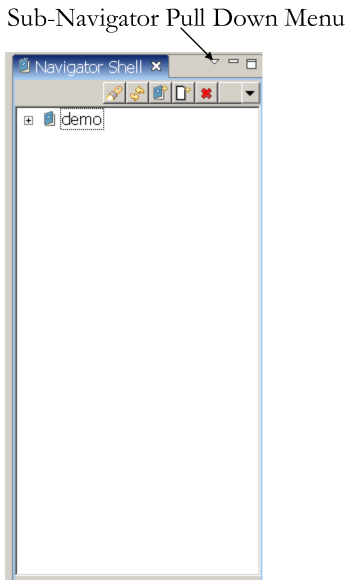


Figure 39. *Sub-Navigator* Pull Down Menu.

Another action that can be performed in the *Sub-Navigator* is *Expand/Contract View*. The *Expand/Contract Sub-Navigator* option alters the size of the view. Changing the size of the *Sub-Navigator* affects the size of the *Navigator Shell* view. The user can adjust the size to suit his/her preference. The icon for this action can be seen in Figure 40.

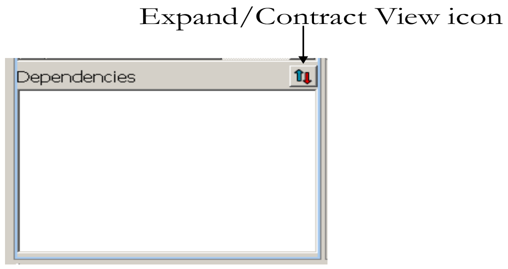


Figure 40. Expand/Contract View icon

3.3.1 Dependent Manager

This view allows the user to see all the *Entries* that depend on the selected *Entry* in the *Navigator Shell*. A list of dependencies follows:

1. Experimental outputs depend on the experiment.
2. An Experiment depends on data source(s) it uses.
3. An Experiment depends on the workflow(s) it executes.
4. A Workflow depends on workflow/resource instances it contains
5. A Workflow depends on any data source(s) it contains
6. Every *Entry* depends on the *Journal* in which it is created (so users cannot delete a *Journal* if it contains any *Entries*).

In this view, the first level of this hierarchy includes all *Entries* that directly depend on the selected *Entry*. Beneath each of these *Entries* will appear the *Entries* that directly depend on the corresponding *Entry* (and so indirectly depend on the *Entry* selected in the main *Navigator*), and so on. Figure 41 shows an example of what can be viewed in this *Sub-navigator*.

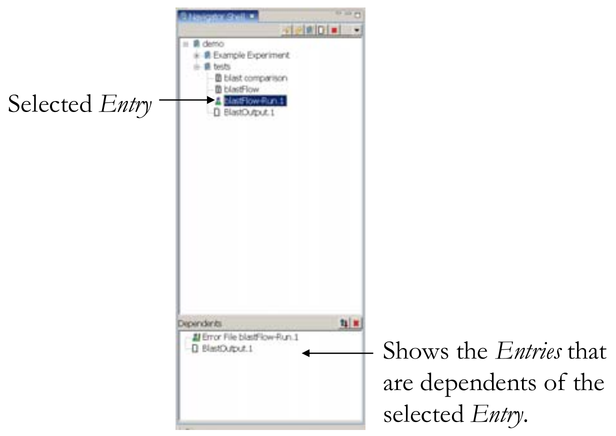


Figure 41. Sample view in the *Dependent Manager*.

This *Sub-navigator* is useful to determine how widely a particular piece of data or workflow definition is used in experiments by any CJ users. If the user is considering deleting an *Entry*, he/she may want to use the *Sub-navigator* to first investigate if it is used elsewhere.

Entries in this view can be deleted. The *Delete Selection* option is describe in Section 3.1.2.4 of this manual.

3.3.2 Dependency Manager

This view allows the user to view the *Entries* that the *Entry* selected in the main *Navigator* depends on. This view is effectively the inverse of the preceding *Dependent Manager*. This *Sub-navigator* is particularly useful to observe the provenance of a particular piece of data.

3.3.3 Annotations

This view allows the user to view the *Annotations* of the *Entries* present in the *Navigator Shell* view. An *Annotation* is an additional piece of information relevant to the associated *Entry*. Users are free to attach whatever annotation(s) they find useful to another *Entry*. A few examples of uses for *annotation* include:

1. Frequently, the significance of a data set can be concisely described by the owner in a brief textual description.
2. A graph or an image of a dataset (perhaps generated by an external tool).
3. A reference (perhaps a URL) on which a workflow or experiment is based.

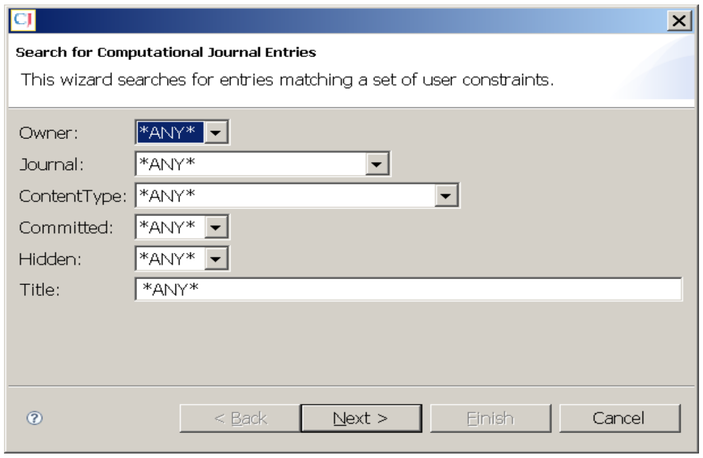
In addition to the normal operations for viewing the *Entries* in this view, this *Sub-navigator* provides actions to create new annotations for the *Entry* selected in the main *Navigator*.

3.3.3.1 Creating a new annotation

This action implicitly invokes the action to create a new *Entry*, which is fully described in Section 2.3. The newly created *Entry* is automatically linked to the *Entry* selected in the main *Navigator* as an annotation.

3.3.3.2 Annotation with an existing *Entry*

Annotating a selected *Entry* with an existing *Entry* uses a search functionality to find the existing *Entry*. When this option is selected, a dialog box will appear that allows the user to set specific search criteria (See Figure 42).

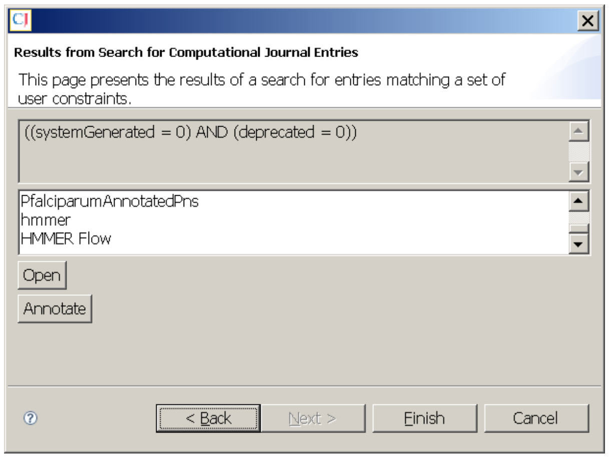

Search for Computational Journal Entries

This wizard searches for entries matching a set of user constraints.

Owner:	*ANY*
Journal:	*ANY*
ContentType:	*ANY*
Committed:	*ANY*
Hidden:	*ANY*
Title:	*ANY*

Figure 42. *Search for Computational Journal Entries* dialog box.

Once the criteria has been set, the user clicks the *Next* button and the search results are presented to the user in another dialog box (See figure 43). The user selects the appropriate *Entry* then clicks the *Annotate* button.


Results from Search for Computational Journal Entries

This page presents the results of a search for entries matching a set of user constraints.

((systemGenerated = 0) AND (deprecated = 0))

PfalciparumAnnotatedPns
hmmer
HMMER Flow

Figure 43. *Results from Search for Computational Journal Entries* dialog box.

4 Workflow Editor

The Workflow Editor is used for creating/editing and viewing Workflows. There are two general categories in which the creation of a Workflow falls:

1. A tool/utility for reuse, such as a step that can be used in a variety of experiments.
2. Defining an experiment or a specific component of an experiment.

Depending on the user's intent for the workflow, different construction styles will be more applicable than others. For example, if the user is more interested in reuse, the user will want to promote many property values of resources. Also the use of input ports allows for a more general use for the workflow since it is not set to a specific input. On the other hand, if the user wants to create a very specific experiment, the user will want to set many of the property values of resources and promote very few of them. Also, the use of *DataSources* may offer more control of input to the resources.

4.1 Palette Options

Most Workflow Editor actions are invoked from its palette. The palette options are:

1. Select
2. Marquee
3. DataSource
4. Resource
5. Workflow
6. Connection
7. Input
8. Output

Most palette options are persistent, that is they remain in effect until another option is selected.

4.1.1 Selection

The user activates the *Select* option to manipulate, alter property values, or access any input ports, output ports or blocks in the Workflow.

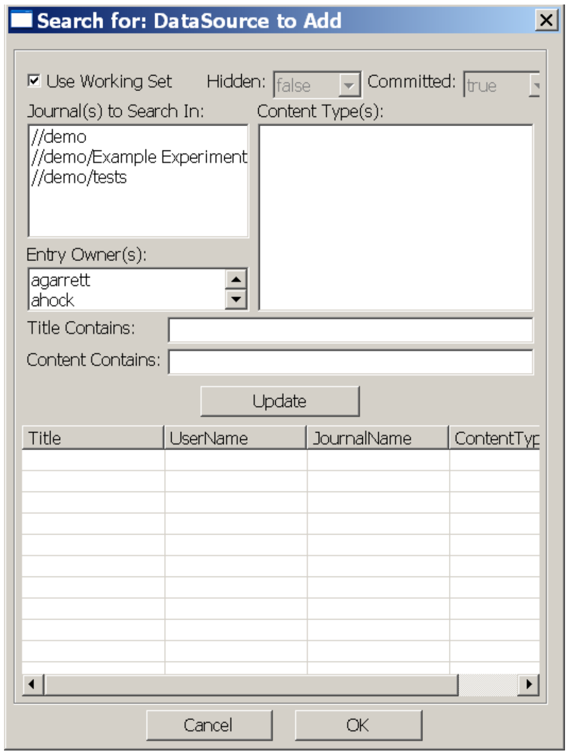
The *Marquee* option allows the user to select several blocks/objects and operate on them as a group. To use this tool, the user has to click on the mouse and drag the *Marquee* box around the blocks they wish to group. The group actions are limited to *Delete* and moving the blocks.

4.1.2 Adding Components

The options, *DataSource*, *Resource*, and *Workflow* are very similar. Each of these options allows the user to add a specific class of block to the diagram.

1. DataSource
   A block that provides the contents of an arbitrary entry on its output port. For example, if you want to provide a specific genomic sequence as an input to another block, you would add a *DataSource*, and select the entry containing the desired sequence from the query dialog.
2. Resource
   A block representing a core computational program.
3. Workflow
   A block representing a previously defined workflow. It will appear as a single block displaying only the inputs and outputs of the corresponding workflow definition (not each of that workflow's internal blocks). This allows the user to define increasingly complex workflows in a hierarchical fashion. It also promotes the re-use of common experiment steps.

Each of these options employs a similar interface. To add a *DataSource*, the user clicks on the *DataSource* option in the *Palette*. The user then has to move the cursor to the *Editor Area* to view the *Search for: Datasource to Add* dialogue box (See Figure 44). The user then must locate the *DataSource* he/she wishes to add then click the *OK* button.


Search for: DataSource to Add

☑ Use Working Set   Hidden: false   Committed: true

Journal(s) to Search In:
//demo
//demo/Example Experiment
//demo/tests

Content Type(s):

Entry Owner(s):
agarrett
ahock

Title Contains:

Content Contains:

Title	UserName	JournalName	ContentTyp

Figure 44. *Search for: DataSource to Add* dialogue box.

Additional details for adding a Resource and Workflow to a *Workflow* are described in Section 2.4.1 of this manual.

4.1.3 Connections

The Connections option allows the user to connect the data blocks (*DataSources*, *Resources* or *Workflows*), *Input* ports, and *Output* ports together, to define the data flow through the *Workflow*. Every solid colored port in a *Workflow* definition must be connected to another port in order for the *Workflow* definition to be complete. Shaded ports indicate optional ports. You may make connections to/from these ports if you want to supply/use the data associated with the port.

After selecting this option, the user moves the mouse to the source port for a connection and clicks the left button. Next the user selects the destination port with another left button click. If the connection is valid, a line between the source and destination appears. The system remains in connection mode, waiting for the user to select another connection source, or to select a different palette option.

The rules for valid connections are:

1. Connections must be made from source to destination; this means the source port must be the initial selection.
2. Connections are only allowed from block outputs to block inputs (and workflow input ports to block inputs, and block outputs to workflow port outputs)
3. Only one connection can be made to a block input (or workflow output port)
4. Multiple connections may be made from a block output (or workflow input port)
5. The source and destination port types must be compatible. For example, you cannot connect a FASTA format to a port that requires a BLAST_Report. Port type information is provided in a help dialog that appears whenever the mouse hovers over a port.

4.1.4 Input and Output Ports

Input and output ports in a *Worfklow* definition define the external interface for this component. Input port(s) signify data that must be supplied in order for the workflow to execute, and output port(s) represent data that will be accessible after it has executed. The ports must be connected to block inputs (outputs), using the connection option described in the previous section, in order to complete the workflow definition.

Typically, output ports are defined for the final block outputs. But, they can also be used to make data that is transferred between two blocks in the workflow visible externally. To do so, simply define another output port, and connect the input source for the internal datapath to this port. Then, the data will used as input for the block to which it is connected, and preserved as an external output generated by the workflow.

The names used for input and output ports must be distinct in the workflow definition. Default unique names are automatically generated, but it is highly recommended that the user change the port names to more meaningful names once they are added to a Workflow. This is done in the *Properties* view. The type of input and output ports is automatically derived from the type of the block port to which it is eventually connected. Examples of adding both *Input*s and *Output*s are provided in Section 2.4.2 of this manual.

4.2 Non-palette Actions

This section describes the remaining Workflow Editor actions and how to invoke them.

4.2.1 Argument Value Assignment

Each *Resource/Workflow* has zero or more arguments (parameters) that control or modify its execution. Once a *Resource/Workflow* has been added to a *Workflow* definition, the user has the option to set these argument values. This action is performed in the *Properties View* that typically appears as one of the tabbed views directly below the editor area.

The user must select the block in the *Workflow* whose values he/she wishes to adjust. The arguments of that *Resource* and the current values to them will appear in the *Properties View*.

For each argument the user can choose to:

1. Allow the program to choose a default value for the argument.
   To do this, the argument value should be left empty. For some arguments, the user must provide a value, however, and the workflow definition will not be complete until the user makes a value assignment.
2. Set the argument to a specific value.
   To do this, simply enter or select the desired value. A validation function is performed on argument values. If an invalid value is entered, it will not be accepted and a brief error message will appear in the status line at very bottom of the CJ window. When an argument value is set to a specific value, every experiment that is defined using that Workflow will use this assigned value.
3. Promote the argument to become an argument for the Workflow being defined.
   To promote an argument, the user checks the box located to the left of the argument name. This action implicitly creates a new argument (with this argument's name) for the Workflow being defined. The value used for this argument for the selected block is deferred to the user for the workflow being defined. Promoting an argument allows the user to decide what the value of that argument should be for each experiment that is defined using this Workflow. This allows the Workflow to be more flexible in defining experiments. This option prevents the necessity of having several copies of virtually the same Workflows that only vary in the values that are given to the properties of the Resources in the Workflow.
   If a value is entered in the value field of a promoted argument, this value will be used as the default value for the promoted argument.

4.2.2 Context Menu Actions

When you click the right mouse button inside the Workflow Editor, a context menu appears with actions:

1. Undo – removes the effects of the most recent editor action. You may invoke this action repeatedly to step back through multiple editor actions.
2. Redo – repeats the most recent undo action. If you have performed undo multiple times, you may use redo to repeat each of these actions in order.
3. Delete – removes the currently selected object(s) from the editor.

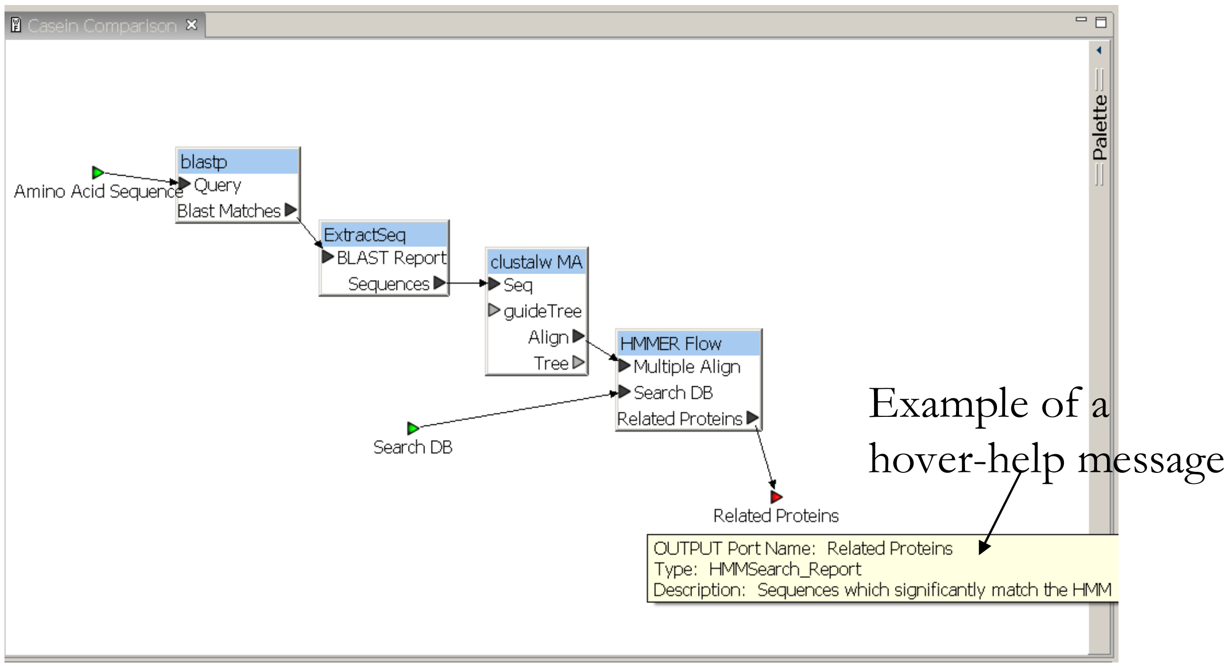
4.2.3 Save Actions

The main File menu provides several actions to allow you to save your work.

1. Save – saves the current contents of the active editor in the CJ environment, and allows you to continue editing the workflow.
2. Commit –saves the current contents of the active editor in the CJ environment AND marks the associated entry as *committed*. Once committed, the workflow can be used to define an experiment, or in the construction of more complex workflows. However, it can no longer be modified.
   Since the workflow cannot be modified after this action, the workflow editor first performs a series of tests to verify that the definition is both complete and consistent. For example, it ensures that no required block port has been left unconnected; and that every argument value assignment is valid. If any problems are detected, a dialog box appears explaining the problem(s), and the commit action is canceled.
   After the commit action completes, the active edit session is implicitly closed, and a browser is opened on the same object. A browser is an editor with limited capabilities; it does not permit you to modify or save the object.
3. Save As – creates a new workflow entry with the name you choose that is initialized with a copy of the contents of the active editor window. The active editor is implicitly closed (without saving any changes), and a new editor window is opened on the newly created workflow.
4. Close – closes the currently active editor window. If the object has been modified you will be asked if you want to save the contents before closing.

4.2.4 Help

To assist the user in the CJ environment, there is hover help. The user places the cursor over objects in the Workflow Editor, and a brief help message about the object appears on the screen. Figure 45 shows an example of a hover-help message.



OUTPUT Port Name: Related Proteins
Type: HMMSearch_Report
Description: Sequences which significantly match the HMM

Figure 45. Example of a hover-help message.

5 Launching an Experiment

Once a *Workflow* has been committed, experiments can be defined and executed using that *Workflow*. The *Workflow* of interest needs to be active in the Workflow Editor. In this section, a *Workflow* (*MultipleAlignExample*) that uses three *Resources*, *Simple Blastall*, *ExtractSeq* and *clustalw MA*, will be used to describe the process of launching an experiment. In this *Workflow*, the *Simple Blastall Resource* has its parameters promoted. The promotion of these parameters means that the user will have to decide the values for these parameters during the process of defining an experiment based on this *Workflow*.

5.1 Starting the definition process

To start the definition process, the user needs to first go to the *Experiment* Menu and select the *Define Experiment From Workflow* option (See Figure 46)

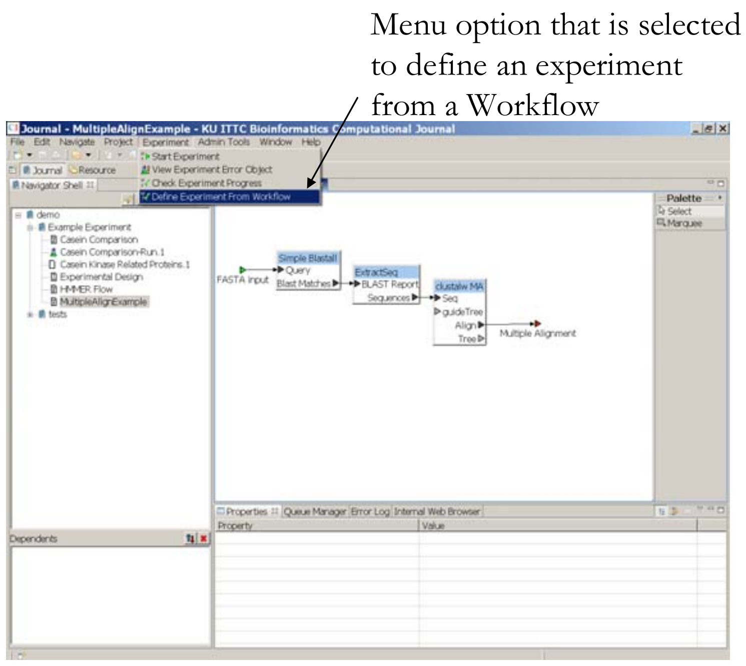
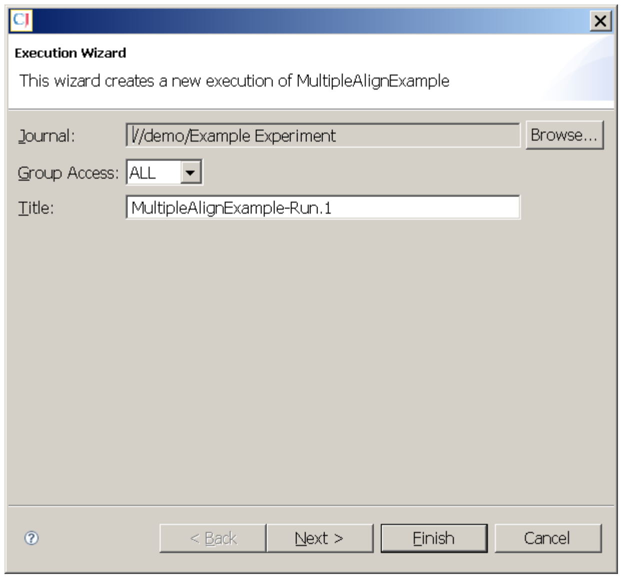

Menu option that is selected to define an experiment from a Workflow
Journal - MultipleAlignExample - KU ITTC Bioinformatics Computational Journal
File Edit Navigate Project Experiment Admin Tools Window Help
Start Experiment
View Experiment Error Object
Check Experiment Progress
Define Experiment From Workflow
Journal Resource
Navigator Shell
demo
Example Experiment
Casein Comparison
Casein Comparison-Run.1
Casein Kinase Related Proteins.1
Experimental Design
HMMER Flow
MultipleAlignExample
tests
FASTA input
Simple Blastall
Query
Blast Matches
ExtractSeq
BLAST Report
Sequences
clustalw MA
Seq
guideTree
Align
Tree
Multiple Alignment
Palette
Select
Marquee
Properties Queue Manager Error Log Internal Web Browser
Property
Value
Dependents

Figure 46. *Experiment* Menu option to define an experiment from an active *Workflow*.

When this menu option is selected, a dialog box will appear (Figure 47). The user is asked to specify a name for the entry to contain this experiment definition; and to specify the *Journal* and the *Group Access* for the experiment and the results it will generate.



Execution Wizard

This wizard creates a new execution of MultipleAlignExample

Journal: \\/demo/Example Experiment [Browse...]

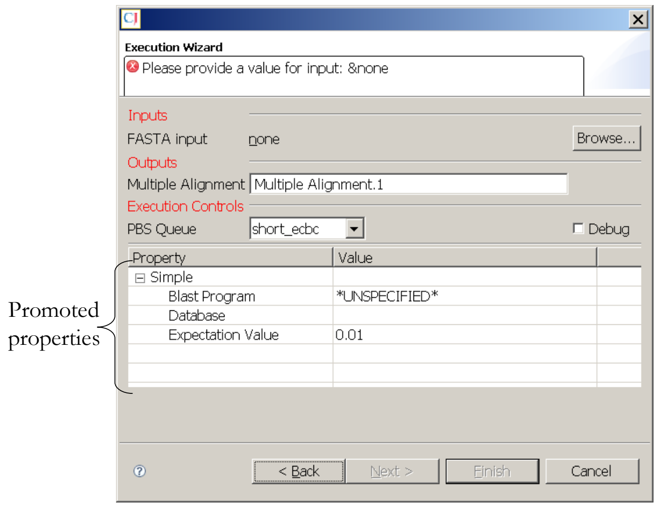
Group Access: ALL

Title: MultipleAlignExample-Run.1

< Back | Next > | Finish | Cancel

Figure 47. *Execution Wizard* dialog box.

To define the experiment's parameters, the user clicks the *Next* button. On the following page, the user specifies the needed input(s) for the experiment and the values for the promoted parameters. A message at the top of the dialog box will inform the user what he/she is required to enter (See Figure 48).



Execution Wizard

Please provide a value for input: &none

Inputs

FASTA input none Browse...

Outputs

Multiple Alignment Multiple Alignment.1

Execution Controls

PBS Queue short_ecbc Debug

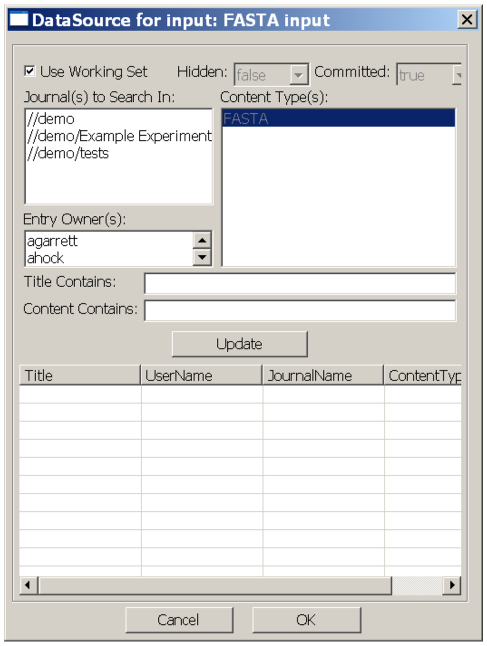
Property	Value
Simple	
Blast Program	*UNSPECIFIED*
Database	
Expectation Value	0.01

< Back Next > Finish Cancel

Figure 48. *Execution Wizard* dialog box.

5.2 Adding Input

To specify an input for the experiment, click the *Browse…* button associated with the input port. A search dialog box will appear (See Figure 49).



DataSource for input: FASTA input

Use Working Set Hidden: false Committed: true

Journal(s) to Search In:
//demo
//demo/Example Experiment
//demo/tests

Content Type(s):
FASTA

Entry Owner(s):
agarrett
ahock

Title Contains:

Content Contains:

Update

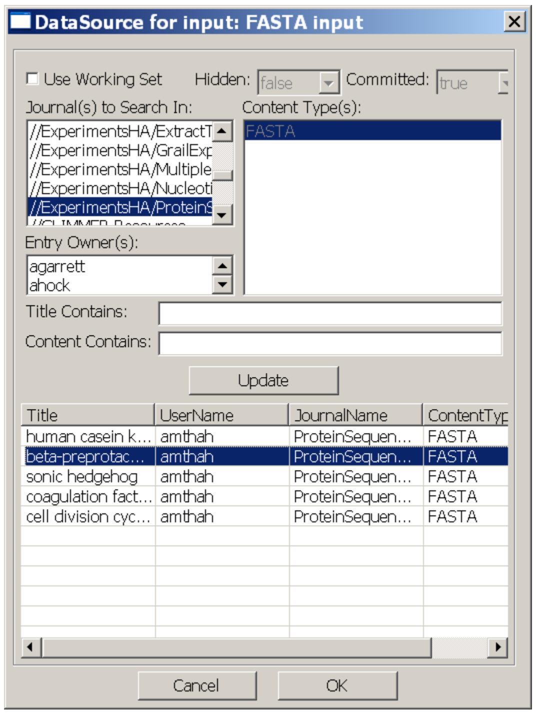
Title	UserName	JournalName	ContentTyp

Cancel OK

Figure 49. Search dialog box that appears when looking for input.

Notice that the search dialog box automatically limits the *Content Type* of the search to the type that is appropriate for the specified input port. In this example, FASTA formatted data is required. Thus, the user can only select *Entries* of this *Content Type* to add as input for the experiment. If the user can see an *Entry* he/she wants in the *Navigator Shell*, but not in the search dialog box, he/she may want to use the *Show Entry Properties* ring-button option in the *Navigator Shell* to compare the desired *Entry's* properties to what is required.

To help find the desired input, the user can alter the search criteria to restrict or expand the search space in the CJ environment. Also, only committed *Entries* can be selected. For this example, an amino acid FASTA entry called *beta-preprotachykinin* will be used (See Figure 50).



Title	UserName	JournalName	ContentTyp
human casein k...	amthah	ProteinSequen...	FASTA
beta-preprotac...	amthah	ProteinSequen...	FASTA
sonic hedgehog	amthah	ProteinSequen...	FASTA
coagulation fact...	amthah	ProteinSequen...	FASTA
cell division cyc...	amthah	ProteinSequen...	FASTA

Figure 50. Completed dialog box for an experiment input.

5.3 Execution Controls

5.3.1 PBS Queue

The user needs to specify the PBS (Portable Batch queue System) Queue in which to execute the experiment. All jobs executed using the cluster computing environment are controlled by a queuing system, PBS, to facilitate sharing of the resources. Many different queues are defined on the system that control priority of the jobs submitted to them and the range of resources available to those jobs.

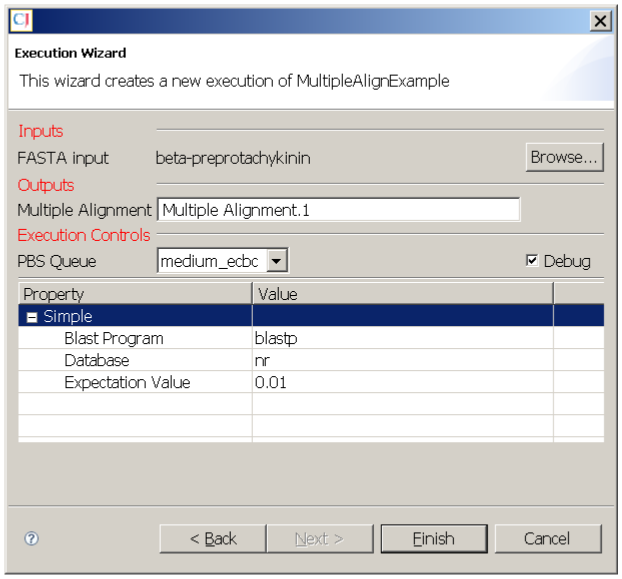
Before attempting to run experiments, the project supervisor and/or the system administrator needs to be consulted about which queues can be used and the restrictions they impose on jobs submitted. It is the user's responsibility to choose an appropriate queue (short, medium, long or max) for the defined experiment. If inadequate resources are available in the selected queue (for example, maximum execution duration), the experiment will not be able to complete successfully. In this example, the *medium_ecbc* queue will be used.

5.3.2 Debug

If the *Debug* option is selected, additional information will be saved during the experiment execution. An *Error Object* will be generated when the experiment runs, even if the experiment completes successfully. The additional information collected in the *Error Object* can help a user understand why an experiment does not generate the expected results even though it completes successfully. The *Debug* option allows the user to trace back through intermediate results to help the user discover where the results began to differ from what was expected. In this example, *Debug* will be selected. Additional details about the content of an *Error Object* are provided in Section 6 of this manual.

5.4 Assigning Values to Promoted Parameters

Since the *Simple Blastall Resource* has three promoted parameters, the user needs to provide values for these parameters. Since an amino acid sequence was selected for input, a BLAST program that can accept that format should be selected. In this example, *blastp* will be selected. The default database, `nr`, will be entered, and for the expectation value we will use the default value in the dialog, `.01`. Figure 51 shows the completed dialog box.


Execution Wizard

This wizard creates a new execution of MultipleAlignExample

Inputs

FASTA input: beta-preprotachykinin

Outputs

Multiple Alignment: Multiple Alignment.1

Execution Controls

PBS Queue: medium_ecbc

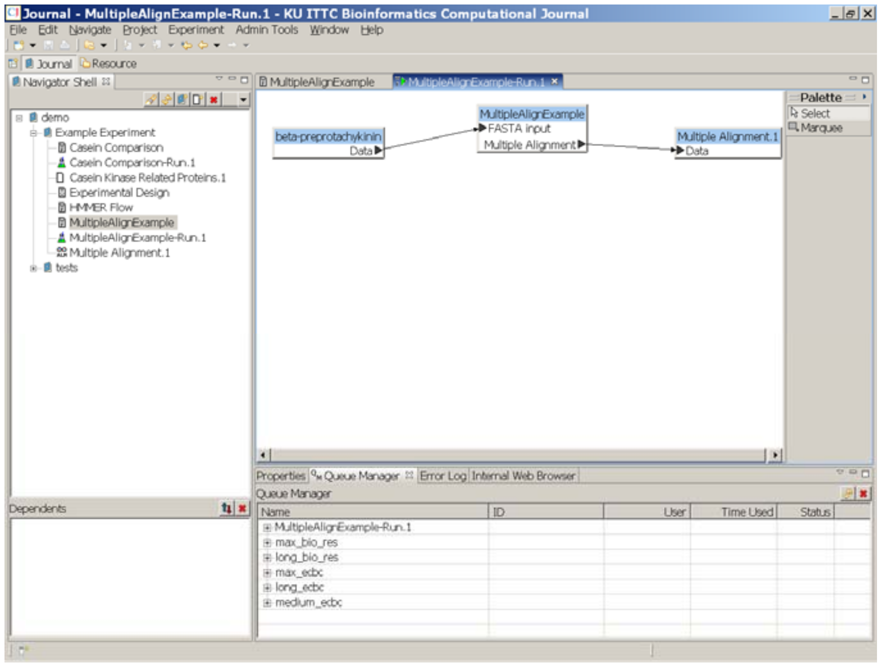
☑ Debug

Property	Value
⊟ Simple	
Blast Program	blastp
Database	nr
Expectation Value	0.01

Figure 51. Completed *Execution Wizard* dialog box.

5.5 Starting the Experiment

To start the experiment, the user clicks the *Finish* button. After the *Finish* button is clicked, the CJ will create an instance of the experiment, *MultipleAlignExample-Run.1*. We will see a graphical representation of the experiment in the *Editor Area* (See Figure 52).



Name	ID	User	Time Used	Status
MultipleAlignExample-Run.1				
max_bio_res				
long_bio_res				
max_ecbc				
long_ecbc				
medium_ecbc				

Figure 52. Experiment in the Editor Area.

5.6 Tracking Experiment Progress

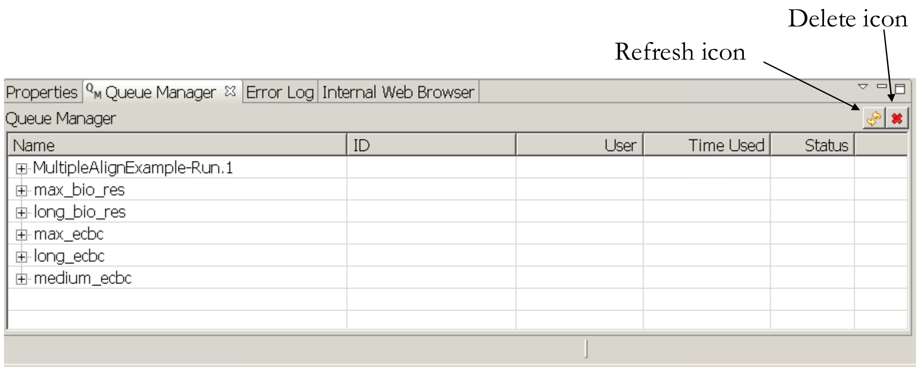
It is useful to track the status of an experiment since bioinformatics applications are often computationally intensive and have long execution time. Once an experiment has been started the user can track the progress of the experiment using the *Check Experiment Progress* action or the *Queue Manager View*.

5.6.1 Check Experiment Progress

The action *Check Experiment Progress* on the *Experiment* menu opens a simple dialog presenting the current state of the active experiment definition in the *Editor Area*. This action simply reports that the experiment is still running, has completed successfully, or completed with errors. If the experiment completes successfully, the *Entry(ies)* associated with the experiment's output ports will contain the experiment results. If the experiment completes with errors, the user will want to review the *Error Object* to determine the cause of the errors. The *Error Object* is described more fully in Section 6 of this manual.

5.6.2 Queue Manager

The *Queue Manager* view, which is a tabbed view that typically appears directly below the *Editor Area*, provides a more detailed view of an experiment's progress. This view provides a summary of the current status of all jobs that have been submitted to execute experiment(s) on your behalf, as well as all other active jobs on the cluster. Figure 53 provides a sample view from the *Queue Manager*.



Queue Manager

Name	ID	User	Time Used	Status
⊞ MultipleAlignExample-Run.1				
⊞ max_bio_res				
⊞ long_bio_res				
⊞ max_ecbc				
⊞ long_ecbc				
⊞ medium_ecbc				

Figure 53. *Queue Manager* View

Typically, several jobs are generated for the execution of a single experiment. At the top of the *Queue Manager* view are the experiment(s) the user has submitted that are still running, or have recently completed. At the next level of the hierarchy is the collection of jobs that are currently active for this experiment. The names of these jobs are automatically generated, but include the name of the resource they are executing. For each job, additional information such as the job status and amount of execution time is provided.

In this view, the active queues (those with jobs currently running) on the cluster are also presented below the user's active experiments. This information can be helpful when the user is choosing which queue to submit a new experiment.

The *Queue Manager* view is not automatically updated. The user must explicitly request this view to update its presentation of the queue status by selecting the Refresh icon at the top, right edge of the view. After an experiment completes, the *Queue Manager* provides a special status line for this experiment:

```
Recently finished <experiment title>
```

This message appears only once. The next time that you refresh the *Queue Manager* view there will be no mention of this completed experiment.

The *Queue Manager* view recalls the experiments that the user has started in previous CJ session(s), so if the user has a long running experiment, he/she can continue to track its progress with the *Queue Manager* view, if he/she exits and then restarts his/her CJ session.

The *Queue Manager* view provides no information about the status of a completed job, that is, if it completed successfully or with errors. For this information the user needs to use the menu action, *Check Experiment Progress*.

5.7 Stopping Experiment Execution

The *Queue Manager*, described in the previous section, also allows the user to halt the execution of an experiment, in case the user determines that there was a problem in the experiment definition, or if the experiment is executing for much longer than anticipated.

To stop a running experiment, select the experiment in the *Queue Manager*, and then click on the *Delete* menu icon in the upper right corner of the *Queue Manager View*.

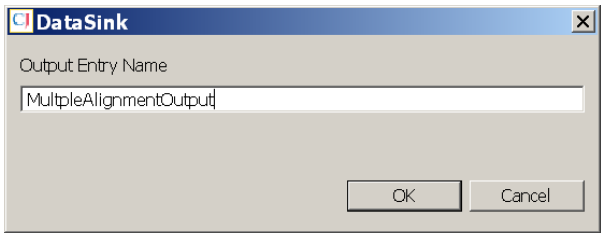
5.8 Viewing Results

A result/output *Entry* is created as soon as the experiment begins, but the contents will be empty until the experiment has completed. Once the experiment has completed, the results of the experiment can be viewed by double-clicking the output block in the experiment or by selecting the output *Entry* in the *Navigator Shell*.

5.9 Editing Experiment Definitions

Once an experiment has begun execution it cannot be modified. It remains as a permanent record of the experiment. However, if you would like to run a similar experiment changing one or more inputs and/or parameter values, you can use the *Save As* action on the *File* menu.

The *DataSink*s (output entries associated with the experiment output ports) are not copied. You must define new entries for the outputs of this new experiment. To add a *DataSink*, the user selects the *DataSink Palette* option. A dialog appears requesting the user to provide a name for the new entry that will be created. In this example, the output will be called *MultipleAlignmentOutput* (See Figure 54). The user then needs to click in the *Editor Area* to place the sink into the experiment. Finally, the user needs to connect the *DataSink* block with the appropriate output port.



MultpleAlignmentOutput

Figure 54. Shows the *DataSink* dialog box

In addition the user can change the *Entry* used as an input to the experiment by first removing the *DataSource* in the diagram to be replaced, and then selecting the *DataSource Palette* option to create a new *DataSource*. This will present a search dialog from which the user selects the *Entry* he/she wants to use as the input source. This search dialog places no constraint on the content type, since the port to which you will attach the *DataSource* has not been identified.

After placing the new block in the diagram, the user must connect it to the appropriate input port. Note that if the *Content Type* of the *Entry* the user selected as the *DataSource* is not compatible with the input port, the connection will be refused.

To change one or more system parameters, select the *Workflow* block in the diagram and use the *Properties View* to check and/or change the experiment parameter values. Chapter 4, that describes the Workflow Editor, provides details about the usage of the *Property View*.

The user can then run the copy of the experiment by selecting the *Start Experiment* option in the *Experiment* menu. Figure 55 shows the copy of the experiment and that it is running (Queue Manager).

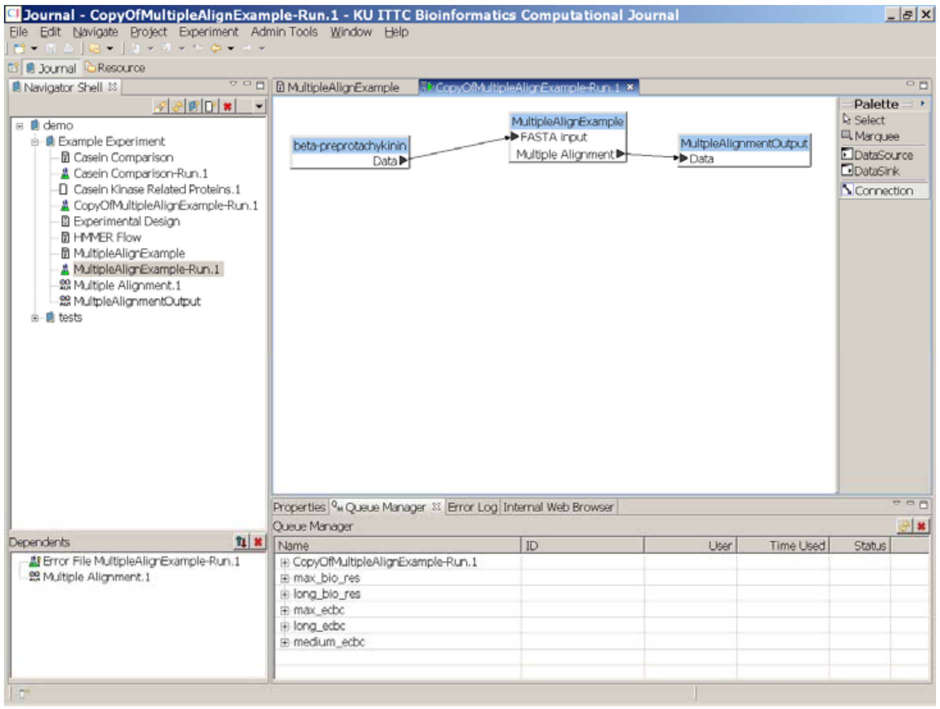
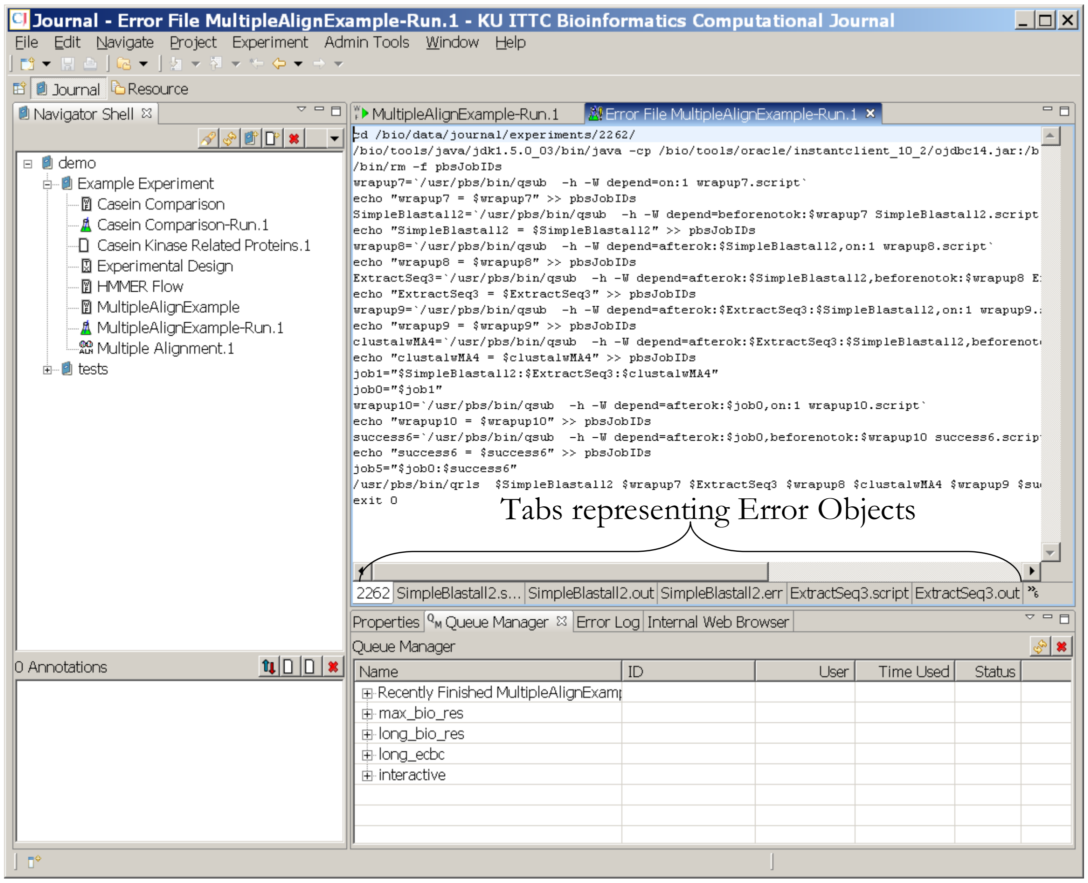


Figure 55. *CopyOfMultipleAlignExample-Run.1*

6 Error Object

If an experiment fails to complete successfully an *Error Object* is generated that contains additional details about the experimental process. An *Error Object* is always generated for experiments that are executed with the *Debug* option set.

To view the *Error Object* associated with an experiment, select the *View Experiment Error Object* action on the Experiment menu, when the experiment definition is active in the editor area. The viewer for an *Error Object* uses a multi-object editor paradigm. At the bottom of the window is the list of objects that can be presented. The viewing area contains the data associated with the selected tab. To view another object, the user simply selects the appropriate tab. Figure 56 shows an example of the view of an *Error Object*.



```
cd /bio/data/journal/experiments/2262/
/bio/tools/java/jdk1.5.0_03/bin/java -cp /bio/tools/oracle/instantclient_10_2/ojdbc14.jar:/b
/bin/rm -f pbsJobIDs
wrapup7=`/usr/pbs/bin/qsub  -h -W depend=on:1 wrapup7.script`
echo "wrapup7 = $wrapup7" >> pbsJobIDs
SimpleBlastall2=`/usr/pbs/bin/qsub  -h -W depend=beforenotok:$wrapup7 SimpleBlastall2.script
echo "SimpleBlastall2 = $SimpleBlastall2" >> pbsJobIDs
wrapup8=`/usr/pbs/bin/qsub  -h -W depend=afterok:$SimpleBlastall2,on:1 wrapup8.script`
echo "wrapup8 = $wrapup8" >> pbsJobIDs
ExtractSeq3=`/usr/pbs/bin/qsub  -h -W depend=afterok:$SimpleBlastall2,beforenotok:$wrapup8 E
echo "ExtractSeq3 = $ExtractSeq3" >> pbsJobIDs
wrapup9=`/usr/pbs/bin/qsub  -h -W depend=afterok:$ExtractSeq3:$SimpleBlastall2,on:1 wrapup9.
echo "wrapup9 = $wrapup9" >> pbsJobIDs
clustalwMA4=`/usr/pbs/bin/qsub  -h -W depend=afterok:$ExtractSeq3:$SimpleBlastall2,beforenot
echo "clustalwMA4 = $clustalwMA4" >> pbsJobIDs
job1="$SimpleBlastall2:$ExtractSeq3:$clustalwMA4"
job0="$job1"
wrapup10=`/usr/pbs/bin/qsub  -h -W depend=afterok:$job0,on:1 wrapup10.script`
echo "wrapup10 = $wrapup10" >> pbsJobIDs
success6=`/usr/pbs/bin/qsub  -h -W depend=afterok:$job0,beforenotok:$wrapup10 success6.scrip
echo "success6 = $success6" >> pbsJobIDs
job5="$job0:$success6"
/usr/pbs/bin/qrls  $SimpleBlastall2 $wrapup7 $ExtractSeq3 $wrapup8 $clustalwMA4 $wrapup9 $su
exit 0
```

Figure 56. Error Object and the tabs for the other objects

To better understand the meaning of the objects contained in an *Error Object*, a brief overview of the process to execute an experiment defined in the CJ will be provided. An experiment definition typically describes the execution of several core programs on the computer cluster in a specific order. To execute an experiment, a distinct job is defined to control the execution of each core program (*Resource* in a CJ *Workflow* definition). If the output(s) of the *Resource* are not connected to a *DataSink* in the experiment definition, a temporary file will be allocated for the output. The name for each of these files is automatically generated, beginning with `tmp`. These temporary files are used to transfer data internally between connected *Resources*.

A schedule is created for the set of jobs defined to execute the experiment so that a job begins only after all jobs that are responsible for generating its input source(s) have completed – without failing. If one or more of a job's dependents fails, then this job is cancelled.

For each job that does execute, three files appear in the *Error Object*:

1. `<name>.script` that contains the command line used to start this program.

2. `<name>.out` that contains information output to `stdout` by this program. The content of this file is completely dependent on the program that is run. It may contain information about invalid inputs and/or warning messages.
3. `<name>.err` that contains information output to `stderr` by this program. The content of this file is completely dependent on the program that is run. It typically contains error information that is likely to explain why a program failed to run correctly.

In addition, a portion of each of the internal files that contain data transferred between internal blocks in the diagram will appear in the *Error Object*. The contents of these files are presented only with a simple text editor. So if the output contains data in binary format, the *Error Object* will not provide capabilities to view it properly. Observing the information that flows along internal connections will often help explain why an experiment failed, or at least did not generate the expected results. The content of *DataSinks* do not appear in the *Error Object* since these entries can be viewed directly in the CJ environment.

7 Incorporating External Viewers

For some *Content Types* (data formats) sophisticated viewing programs already exist that may not be easy to directly incorporate into the CJ environment. The CJ environment allows users to integrate these existing tools into his/her usage of the CJ through the definition of a *Local Service*. This mechanism allows the user to associate a program running on his/her client machine for viewing a specific *Content Type*.

When defining a new *Local Service*, the user identifies:

1. The exact location of the program on his/her machine;
2. The command line argument(s) that must precede the input filename when this program is invoked; and
3. The content type for which this program should be used as the viewer.

Each user must define the *Local Services* installed on his/her machine that he/she intends to use from the CJ. This is not a system-wide setting because different users will choose to include different programs; and these programs can be installed at arbitrary locations. It is the user's responsibility to properly install these external programs on his/her machine.

This action is invoked from the *File* Menu, *New* option, and then *Local Service* (See Figure 57).

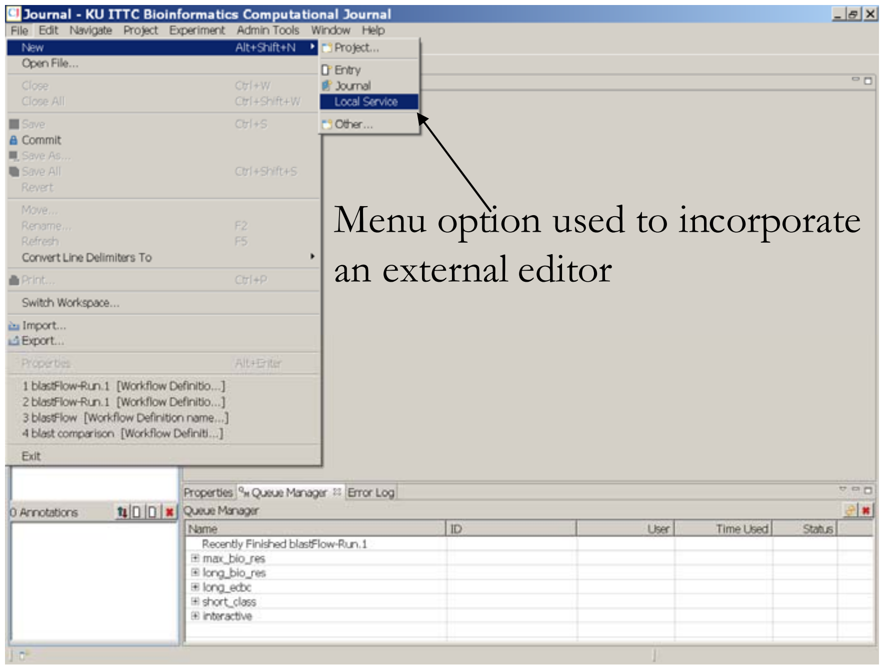
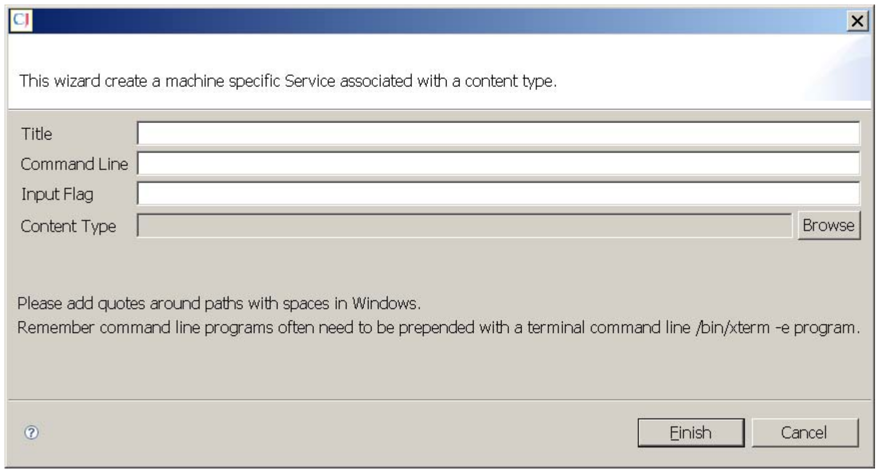

Journal - KU ITTC Bioinformatics Computational Journal
File Edit Navigate Project Experiment Admin Tools Window Help
New Alt+Shift+N
Open File...
Close Ctrl+W
Close All Ctrl+Shift+W
Save Ctrl+S
Commit
Save As...
Save All Ctrl+Shift+S
Revert
Move...
Rename... F2
Refresh F5
Convert Line Delimiters To
Print... Ctrl+P
Switch Workspace...
Import...
Export...
Properties Alt+Enter
1 blastFlow-Run.1 [Workflow Definitio...]
2 blastFlow-Run.1 [Workflow Definitio...]
3 blastFlow [Workflow Definition name...]
4 blast comparison [Workflow Definiti...]
Exit
Project...
Entry
Journal
Local Service
Other...
Menu option used to incorporate an external editor
Properties Queue Manager Error Log
0 Annotations
Queue Manager
Name ID User Time Used Status
Recently Finished blastFlow-Run.1
max_bio_res
long_bio_res
long_ecbc
short_class
interactive

Figure 57. Menu option to add an external viewer to the CJ environment.

A dialog box appears that allows the user to incorporate an external viewer to the CJ environment (See Figure 58).



This wizard create a machine specific Service associated with a content type.

Title	
Command Line	
Input Flag	
Content Type	

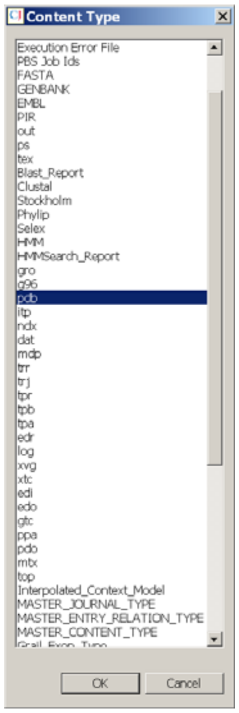
Please add quotes around paths with spaces in Windows.
Remember command line programs often need to be prepended with a terminal command line /bin/xterm -e program.

Figure 58. *Wizard* to define a Local Service.

The following steps demonstrate how to define the Visual Molecular Dynamics (VMD) program (http://www.ks.uiuc.edu/Research/vmd/) as the default viewer for `pdb` files.

1. Choose a title for the service. In this example, the title will be VMD-pdb.
2. Provide the command line sequence to run the program. Normally, this is simply the full pathname to the location of the program on your computer. Since VMD was downloaded to the default location on this local machine, the command line is:
   For Windows: `C:\Program Files\University of Illinois\VMD\vmd.exe`
   For Unix: `/usr/local/bin/vmd.exe`

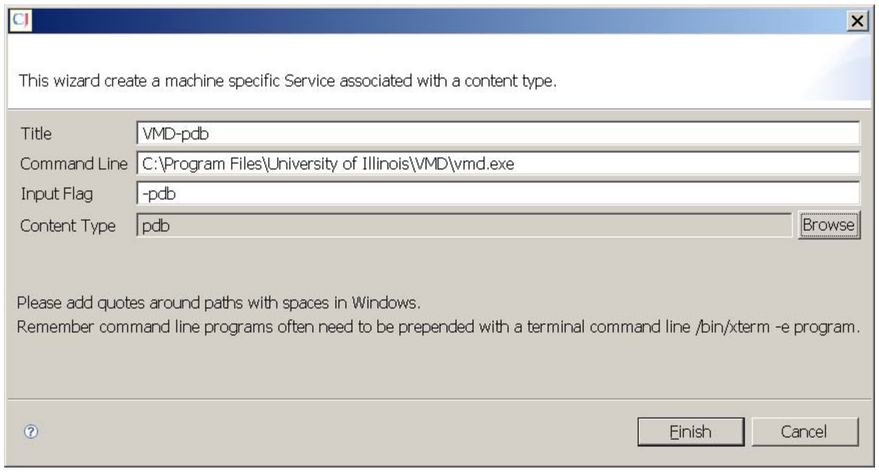
3. Supply the command line argument to precede the filename. For the VMD program to accept an input file in pdb format this is:
   -pdb
4. Select the *Content Type* to associate with this program.
   In the CJ, the *Content Type* associate with `pdb` files is named *pdb*. To find this type, click on the *Browse…* button, and select *pdb* then click the *OK* button (See Figure 59).



Execution Error File
PBS Job Ids
FASTA
GENBANK
EMBL
PIR
out
ps
tex
Blast_Report
Clustal
Stockholm
Phylip
Selex
HMM
HMMSearch_Report
gro
g96
pdb
itp
ndx
dat
mdp
trr
trj
tpr
tpb
tpa
edr
log
xvg
xtc
edi
edo
gtc
ppa
pdo
mtx
top
Interpolated_Context_Model
MASTER_JOURNAL_TYPE
MASTER_ENTRY_RELATION_TYPE
MASTER_CONTENT_TYPE

Figure 59. List of *Content Types* that can be selected.

The completed dialog box can be seen in Figure 60.



This wizard create a machine specific Service associated with a content type.

Title: VMD-pdb
Command Line: C:\Program Files\University of Illinois\VMD\vmd.exe
Input Flag: -pdb
Content Type: pdb

Please add quotes around paths with spaces in Windows.
Remember command line programs often need to be prepended with a terminal command line /bin/xterm -e program.

Figure 60. Completed dialog box to add the external viewer.

Now, whenever this user selects an *Entry* with content type, `pdb`, the *Entry* contents will be displayed with VMD.

The user must ensure that the external program is properly installed on his/her computer, and that the command line and input flag are specified correctly. It is a good idea to test the usage of an external viewer directly from the command line on the user's machine before attempting to define a new *Local Service*.

After successfully defining a local service, whenever an *Entry* with the corresponding content type is opened, the external viewer will be used to display the contents. Figure 61 shows a screen view after selecting the *Entry*, 1AK1. Because its content type is `pdb`, VMD was used to present its contents.

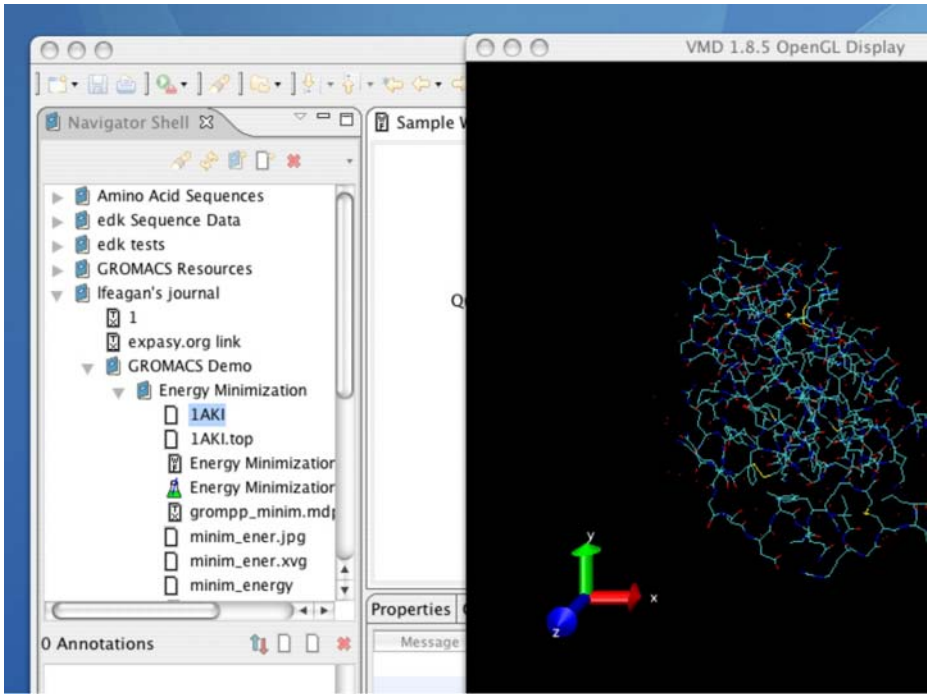


Figure 61. Screen view of *Entry* 1AK1.