CharGer©

A Conceptual Graph Editor by Harry Delugach

User's Guide

Contents

Introduction

CharGer is a conceptual graph editor intended to support research projects and education. Its current version is primarily an editor to create visual display of graphs. It is a research tool for conceptual graph researchers to explore implementation issues in conceptual graph interfaces. For some specialized purposes, it also supports an interface to Wordnet (a popular dictionary/thesaurus) and a repertory grid tool for acquiring requirements (CRAFT). For known bugs and limitations, see the section on Known Bugs and Limitations.

Users of the software should have some familiarity with conceptual graphs, including knowing about concepts and relations, type hierarchies and type/referent pairs. Knowing about actors will also be very helpful. For information about conceptual graphs, see the Web page: http:;//conceptualgraphs.org.

This manual documents CharGer as of its version v3.6 2008-08-06 which is supported by any Java VM with version 1.5 or higher.

Features currently supported

These features are currently supported (as of version v3.6 2008-08-06):

- Save and retrieve graphs from files, in an XML format called CGX (see Appendix). These are ordinary XML text files and are portable across all platforms.
- Save and retrieve graphs from earlier CharGer files, in a proprietary (i.e., non-standard) text format. These graphs have the extension .cg and are portable across all platforms.
- Save graphs in Conceptual Graph Interchange Format (CGIF) standard interchange format conforming to ISO/IEC 24707:2007 Annex B. These graphs have the extension .cgif and are portable across all platforms.
- Display a graph's CGIF version or a graph's (crude) natural language paraphrase, copy it to the clipboard as text, or save it to a text file.
- Save a graph in some graphics formats: EPS, PDF, SVG, EMF, GIF and PNG
- Copy/paste of graphs using an internal clipboard.
- Copy and paste from CharGer to other applications.
- Any number of graph windows may be opened for editing.
- Concepts, relations and actors are all supported for editing.
- Type and relation hierarchies may be edited and saved the same way as a graph, and may be intermixed on the same sheet of assertion.
- Graphs may be labeled as to their intent (e.g., query, definition, etc.)
- Contexts are supported, including arbitrary nesting. Negated contexts (“cuts”) are also supported.
- Save-before-close to prevent losing a graph, and other modifications to prevent losing data.
- Zoom in and out on a graph’s view
- Unlimited undo/redo for editing changes (limited only by available system memory)
- Portability to all major platforms (i.e., as portable as Java based on JDK 1.4.2)
- Activation of some built-in actors, including several primitive ones for arithmetic and elementary operations, with an optional “animation” to show their operation.
- Database access through actors, although restricted to tab-separated text file “databases” at present.
- User-written (in Java) actor plug-ins with a published plug-in interface.

- Capability to automatically create a skeleton graph from a database suitable for providing database semantics.
- Keystrokes to move and explicitly resize graph objects and contexts
- Keystrokes to switch editing tools.
- A natural language paraphrase feature, to paraphrase a graph in English (other languages on the way)
- Ability to set user preferences and save them between sessions.
- Some conceptual graph operations (e.g., join, match) (see Known Bugs and Restrictions)
- Ability to attach Wordnet glossary definitions or generic glossary entries to concepts and types
- Adjustable parameters for the matching scheme applied to joins and matching
- Custom colors for all graph objects, and the ability to set one's own default color scheme.
- Ability to perform some concept acquisition via repertory grids
- Ability to export a repertory grid to Burmeister (CXT) format.

Features not currently supported

The following useful features are not currently supported; plans are to implement them in the future. Users are urged to note the list of limitations and bugs on page 40.

- Validation facilities (except for enforcing CG formation rules)
- Pasting (as in copy/paste) from other applications to CharGer windows.
- Ability for actors defined within CGs (actor plug-ins, written in Java, are supported).
- Type and relation hierarchies in CGIF format.
- CGIF import (under development)

Why the name "CharGer"?

The University of Alabama in Huntsville has several sports teams nicknamed the "Chargers". A catchy name at that. Since the "CG" initials appear in it, it seemed a natural choice. And I like the notion of forging ahead, attacking research problems and leading the way. So there it is.

Installation

System Requirements

The minimum requirements for ***CharGer*** are:

- A Java Virtual Machine (VM) such as found in Sun's JDK or JRE, version 1.5 or higher.
- A color monitor, 800 x 600 resolution or higher.

Software

The installation procedure is as follows. You must have a Java installed (not Javascript) to run the editor. A Java VM of version JDK 1.5 or higher is required; see http://java.sun.com for information on how to get a freely available version, or use any commercial Java package. Java may already be installed on your system; to find out, get to a command prompt and enter "**`java –version`**". If the command doesn't return an error, make sure the version is 1.5 or higher.

To install the software, there are these two steps:

- Obtain the **`chargerNN.zip`** file.
- Un-zip the **`chargerNN.zip`** file. It should create a top level **`CharGer`** folder, containing the **`CharGer.jar`** file, a **`CharGerManual.pdf`** file and two sub folders, **`Databases`** and **`Graphs`**, which contain samples; you don't need these subfolders to run ***CharGer***.
- Double-click the **`CharGer.jar`** file, and you're on your way! Alternatively, you may want to run it from the command line as shown below.

Folder Structure

A simple folder structure is expected when running ***CharGer***. The top-level installation folder is shown as **`CharGer`** in these examples. After installation, the structure should look like Figure 0-1. Your own graphs and database folder can be located anywhere you want; see the Preferences Panel for how to tell CharGer where they are.

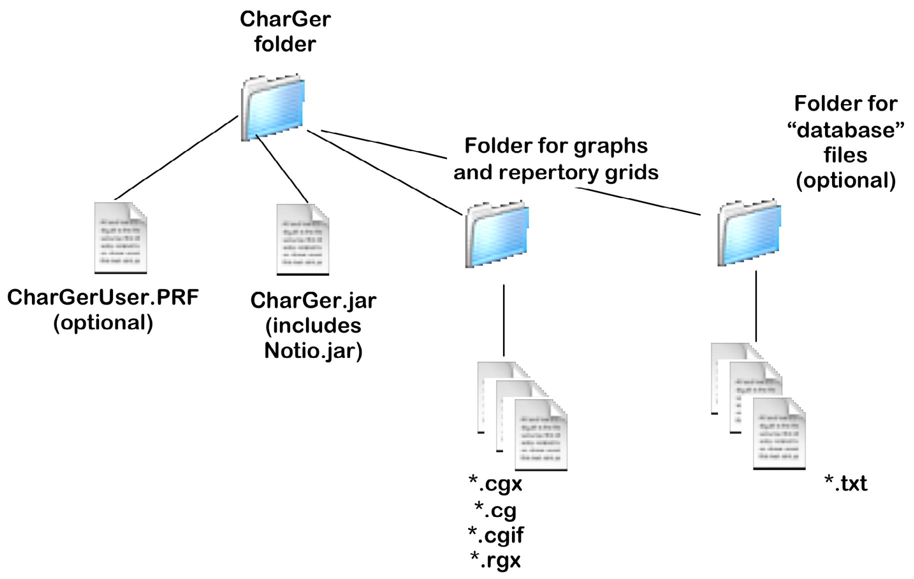

CharGer folder
Folder for graphs and repertory grids
Folder for "database" files (optional)
CharGerUser.PRF (optional)
CharGer.jar (includes Notio.jar)
*.cgx
*.cg
*.cgif
*.rgx
*.txt

Figure 0-1. *CharGer* **File and Folder Structure.**

A brief explanation of these is as follows:

`CharGer.jar`

Java archive (jar) containing the classes, icons and default preference file needed to execute ***CharGer***; these are platform independent at the JDK 1. 5 level. This file is ready to run as long as Java, or at least a Java Virtual Machine (VM) is installed on your computer. This file includes all needed external libraries, including the Notio package whose Web site is `http://backtrack.uwaterloo.ca/CG/projects/notio`, courtesy of Finnegan Southey (Thanks Finn!)

`Graphs` Where *CharGer* expects to find graphs (*.cg, *.cgif, *.cgx files) by default. Graphs can be opened and saved from/to any folder, however.

Note: The graph file fomat (.cg) has been changed! CharGer can read in these old files, but will only save files under the new 3.0 format. These files are not backward compatible with versions before 3.0.

`Databases` Where *CharGer* expects to find its tab-separated text databases for the **`<lookup>`** or **`<dbfind>`** actor. See below for more information.

Note: If you have created graphs under previous versions of *CharGer*, they will still work in the new version; however, you may have to copy your old graph files to your new graphs folder under the new **`CharGer`** folder.

If you're updating from CharGer 3.4b3 or earlier, you'll need to re-select your preferences and save them, or you can copy your old **`./Lib/Preferences.User`** or **`./CharGerUserPrefs.PRF`**. file to **`./CharGerUserPrefs.PREF`**. You may also discard your **`Lib`** folder (unless you're using it for something else!).

`Grids` Where *CharGer* expects to find its RGX repertory grid files. If you are not using CharGer for repertory grid acquisition, then you won't need this directory. See below for more information.

There is no particular reason why the Graphs, Databases and Grids directories are separate, except for convenience. There is a **`Browse For All`** button that will allow you to select one directory for all three kinds of files.

Preferences File

CharGer's preferences panel allows the user to save their own set of preferences. These are saved in the **`Charger`** folder as the **`CharGerUserPrefs.PREF`** file.

A list of preferences is built-in to the jar file called **`CharGerDefaultPrefs.PREF`**. Normally you won't see this unless you go browsing within the jar file's contents. Preferences consist of two kinds of lines:

- Parameter lines of the form:

  `Parametername=parametervalue`

  Note: Blank spaces in this line will probably invalidate the parameter line.

- Comment lines of the form

  `//= anything….`

  Note: Do not put leading spaces in front of the //=

  No apologies for the very restrictive syntax! A blank line will probably cause an exception.

GIF Images

There are a few **`.gif`** images that are used in the banner screen and as tool button images. They are part of the **`jar`** file and you shouldn't have to worry about them.

Running CharGer under JDK

The easiest way to start ***CharGer*** under JDK is through the command-line interface, invoking the following command in the top-level folder:

```
...\CharGer> java -jar CharGer.jar
```

This should invoke the application and bring up the ***CharGer*** Main Window.

> **Note:** Some users may experience problems getting this part going; please let me know what works and what doesn't. Previous versions of ***CharGer*** had somewhat longer names.

If you are unable to run a **`jar`** file directly like this, you should be able to unpack the .jar file (using the "**`jar xvf CharGer.jar`**" command) and work with the classes themselves. In that case, the command to run ***CharGer*** would be as follows (note the "." for the class path):

```
...\CharGer> java –cp . CGMain
```

Documentation

Documentation of ***CharGer*** consists of a set of HTML files (generated from javadoc) documenting the classes (primarily useful for developers), some tool tips and informative messages during execution, and this manual.

General Information

Bear in mind that the ***CharGer*** editor is currently (always?) a prototype version. Although it is unlikely to do serious damage to your computer files, it may not be useful at all. Please report all bugs, suggestions, etc. to the author: Harry Delugach (delugach@cs.uah.edu). The usual disclaimers hold with respect to my responsibility for any problems you may have with this software. In other words, **use at your own risk**.

User Windows

There are several kinds of windows in ***CharGer***. One is the Main Window, which looks something like Figure 0-1. The version number displayed on your Main Window may be different; yours reflects the actual version you're using.

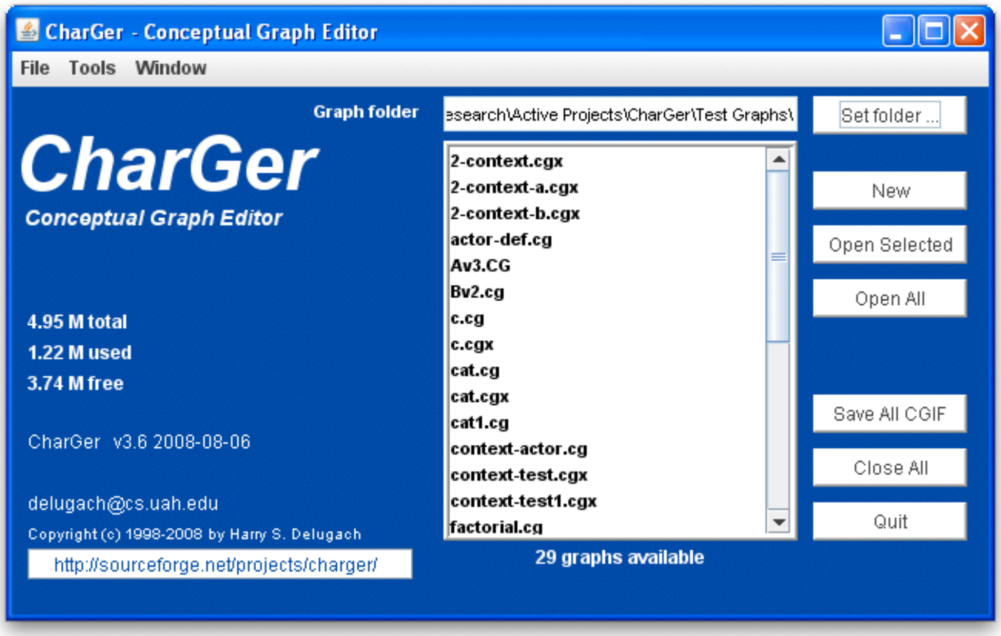


Figure 0-1. ***CharGer*** **Main Window.**

From this window, one or more editing windows may be opened. An editing window looks something like Figure 0-2. Details of its areas and buttons are given below.

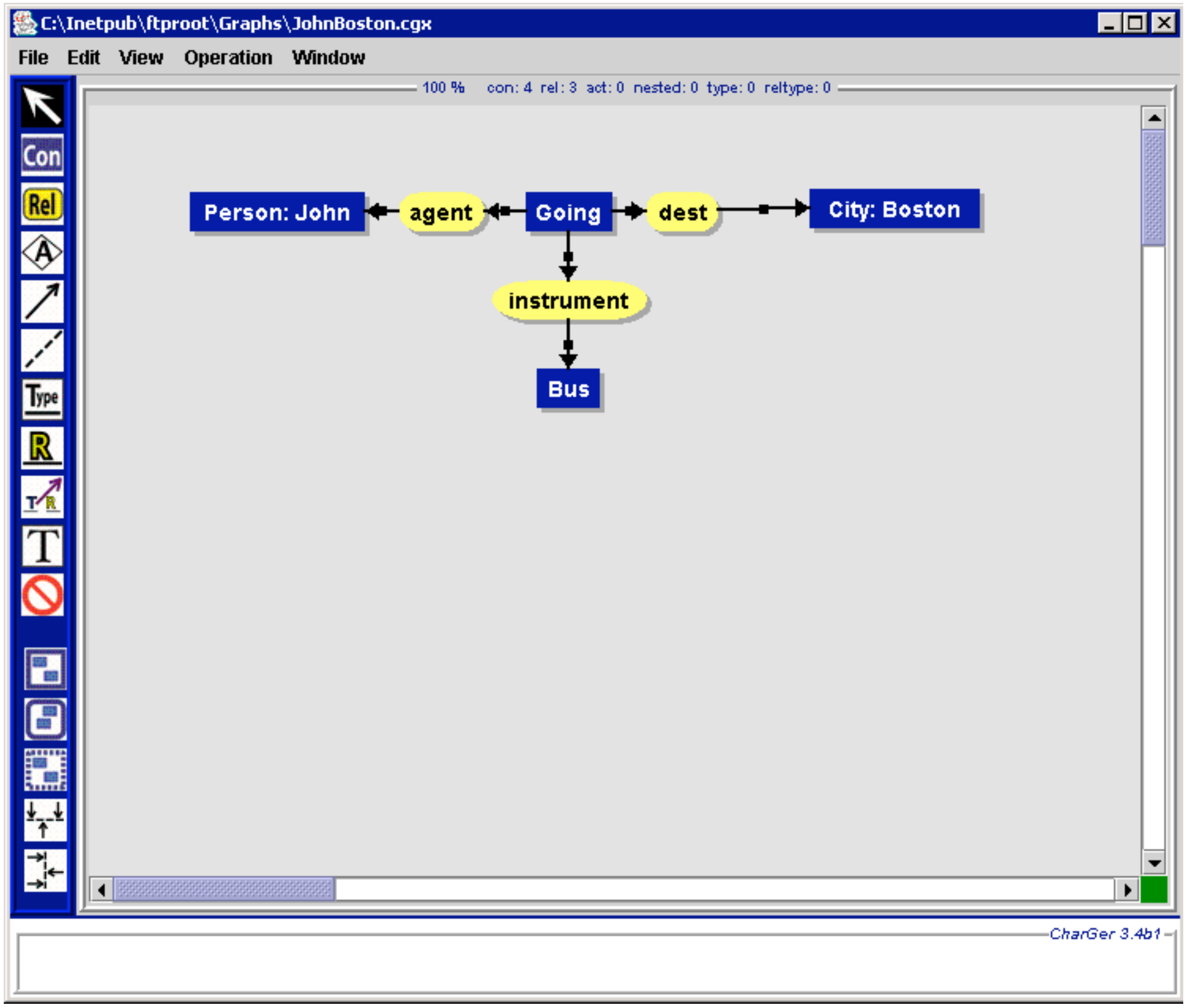

C:\Inetpub\ftproot\Graphs\JohnBoston.cgx
File Edit View Operation Window
100 % con: 4 rel: 3 act: 0 nested: 0 type: 0 reltype: 0
Person: John
agent
Going
dest
City: Boston
instrument
Bus
CharGer 3.4b1

Figure 0-2. Editing Window.

There are additional window types (e.g., Database Linking Tool Window and other display windows) described below.

GETTING HELP

Many buttons, fields, etc. have “tool tips” that will help you understand what they do. To view a tool tip, place the cursor above some element in a window. After a short pause, a one-line help description will appear. If nothing appears, then there’s no tool tip for that element; if you think it needs one, send a note to the author. In the meantime, look in this manual for more help.

GRAPH STRUCTURE

A **`graph`** in the ***CharGer*** environment is any collection of concepts, relations, actors and type labels, linked by their appropriate arrows, co-referent links, input/output arrows or super/sub-type arrows. A collection of un-connected elements is still considered a graph. All the elements of each graph are contained in a single file. The notion of multiple graphs means elements in multiple files.

We have introduced two new graphical elements not previously defined in conceptual graph display environments, namely, a **`typelabel`** and **`relationlabel`** shown as a text string above a horizontal line, as **`ANIMAL`**, This gives us an easy facility to draw type hierarchies and have them stored either separately, or included as part of another graph.

"MODALITY" OF A GRAPH

Several authors have made reference to a graph's *intent* or *purpose*, that is, whether the graph is intended as a query, an empirical observation, a rule, a schema, a definition, etc. Sometimes this may be an important aspect of the content of a graph and CharGer therefore include facilities for indicating such intentions. As yet, there is no further use for these labels; they can be safely de-activated in the current version.

The Graph Drawing Area

There is a drawing area in an editing frame that serves as the "canvas" on which the user draws a graph. This area operates as a constrained version of a typical picture-drawing program, except that it will try to enforce the conceptual graph formation rules.

Note: The "philosophy" behind the editor is that it supports an open world. That is to say, the user may enter anything he/she wants as long as it is well-formed by the general rules of conceptual graphs. If the user enters some element that is already defined, then the system will try to use that definition to enforce any constraints that result from it. If the user enters an element that is *not* already defined in the system, it can still be included, but no constraints will be enforced on it and little processing should be expected. Of course, the user is free to then provide definitions, facts, etc. that deal with the new elements.

Main Window

The main window lists the available graphs. This scrollable list has all the graphs available to the ***CharGer*** editor in the Graphs folder. The default naming scheme is as follows.

```
<GraphName> . cgx
```

Note: If modality labels are active (see below), then the naming scheme is:

```
<GraphName> . <modality> . cgx
```

where <GraphName> is string, <modality> is one of the labels described in the section on the Preferences Panel, and .cg indicates a graph file type according to its modality. The entire notion of modality can be turned on/off by a checkbox in the Preferences file.

Selecting one or more graphs from the list will cause each of them to be opened in an editing window when the **`Open Selected`** button is clicked. (see below).

Note: Use the **`Quit`** button or menu item to end the session.

Graphs are currently saved in a standard XML form. This ***CharGer*** XML form is an easy-to-use text form, strictly for purposes of saving graphs within the ***CharGer*** editor. As text, they are intended to be portable across all platforms. This format is not intended as a replacement or alternative for the standard CGIF format.

Main Window Buttons

The following buttons in the Main Window are supported:

Set Folder…

Lets you select a directory where your graph files are located. Once chosen, it will become the default directory for opening/saving and all the .CG files in that directory will appear in the window.

New

Creates an empty editor window, ready for creating a graph from emptiness.

Open Selected

Opens the selected graph(s) in the displayed list. Double-clicking a single name will do the same thing for a single graph. This button only opens graphs from the list; to open graphs from anywhere else, use the Open… menu item in the File menu.

Open All

Opens the selected graph(s) in the displayed list. Double-clicking a single selection will do the same thing. This button opens all the graphs in the list from the default directory; to open graphs from anywhere else, use the Open… menu item in the File menu.

Save All CGIF

Converts every graph in the list to CGIF form. Does not export type or relation hierarchies yet.

Note: due to some problems with the current Conceptual Graph Interchange Format (CGIF) standard, ***CharGer***'s layout information is temporarily unavailable when using CGIF files. This problem is being worked on and will be fixed in the near future. If layout information is important to you, then use the ***CharGer*** format (*.cg) for the graphs and simply save them as CGIF when you need to.

Close All

Close all editing windows. If a graph has been edited but not saved, you'll be prompted for each changed file as to whether you want to save it or not.

Quit

Aborts the entire ***CharGer*** thread, after giving the user a chance to save any un-saved graphs or grids.

Main Window Menus

There are three menus associated with the main window — the **`File`** menu, the **`Tools`** menu, and **`Windows`** menu. The menus are explained below.

File Menu

NEW

Creates a new editing window. Does the same thing as the New button.

OPEN …

Lets the user choose a graph using a standard file selection dialog window. Operated the same way as the Open button.

OPEN CGIF …

Lets the user open a CGIF (*.cgif) graph as chosen through a standard file selection dialog window.

PREFERENCES…

There is a preferences panel that allows the setting of some preferences by the user. Such settings are in effect for the current session only – ***CharGer*** always starts up with the preferences set from the **`Preferences`** file (see above).

QUIT

User gets a chance to save un-saved changes before finally quitting.

Tools Menu

There are two tool: the **`Database Linking Tool`** (described in Database Linking below) and **`Requirements Acquisition`** (see CRAFT subsystem, below). Other tools will go here.

Windows Menu

<GRAPH LIST>

If there are any already open graph editing windows, their names will appear here. The naming scheme is similar to the one that governs the file names (see above).

Editing Window

This is where most interesting work gets done. There are some editing and tool buttons to the left of the drawing area. The upper set of tool buttons are editing modes; the lower set are one-time command directives. The set of tools are described below. Figure 0-1 is a summary. Each tool is described below.

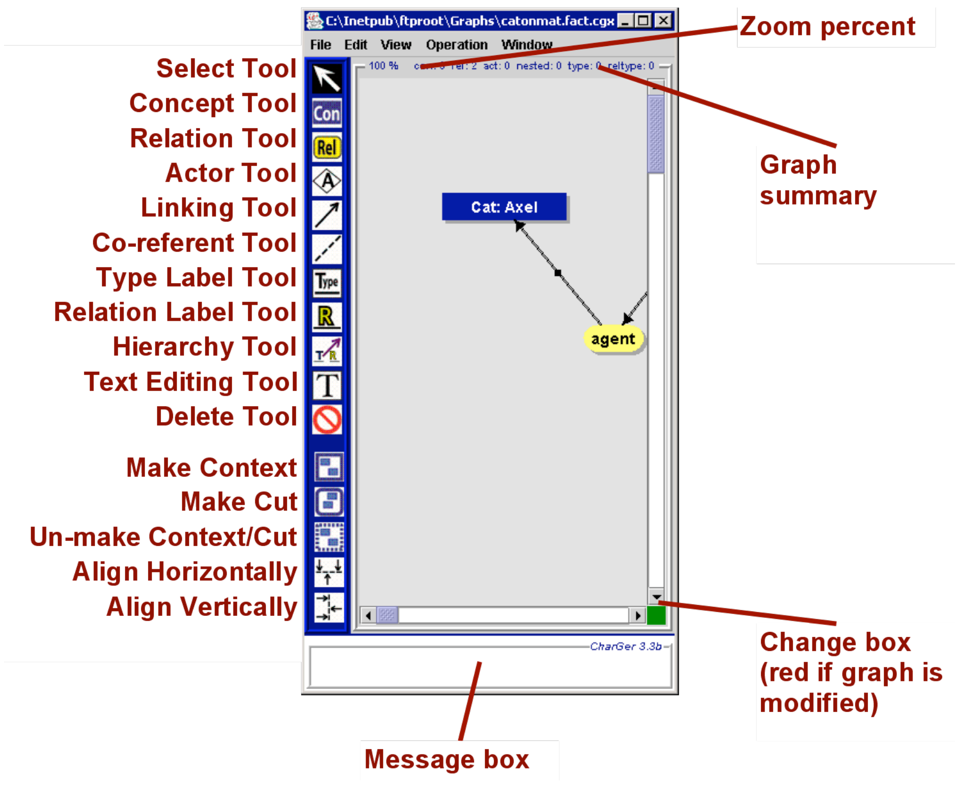


Figure 0-1. Basic Editing Features.

There are five menus on the Editing Window, three of which are shown in Figure 0-2.

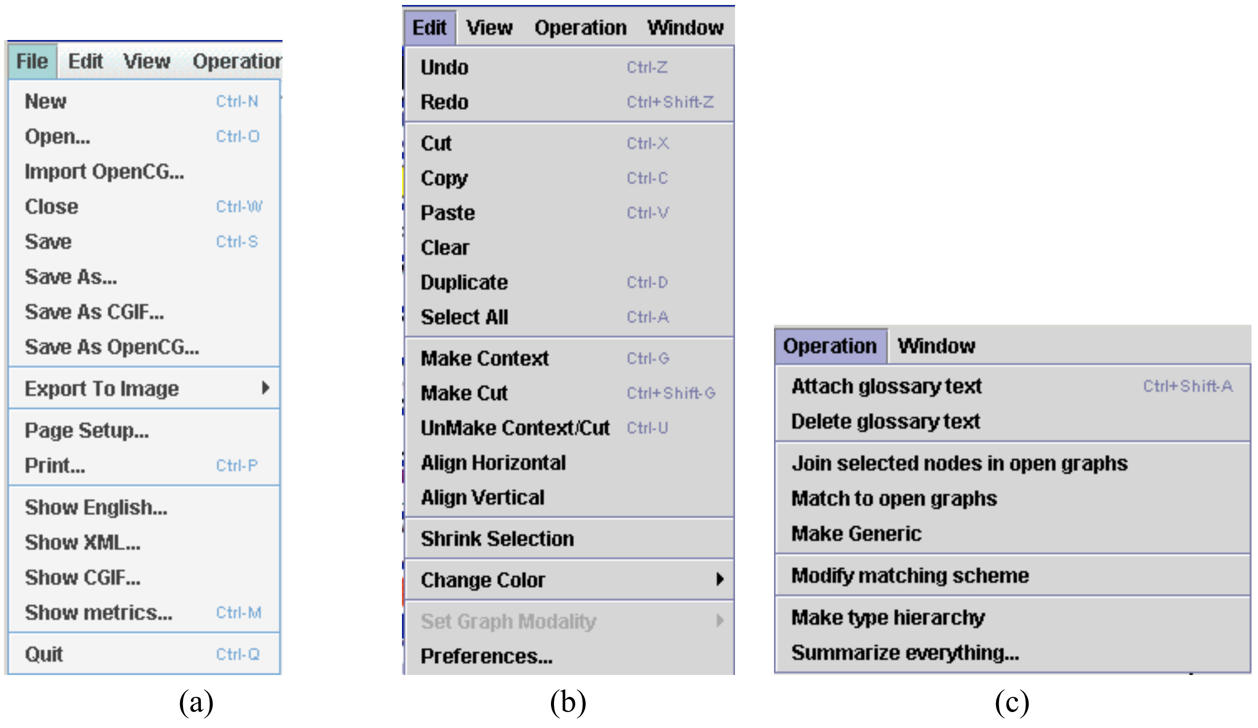


Figure 0-2. Some Menus of the Editing Window.

Menus

Some of the menu items are also found as command buttons.

File Menu

The File Menu items are as follows. Many of them perform the same functions as the tools and commands below.

NEW

Opens a new editing window, with an empty graph.

OPEN ...

Invokes a file open dialog for locating a graph file to be loaded. If the current file is un-saved, user has a chance to save it before opening a new one.

IMPORT CGIF ... (TEMPORARILY DISABLED)

A parser for CGIF files is being developed but is not yet available.

IMPORT OPENCG...

Invokes a file open dialog for locating an OpenCG (*.ocg) file to be loaded. See http://sourceforge.net/projects/opencgraphs/ for more information about OpenCG.

CLOSE

Closes the current graph. If it has not been saved in its present form, user is prompted before it closes.

SAVE

Saves the current graph without prompting for its name.

SAVE AS ...

Invokes a file dialog for naming a graph file to be saved.

SAVE AS CGIF...

Opens a file dialog where the user can specify a file in which to save a CGIF version of the graph. The graph's ***CharGer*** information (i.e., the screen layout) will be embedded in the CGIF form if the "Include CharGer info with CGIF" preference is checked in the Preferences Panel.

SAVE AS OPENCG...

Saves the current graph in its OpenCG version. See http://sourceforge.net/projects/opencgraphs/ for more information about OpenCG.

EXPORT TO IMAGE

Opens a sub-menu of format options that each open a file dialog where the user can specify a file in which to save an image of the file into several popular graphics formats. The file is then usable for inserting/editing by other software. The following export formats are supported:

Vector graphics (scalable, suitable for laser printing):

- EPS (Encapsulated Postcript)
- PDF (Adobe Acrobat)
- EMF (Enhanced Metafile) – Windows specific
- SVG (Scalable Vector Graphics)

Raster (bitmap, generally low resolution)

- GIF (Graphics Interchange Format)
- PNG (Portable Network Graphics)

There is no way to import a graphics file into ***CharGer***.

PAGE SETUP…

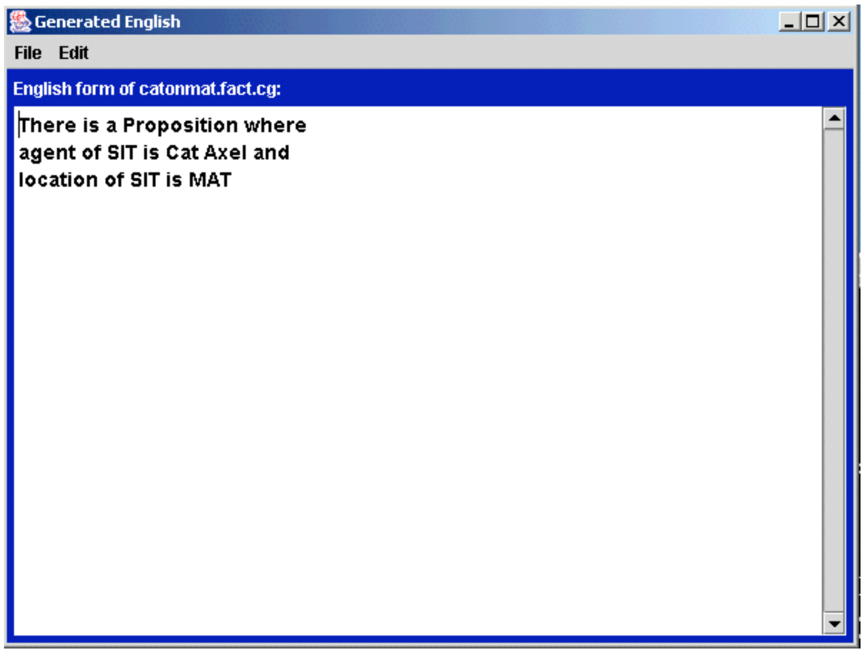
Invokes a page setup dialog, platform-specific to your operating system. Even if there are no changes to make, it's probably a good idea to click OK or Apply anyway (see Print… below). Page setup must be called once for each separate editor window you want to print.

PRINT…

Prints the display form of the graph on the currently selected printer. When invoking print in an editor window for the first time, you are first subjected to a page setup dialog with landscape mode already selected (see Page Setup above). It is recommended that you select landscape mode when printing out wide graphs. ***CharGer*** will attempt to scale your graph so that it will fit on a single page; there is no multi-page printing supported. (Page setup must be called once for each separate graph window you want to print.)

SHOW ENGLISH…

Create a natural language paraphrase of the current graph and displays it in a separate window, whose text can be selected and/or copied. The display looks like Figure 0-3. The natural language generating algorithm used here is a very simple one, primarily to illustrate feasibility. Only English is currently supported (apologies to my French-, German- and other-speaking friends).



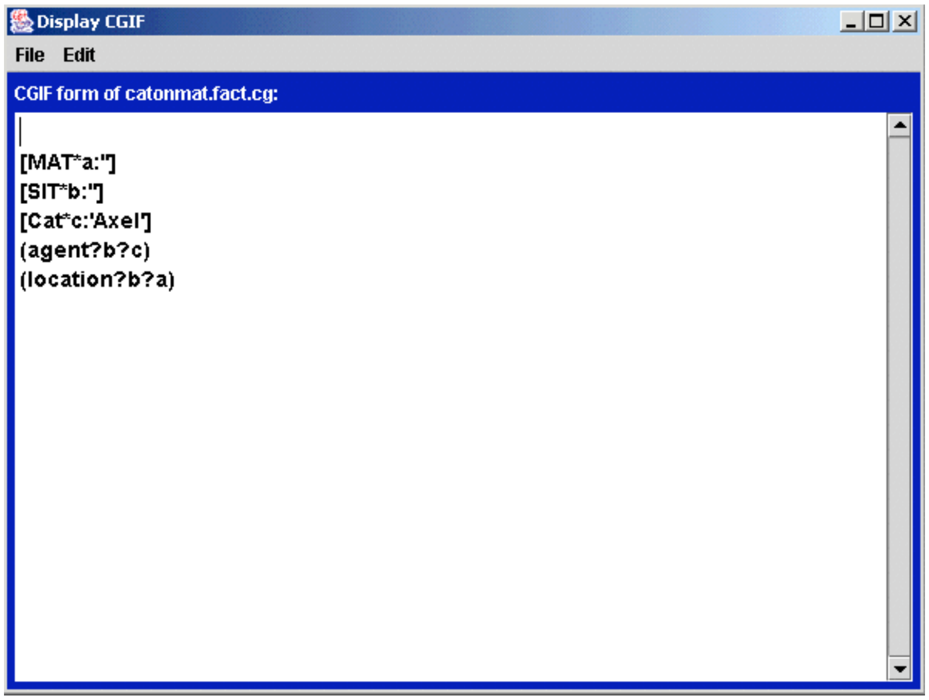
```
There is a Proposition where
agent of SIT is Cat Axel and
location of SIT is MAT
```

SHOW XML…

Displays the XML form of the graph in a separate window. The window's Edit menu allows you to copy part or all of the contents as text.

SHOW CGIF…

Displays the CGIF form of the graph in a separate window. The display looks like Figure 0-3. The window's Edit menu allows you to copy part or all of the contents.



```
[MAT*a:'']
[SIT*b:'']
[Cat*c:'Axel']
(agent?b?c)
(location?b?a)
```

Figure 0-3. Display CGIF window.

SHOW METRICS…

Some rudimentary metrics on the graph are displayed. This can be considered experimental at present; suggestions from users are welcome.

QUIT

User gets a chance to save un-saved changes before actually quitting.

Edit Menu

The items in the menu are as follows. Many of them perform identical functions to the tools and commands below.

Selections

When an edit item refers to "the selection" the reader should keep in mind some simple conventions:

- Edges (arrows, co-referent links, etc.) are in the selection if they are clicked on, shift-clicked on, or if the nodes they connect are selected.
- If a concept, relation, actor or type label is cut or cleared, its arrows, co-referent links, etc. are cleared with it, even if they lie outside the selection; it makes no sense to have an arrow without a graph node at both its ends.
- If a selection is pasted, only those arrows, co-referent links, etc. that are entirely within the selection are pasted; other arrows, co-referent links, etc. are not pasted, even if they were selected; this is because there is no practical way to connect the links' other ends.

Note: Cut, Copy and Paste operate on either regular text (when editing labels in graphs) or on graph objects (when drawing). They operate separately, however; cutting or copying text has no effect on cut or copied graph objects and vice versa.

UNDO/REDO

CharGer supports unlimited levels of "undo". Undo does not operate within the editing of a text field, although the previous contents of a text field (i.e., before text editing starts) can be restored through Undo/Redo.

Undo restores the graph to its state prior to the last altering action. A subsequent undo restores the state prior to that one, and so on. In general, a user can undo all the way back to when the graph was opened, was saved to a file, or was started on an empty (new) window. Redo "undoes" undo -- i.e., it restores the graph to its state after the last altering action was performed. After choosing "undo" or "redo" any action other than a subsequent undo or redo will erase any further actions to "redo", but leave the "undo" actions unaffected. Dimmed if there is nothing prior to undo or nothing to redo.

CUT

Copies the selection to the (internal) clipboard and deletes it from the graph. `Undo` restores it, but leaves it on the clipboard. If the user is editing a text field (see below), the system clipboard is used, allowing transfer of the text to another application. If the user has selected one or more contexts, all the contents of the context are also selected and cut to the clipboard. Dimmed if there is no selection.

If the user then pastes into CharGer, they are pasted as CharGer objects and can be edited in the usual way. If the user pastes into some other application, the selected objects will be pasted as a raster image and treated the same way that application treats other raster figures. The format of what's pasted can be controlled by a preference in the Compatibility pane.

Copy

Copies the selection to the clipboard, leaving it intact for the current graph. If the user is editing a text field (see below), the system clipboard is used, allowing transfer of the text to another application. If the user has selected one or more contexts, all the contents of the context are also selected and copied to the clipboard. Dimmed if there is no selection.

If the user then pastes into CharGer, they are pasted as CharGer objects and can be edited in the usual way. If the user pastes into some other application, the selected objects will be pasted as a raster image and treated the same way that application treats other raster figures. The format of what's pasted can be controlled by a preference in the Compatibility pane.

Paste

Inserts the contents of the clipboard. If graph objects were cut/copied in ***CharGer***, then they are pasted into the current editing window, where they can be edited in the usual way. **`Paste`** leaves the clipboard contents intact. If the user is editing a text field (see below), whatever text is on the system clipboard is used, even if it was copied/cut from another application. If the contents of the clipboard are from some other application besides ***CharGer***, unpredictable actions may occur. Dimmed if there is nothing on the clipboard.

Clear

Erases all the selected objects, without affecting the clipboard. **`Undo`** restores them. Dimmed if there is no selection.

Select All

If editing in a text box, the entire contents of the text box are selected. Otherwise, everything on the drawing window is selected.

Make Context

Make the current selection into a context. Use the Selection Tool to establish a selection before using the **`MakeContext`** Tool. If the selection would cause overlapping contexts, the user is prevented with a warning. Dimmed if there is no selection, or if the selection is already a context/cut. There is no direct way to make a cut into a context – you must first unmake the cut and then re-make it into a context.

Make Cut

Make the current selection into a "cut" (negated context). Use the Selection Tool to establish a selection before using the **`MakeCut`** Tool. If the selection would cause overlapping contexts, the user is prevented with a warning. Dimmed if there is no selection or if the selection is already a context/cut. There is no direct way to make a context into a cut – you must first unmake the context and then re-make it into a cut.

UNMAKE CONTEXT/CUT

Removes the outermost context/cut border in the selection, thus attaching its contents to the graph in which it is nested (or the entire graph if the context was not nested). If there are concepts, etc. selected that are not in the outermost context, they are ignored. If there is more than one equally nested outermost context, one of them is chosen arbitrarily to be un-made. Dimmed if there is no selection.

ALIGN HORIZONTAL

Same as Align Horizontal command button. Dimmed if there is no selection.

ALIGN VERTICAL

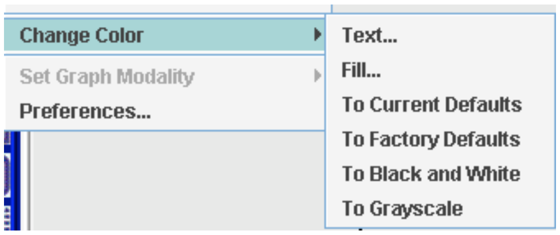
Same as Align Vertical command button. Dimmed if there is no selection.

SHRINK SELECTION

Make the selected objects as small as possible, shrinking around their centers. The same effect can be achieved by using the "-" key repeatedly. Dimmed if there is no selection.

CHANGE COLOR

Change the color of the selected objects. Dimmed if there is no selection.



There are six choices:

Text…	Change the text color of all selected objects, regardless of what kind.
Fill…	Change the fill color of all selected objects, regardless of what kind.
To Current Defaults	Change both the fill and text color of all selected objects to whatever default color scheme is currently set for their kind of object (see Preferences)
To Factory Defaults	Change both the fill and text color of all selected objects to ***CharGer***'s "factory" defaults (you can't change these).
To Black and White	Change the fill to white and the text color to black. Turn on the visible borders for each concept, relation, etc. Good for exporting and/or pasting when you're going to use a black and white printer.
To Grayscale	Change the fill to a gray scale color and change the text to either black or white. Suitable for higher-resolution printing

SET GRAPH MODALITY

A graph's modality is one of several labels. Whether you choose purpose labels or not, or what particular label you choose has no effect on the operation of ***CharGer***. They are for information and/or organizational purposes. To turn them on or off, look for the **Activate graph modality labels** in the Preferences. When they are activated, the menu looks like Figure 0-4.

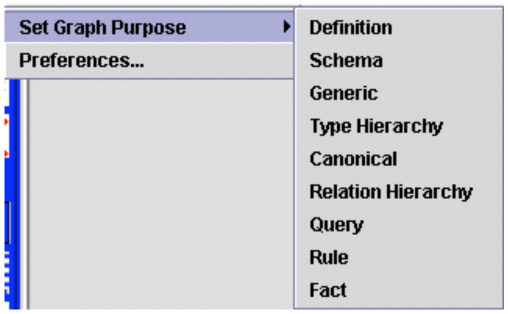


Figure 0-4. Graph Modality.

To see what modality is currently picked, look at the title bar of the editor window.

PREFERENCES…

Open the Preferences Window to make changes to the global preferences (see Preferences below).

View Menu

The only current options in the view menu are those relating to zooming. Zooming is a feature of the graph's appearance on the screen at a given moment; the graph itself keeps the same size it had when you began zooming.

ZOOM IN

Increase the size of the graph's appearance by an increment (currently 10% each time).

ZOOM OUT

Decrease the size of the graph's appearance by an increment (currently 10% each time).

ACTUAL SIZE

Restore the graph's appearance to its normal 100% size.

CURRENT SIZE

Shows the current zooming percentage. Not selectable by the user.

FIND…

Allows searching for a string anywhere in the current graph. If not found, a message will appear in the message box.

FIND AGAIN

Look for the previous string again. If not found, a message will appear in the message box.

SHOW GLOSSARY TEXT

Toggle to indicate whether glossary (definition) entries are to be displayed when some node is selected.

Operation Menu

This menu invokes operations that can be performed on a graph or selection. In the future, there are more operations planned for this window.

Attach glossary text

Allows the user to associate a glossary entry with this particular graph node. A glossary entry may be one of two kinds: one chosen from a set of Wordnet definitions or one supplied by the user. In order to use Wordnet definitions, you must have Wordnet 2.0 installed on your system.

Delete glossary text

Remove the associated text from the particular node. Of course, Wordnet itself is unaffected.

Join selected nodes in open graphs

Performs a join operation using two or more graphs, starting with the current graph. Each graph should have its own set of one or more selected items. If there is a valid way to join the graphs, according to the current matching scheme, a joined graph is created and opened in its own new window.

Note: Each graph has its own selection set, which can be maintained when switching between graphs. Be sure not to click on the drawing canvas when switching graphs, since clicking on the bare canvas will de-select whatever was previously selected. Use the graph window margin or the Windows menu.

Match to Open Graphs

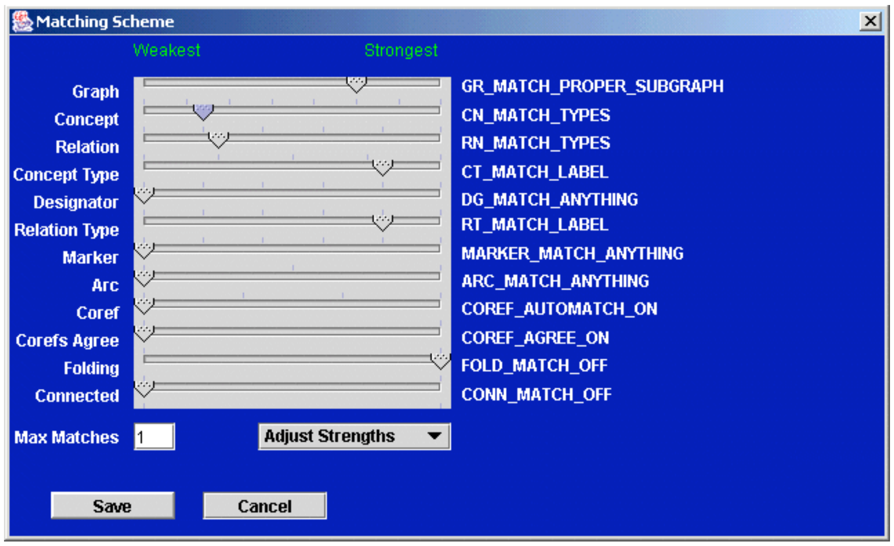
Matches the current graph to any other graphs in an open window. If a match is found, its result is displayed in a new window.

Make Generic

Remove the referents of the selected concepts and contexts. In other words, make individual concepts into generic ones. If a concept or graph has no referent, then it is unchanged. If a context has a graph descriptor, that descriptor is unchanged. Relations, actors, and types are unaffected.

Modify Matching Scheme

Allows the user to adjust the parameters of matching and joining. The command brings up a window like Figure 0-5.



Graph	GR_MATCH_PROPER_SUBGRAPH
Concept	CN_MATCH_TYPES
Relation	RN_MATCH_TYPES
Concept Type	CT_MATCH_LABEL
Designator	DG_MATCH_ANYTHING
Relation Type	RT_MATCH_LABEL
Marker	MARKER_MATCH_ANYTHING
Arc	ARC_MATCH_ANYTHING
Coref	COREF_AUTOMATCH_ON
Corefs Agree	COREF_AGREE_ON
Folding	FOLD_MATCH_OFF
Connected	CONN_MATCH_OFF

Max Matches 1
Adjust Strengths

Figure 0-5. Matching Scheme Parameters.

To find out what the slider positions mean, move the slider and see the meaning of the slider's parameter. (These parameters are obtained from Notio's MatchingScheme class, which defines the parameters for matching in Notio.)

Adjust Strengths

Options for adjusting the "strength" of a match are as follows:

Make Stronger	Move all sliders one position to the right (stronger)
Make Weaker	Move all sliders one position to the left (weaker)
Make Strongest	Move all sliders to their rightmost position
Make Weakest	Move all sliders to their leftmost position

If a slider is already at its limit, no change is made to that slider.

MAKE TYPE HIERARCHY

Find all super- and sub-type links in all open files and attempt to construct a consistent hierarchy out of them. This hierarchy will be placed in a new untitled graph window and can be edited as the user wishes.

SUMMARIZE EVERYTHING

Using all open graphs and all open repertory grids (if any), paraphrase their content in poor English. The paraphrase appears in a text window that can be copied, saved, etc.

Windows Menu

The **`Windows`** menu consists of the following items:

<GRAPH LIST>

If there are any already open graph editing windows, their names will appear here. The naming scheme is similar to the one that governs the file names (see above).

BACK TO MAIN WINDOW

Allows convenient return to the main window.

Tools

Selection Tool

The selection tool is the basic way to make a selection or to move graph objects. In general, connecting lines are selected with what they connect. Double-clicking with the selection tool will invoke the EditText operation on whatever object has been selected.

- **To move a single object**, click on it and drag it where you want it. To move an object from one context to another, click on it and drag the same way.

  Note: Moving an object in or out of a context will delete any links it has between relations and concepts. This is because CG rules don't allow a relation to cross context boundaries. Coreferent links are preserved however. Links to actors can be preserved if the option is set in the Preferences Panel.

- **To select one or more objects as a group**, drag a rectangle across the objects you want selected. The objects should be "highlighted". You can then move them by moving any of the selected objects, or you can perform other operations. See **`Make Context`**, **`Shrink Selection`**, or the cut/copy/paste operations.
- **To add objects to a selection**, hold down the SHIFT key while dragging a new rectangle.
- **To select a context**, click somewhere on its outlined border.
- **To select an arc** (rendered as an arrow), a line of identity (coreferent link), or a generalization/specialization arrow, click on the solid dot (•) at its midpoint. In general, selection ignores arcs, because their meaning is inherent in the concepts/relations/actors/types that they link. In other words, as the nodes are selected/deleted/moved, so are their corresponding arcs.
- **To cancel a selection**, click somewhere on the drawing area that is not occupied by a graph element, or click another tool.

The selection tool can be chosen by pressing "S" or the space-bar.

Concept Tool

The concept tool allows you to insert a new concept onto the graph drawing area. To insert a concept on the drawing area, select the concept tool, then click where you want the new concept to appear. A new concept has a type label of **T** with no referent. To change a concept type or referent, open a Text Edit Box (see p. 25) on its name. The tool remains selected until another tool is selected. The concept tool can also be chosen by pressing the "C" key on the keyboard.

Relation Tool

The relation tool allows you to insert a new relation onto the graph drawing area. A new relation has the name **link**. To insert a relation on the graph drawing area, select the relation tool, then click where you want the new concept to appear. To change a relation's link name, open a

Text Edit Box (see p. 25) on its name. The tool remains selected until another tool is selected. The relation tool can also be chosen by pressing the "R" key on the keyboard.

Actor Tool

The actor tool allows you to insert an actor onto the graph drawing area. To insert an actor on the graph drawing area, select the actor tool, then click where you want the new concept to appear. The actor has the name `f`, which can be changed with the text editing features. The tool remains selected until another tool is selected. The actor tool can also be chosen by pressing the "A" key on the keyboard.

Link Tool

The link tool allows you to connect concepts with relations or actors, according to the normal rules of conceptual graphs. To draw a link, select the tool, then click and drag from one concept (relation or actor) to a relation or actor (concept). The link tool can also be chosen by pressing the "." (period) key on the keyboard.

Line of Identity Tool

This tool draws a coreferent link or line of identity between two concepts, thereby connecting members of a coreference set. To create a coreferent set of more than two members, each member must be linked (via the identity link tool) to at least one other member of the set. Coreferent links for a single coreference set should be drawn between a "dominant" concept and a "subordinate" concept. Redundant links are permitted but add no new information to the graph. The line of identity tool can also be chosen by pressing the "I" key on the keyboard (for line of identity)

Type Label Tool

The type label tool allows you to insert types (usually in a hierarchy) onto the graph drawing area. The type label tool can also be chosen by pressing the "T" key on the keyboard.

Relation Type Label Tool

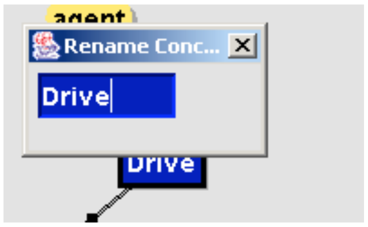
The relation type label tool allows you to insert relation types (usually in a hierarchy) onto the graph drawing area. The relation type label tool can also be chosen by pressing the "L" key on the keyboard.

Generalization/Specialization Line Tool

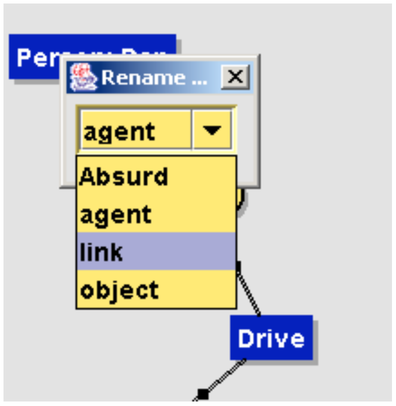
This tool draws a generalization/specialization relationship between two concept types or between two relations. The generalization/specialization tool can also be chosen by pressing the "," (comma) key (an un-shifted "<") on the keyboard.

Edit Text Tool

T With this tool chosen, a click on a concept, relation, actor, context border, or type label will open a Text Edit Box for the selected element's label. For a concept or context, a textedit box will appear, as in Figure 0-6(a). For an actor or relation, a pop-up menu will show the list of pre-defined actor labels from which to choose, as in Figure 0-6(b). The user may pick an actor/relation name from the menu or create a new (undefined) actor by typing in the desired name. To edit the label on an arrow, click on the square dot and edit its label.



(a)



(b)

Figure 0-6. Editing Node Names.

Editing either a relation/actor name or text label is done with a small editing window that appears when needed. Text labels for concepts, relations and contexts can be changed through use of a Text Edit Box as in Figure 0-6(a). Open a Text Edit Box by doing either of the following:

- With the Selection Tool, double-click on the label you want to change.
- With the Edit Text Tool selected, click once on the label you want to change.

Use conventional text editing keys to make your changes. To save the changes, press Return or Enter.

To edit a relation, actor, type label or relation label, a pull-down list of available labels appears. You may choose one from the list, or enter a new one.

Delete Tool

With this tool chosen, clicking on a concept, relation, actor, context border, or type will delete it. To delete an arc (rendered as an arrow), click on the solid box (■) at its midpoint.

Shortcut Keys

Certain keystrokes can be used in the editing window, when not editing text. The shortcuts shown in menu items are available whenever their corresponding menu command is available. In addition, the following hotkeys perform some useful functions:

SHIFT	adds to the current selection when clicking the mouse or dragging across a selection
	increases increment when moving or resizing with arrow keys

ARROW KEYS:	
LEFT	moves the selected object(s) to the left one pixel (with SHIFT key, four pixels)
RIGHT	moves the selected object(s) to the right one pixel (with SHIFT key, four pixels)
DOWN	moves the selected object(s) down one pixel (with SHIFT key, four pixels)
UP	moves the selected object(s) up one pixel (with SHIFT key, four pixels)

S or SPACE	make the selection tool active
C	make the concept tool active
A	make the actor tool active
, (comma)	make the arrow tool active (unshifted >)
I	make the line of identity tool active
T	make the type label tool active
L	make the relation type label tool active
. (period)	make the generalization/specialization tool (unshifted <)
=	enlarge the selected objects by one pixel each (with SHIFT key, four pixels)
- (minus)	reduce the selected objects by one pixel each (with SHIFT key, four pixels) Note: due to a Java compatibility issue, you may need the "-" key on the numeric keypad to achieve this operation

Command Buttons

Make Context

Make the current selection into a context. Use the Selection Tool to establish a selection before using the **`MakeContext`** Tool. If the selection would cause overlapping contexts, the user is prevented with a warning.

Make Cut

Make the current selection into a negated context ("cut") displayed with a rounded rectangular border. Use the Selection Tool to establish a selection before using the **`MakeCut`** Tool. If the selection would cause overlapping contexts, the user is prevented with a warning.

UnMake Context

Removes the outermost context border in the selection, thus attaching its contents to the graph in which it is nested (or the entire graph if the context was not nested). If there are concepts, etc. selected that are not in the outermost context, they are ignored. If there is more than one equally nested outermost context, one of them is chosen arbitrarily to be un-made.

Align Horizontally

Aligns all objects in the current selection horizontally, along their center points.

Align Vertically

Aligns all objects in the current selection vertically, along their center points.

Preferences Window

There is a preferences window available that allows the user to adjust some settings for their own use. These settings will override those in the **`CharGerDefaultPrefs.PREF`** file, which is loaded at ***CharGer*** startup, but they will persist for that session only, unless the user presses the **`Save Preferences`** button. User-saved changes are in the optional **`CharGerUserPrefs.PREF`** file. No changes are made by ***CharGer*** to its own **`CharGerDefaultPrefs.PREF`** file; those changes must be made through a regular text editor outside of ***CharGer***.

Note: Changes in the **`Preferences`** window take effect immediately, except for font changes which are generally reflected in future windows, but not current ones.

Preferences are arranged in three panels, Appearance, Compatibility, and Actor Settings. All of them have the following common options.

File Menu – Close

Closes the preferences window without saving anything

File Menu – Save Preferences

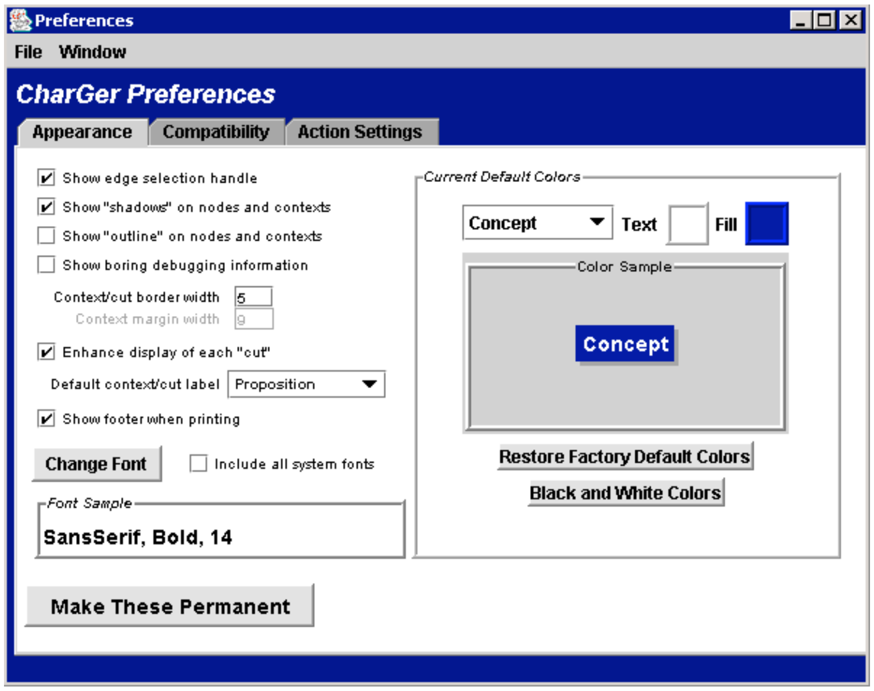
Ordinarily, changes made in the preferences panel take effect immediately, but do not persist between sessions. In order to make the preferences permanent, choose Save Preferences. This updates the CharGerUserPrefs.PREF file (or creates it, if the file does not already exist).

Make Preferences Permanent Button

Does the same thing as the Save Preferences menu item.

Appearance

The appearance panel looks like this:



- [✓] Show edge selection handle
- [✓] Show "shadows" on nodes and contexts
- [ ] Show "outline" on nodes and contexts
- [ ] Show boring debugging information

Context/cut border width 5
Context margin width 9

- [✓] Enhance display of each "cut"

Default context/cut label: Proposition

- [✓] Show footer when printing
- [ ] Include all system fonts

Font Sample: SansSerif, Bold, 14

Concept — Text — Fill

Show edge selection handle

Display the small selection box at the midpoint of each edge; this is automatically disabled for printing regardless of the setting shown.

Show "shadows" on nodes and contexts

Draw a gray "shadow" to give a three-dimensional effect. This option affects both the screen display and printing. The color sample will reflect the current setting.

Show “outline” on nodes and contexts

Show a solid border on all nodes and contexts; this is useful in black-and-white color schemes according to the user’s color preference. The color sample will reflect the current setting.

Show boring debugging information

Display some internal information in the editing window, and on the console (command-prompt window). This is primarily useful in reporting a problem to the developer(s). Most users probably want to leave it unchecked.

Context Border Width

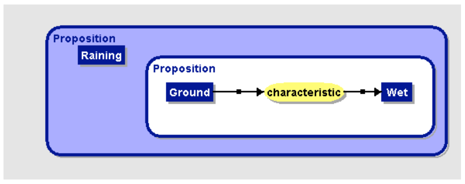
Width in pixels of a context's drawn border. Sets the margin width automatically.

Context Margin Width

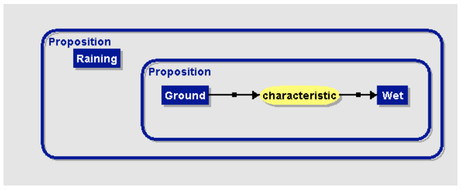
Minimum number of pixels from the outside of a context's drawn border to the closest inner component. Automatically adjusted depending on the context border width and whether to show shadows or not.

Enhance display of each “cut”

Apply shading to the cuts so that they are more prominent and more distinct from non-negated reqular contexts. Here’s an example of the enhancement effect:



When checked



When un-checked

Default context/cut label

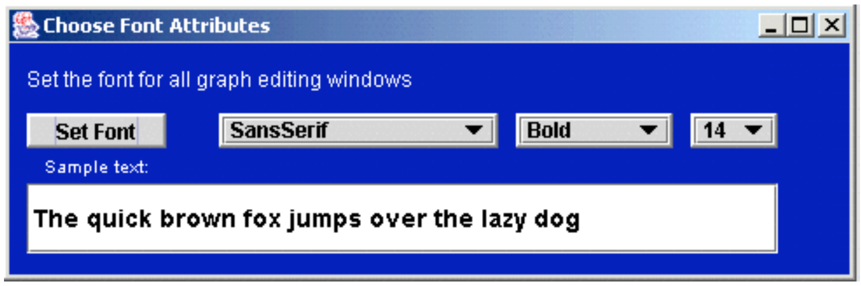
Choose what label will be used for contexts or cuts when they are first formed. This was suggested as a way to make un-labeled cuts without having to edit the context name every time. Of course, the user can change any context or cut's label by double-clicking on its border, which will bring up an editing box on the context/cut label.

Show footer when printing

Include a footer when printing a graph that will show the graph's full path name.

Change Font…

Brings up a brief dialog box to set the display font for graph labels. A sample showing the name, style and size of the font is shown. The dialog box looks something like Figure 0-1.



Set the font for all graph editing windows

Sample text:

The quick brown fox jumps over the lazy dog

Figure 0-1. Font Chooser Dialog.

To make the displayed font active, click on "Set Font" in the dialog box, which will make the displayed font active and close the font window. This font change will take effect immediately but will not be remembered between sessions. To make the font change permanent for subsequent sessions, click on "Save Preferences" in the Preferences Panel.

Include all system fonts

Includes all fonts available on the system. This may be a very long list, and the selected font may not be portable if the graph files are moved to another system where the fonts may not be installed. If left un-checked, only a small set of platform-independent font names will be shown.

Current Default Colors

There are several features which you can use to control the colors of graphs you draw. They are grouped together in the Current Default Colors section of the Appearance Panel.

Object Pull-down menu (Concept, Relation, etc.)

Use this to select which kind of object's colors are to be examined/changed. The Color Sample area shows you what that object currently looks like.

Text

Click here to change the color of the text for the kind of object you've selected. The Color Sample area shows you what that object currently looks like. No graphs are changed yet.

Fill

Click here to change the color of the fill for the kind of object you've selected. The Color Sample area shows you what that object currently looks like. No graphs are changed yet.

Restore Factory Default Colors

Changes the text and fill colors to the factory default scheme (you cannot change it). No graphs are changed yet; that is accomplished by the Change Color menu item in the editing window. The Color Sample area shows you what the current object looks like; however, ALL object types have been restored to these defaults.

Black and White Colors

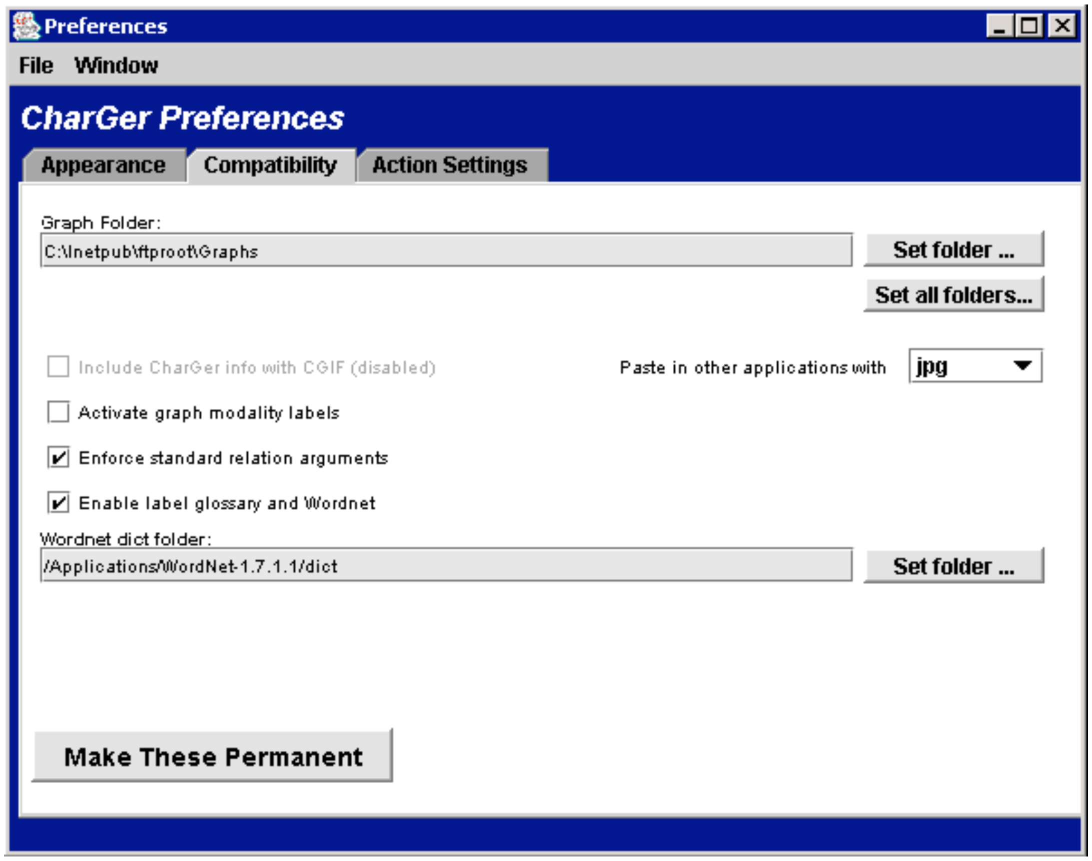
Changes the text and fill colors to a pre-defined black and white scheme (you cannot change it). This scheme can be useful when printing to a black and white printer. No graphs are changed yet; that is accomplished by the Change Color menu item in the editing window. The Color Sample area shows you what the current object looks like; however, ALL object types have been restored to these defaults.

Save All Preferences

See above.

Compatibility

The Compatibility panel looks like this:



Graph Folder: C:\Inetpub\ftproot\Graphs

☐ Include CharGer info with CGIF (disabled)

Paste in other applications with: jpg

☐ Activate graph modality labels

☑ Enforce standard relation arguments

☑ Enable label glossary and Wordnet

Wordnet dict folder: /Applications/WordNet-1.7.1.1/dict

Graph Folder

The default folder from which the graph list in the main window will be constructed. Saved graphs will ordinarily be saved in the same directory from which they were read. In open and save file dialogs, the user can navigate to whatever folder they wish. The **Set folder…** button allows the user to change this default folder. **Set all folders…** sets the graph, database and grid folders to the same name.

Include CharGer info with CGIF (currently DISABLED)

Include the ***CharGer*** information as CGIF comments, most importantly each component's layout on the canvas. This affects both the Display CGIF command and saving to a CGIF file.

Note: If ***CharGer*** information is not included in a saved CGIF file, then ***CharGer*** will not be able to lay out the graph on the screen, although its contents will be correct. All objects will be on top of one another at the top left of the screen.

Note: due to some problems with the current Conceptual Graph Interchange Format (CGIF) standard, ***CharGer***'s layout information is temporarily unavailable when using CGIF files. This problem is being worked on and will be fixed in the near future. If layout information is important to you, then use the ***CharGer*** format (*.cg) for the graphs and simply save them as CGIF when you need to.

Activate graph modality labels

Keep track of whether a given graph is generic, a fact, etc. The labels and their meanings are:

`def`	Definition
`rule`	Rule
`generic`	Generic
`type`	Type Hierarchy (for `convenience`)
`fact`	Empirical fact
`query`	Query
`schema`	Schema
“” `(empty string)`	None of the above

Enforce standard relation arguments

Enforce the standard relation rule such that a relation has at most one output argument. Never enforced for actors, except that pre-defined actors must adhere to their input/output arity signatures.

Enable label glossary and Wordnet

Enable the feature of including a glossary entry with each concept or relation. If Wordnet is not available, you can still include your own glossary entries with each concept or relation.

Wordnet dict folder

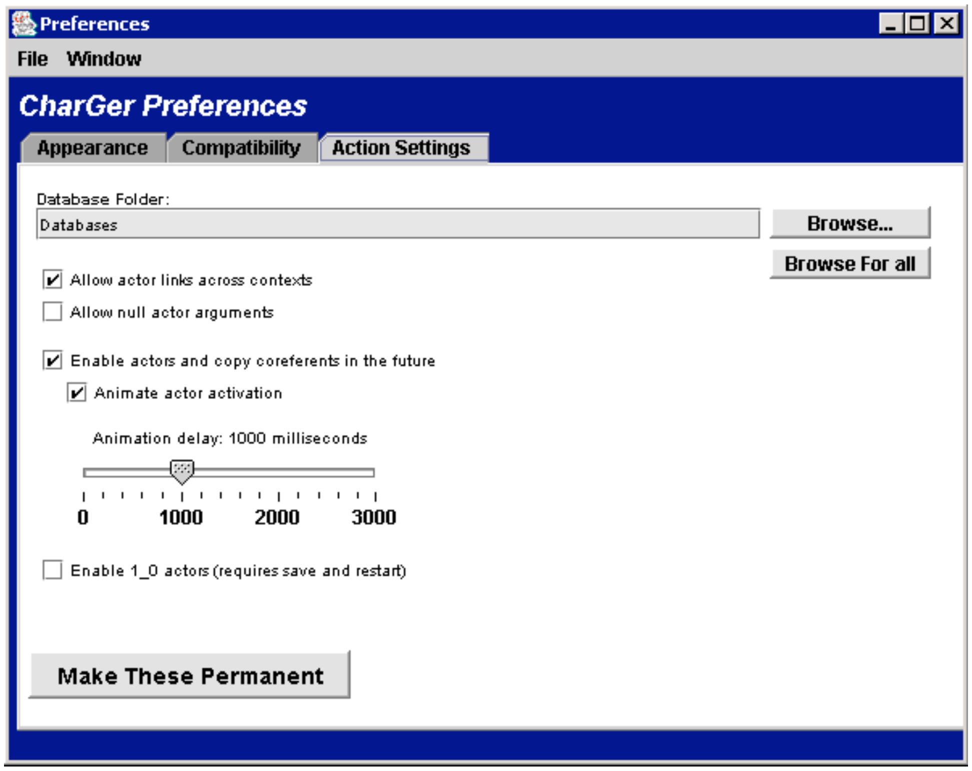
Wordnet’s dictionary is a set of files usually found within a folder named dict; here is where you specify its actual location. Browse… lets you look for the folder’s files on your system. This should usually need to be set only once; moving Wordnet around ought to be a rare occurrence.

Paste in other applications with

When copying all or part of a graph for use in another application, this option controls what format will be used for the copied image. The default is “jpg” but it can be any of the five graphics formats to which CharGer can export (see the File menu for details).

Action Settings

The Action Settings panel looks like this:



Database Folder:
Databases

[✓] Allow actor links across contexts
[ ] Allow null actor arguments
[✓] Enable actors and copy coreferents in the future
[✓] Animate actor activation
Animation delay: 1000 milliseconds
[ ] Enable 1_0 actors (requires save and restart)

Database Folder

The default folder in which database files will be looked for. Database files are used in activating actors (see **Database Linking Tool**). The **Set folder…** button allows the user to change this default folder. **Set all folders…** sets the graph, database and grid folders to the same name.

Allow actor links across contexts

The ANSI standard does not allow relation links across context boundaries (whether an actor or relation). Strictly speaking, an actor should link to a concept in its own context, with a coreferent link from that concept into the other context. Checking this box allows an actor link to connect to any concept, regardless of its context.

Allow null actor arguments

If checked, considers a null argument legal to an actor. Usually means that the actor will ignore the argument.

Enable actors and copy referents in the future

If checked, actors will be enabled for the rest of the session. Un-checking it later may cause unpredictable and/or undesirable consequences, particularly if some actors involve executable plug-ins that are active.

Animate actor activation

Whether to activate actor firings in increments, visually marking those actors and concepts that are involved in each firing.

Animation Delay

The time increment between firings, in milliseconds. Not visible unless actor firing is animated.

Note: The Preferences panel is under constant development. Use tool tips to find out more about individual options – hold the cursor over the option you want to find out more about.

Warning: If an option is shown in the preferences panel as "unstable", use them at your own risk; they are unlikely to help you out very much and will likely lead to errors.

Enable 1_0 actors (requires save and restart)

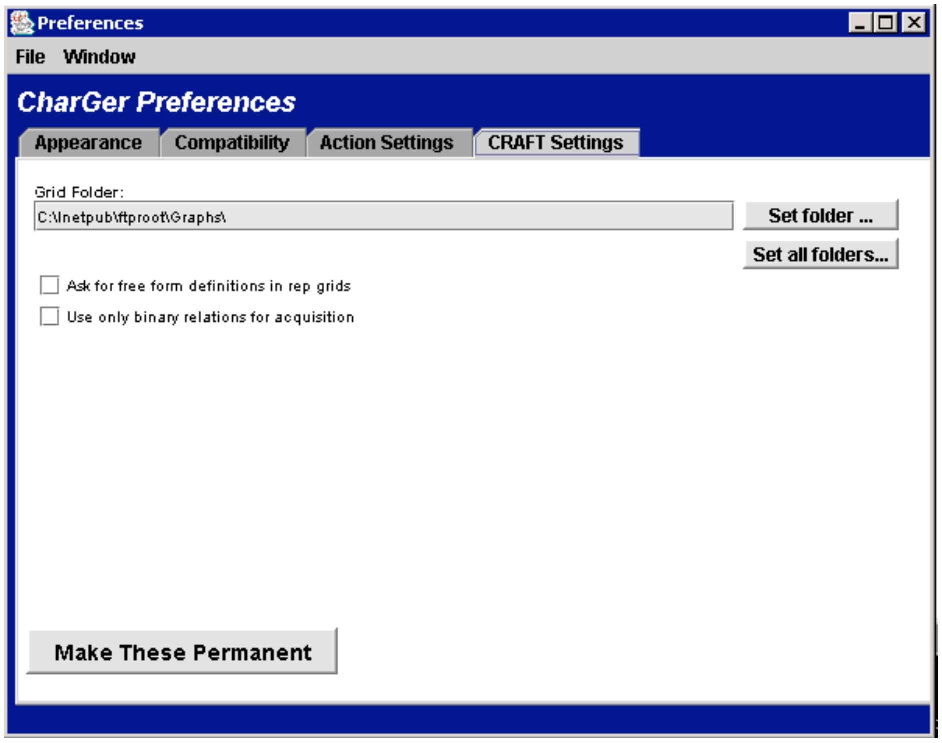
There are a number of built-in comparison actors that produce a zero or one, rather than a true or false. If you want to use these, then check this box, save the preferences and then quit and restart CharGer to take effect.

CRAFT Settings (optional)

Note: If you aren't interested in the repertory grid tools, then ignore this section!

CharGer has a built-in experimental repertory grid interface that can be used for your entertainment. To enable CRAFT, quit CharGer, open the CharGerUserPrefs.PREF file and find the option named "`craftEnabled`" and set it to "`true`". Once saved, this should enable the CRAFT subsystem in CharGer.

These settings involve the CRAFT knowledge acquisition subsystem. The CRAFT Settings panel looks like this:



Grid Folder:
C:\Inetpub\ftproot\Graphs\

☐ Ask for free form definitions in rep grids

☐ Use only binary relations for acquisition

Grid Folder

The default folder in which repertory grid files will be looked for. Grid files are the result of acquisition using a repertory grid (see **Requirements Acquisition**). The **Set folder…** button allows the user to change this default folder. **Set all folders…** sets the graph, database and grid folders to the same name.

Ask for free form definitions in rep grids

If checked, will allow the user to provide their own definitions in addition to Wordnet's. If un-checked, then only Wordnet senses will be accessed. If Wordnet is unavailable and this is un-checked, then no glossary entries (definitions) will be accessible.

Use only binary relations for acquisition

If checked, only relations with exactly one input and one output will be included in the CRAFT main window. If un-checked, every pair of related concepts (or contexts) will be included in the CRAFT main window.

Actor Activation

CharGer supports actors, generally using the techniques described in Sowa's original book. There are a few built-in actors, with pre-defined semantics. Most of these are simple arithmetic operations. For example, there is a plus actor to implement addition. Consider the following graph. It would be good practice for you to draw the graph in Figure 0-1 before proceeding:

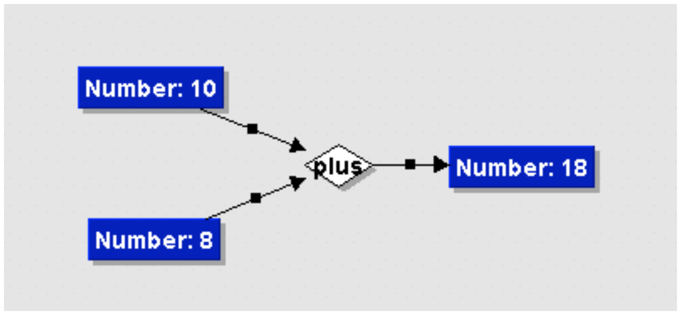


Figure 0-1. Actor example.

Note that the output number's referent is the sum of the two input number's referents. To understand how an actor works, draw the graph above, and then change the input **`Number: 10`** to read **`Number: 5`**. Note how the output number changes, as shown in Figure 0-2:

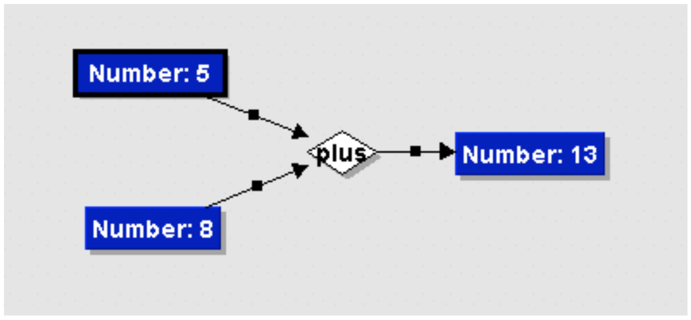


Figure 0-2. Changed actor graph.

Now change the output concept **`Number: 13`** to some other number. Note how it changes back to 13. The reason is that the plus actor denotes a functional dependency where the output concept is functionally dependent upon the input concepts; thus changing it causes the graph to re-evaluate itself and restore the original constraint.

The following executable actors are built-in to ***CharGer***. ⊤ means any type; null means bottom (⊥). In general, actors' inputs are not commutative (i.e., their order matters, as denoted by the numbers on their input arcs.) Future versions will also allow actors to have a varying number of input concepts, when the meaning would be clear (e.g., **`plus`** could have two or more numbers to be added).

Compatibility Note: The actors whose names contain "1_0" are numeric versions of the logical actors returning T or null. They are by default disabled. You can enable them using the Preferences Panel.

	Input Concepts		**Output Concepts**		
Actor Name	**Number**	**Type(s)**	**Number**	**Type(s)**	**Semantics**
`copy`	One	`T`	One	`T`	Input (referent only) is copied to output concept.
`dbfind` `lookup`	Two	`Database` `T`	One	`Number`	Output concept is the value associated with `T`'s referent in the file denoted by the `Database` concept.
`divide`	Two	`Number`	One	`Number`	Output number is concept 1 divided by concept 2.
`displaybar`	One	`Number`	None		Input number is displayed as a bar in a separate window.
`equal`	Two	`T`	One	`T or null`	Output type is `T` if inputs are strictly equal; otherwise `null`
`equal_1_0`	Two	`T`	One	`1 or 0`	Output referent is 1 if inputs are strictly equal; otherwise 0
`exp`	One	`Number`	One	`Number`	Output number is input's referent raised to the power 2.718….
`greaterequal`	Two	`T`	One	`T or null`	Output type is `T` if input 1 greater than or equal to input 2; otherwise `null`
`greaterequal_1_0`	Two	`T`	One	`1 or 0`	Output referent is 1 if input 1 greater than or equal to input 2; otherwise 0
`greaterthan`	Two	`T`	One	`T or null`	Output type is `T` if input 1 greater than input 2; otherwise `null`
`greaterthan_1_0`	Two	`T`	One	`1 or 0`	Output referent is 1 if input 1 greater than input 2; otherwise 0
`lessequal`	Two	`T`	One	`T or null`	Output type is `T` if input 1 less than or equal to input 2; otherwise `null`
`lessequal_1_0`	Two	`T`	One	`1 or 0`	Output referent is 1 if input 1 less than or equal to input 2; otherwise 0
`lessthan`	Two	`T`	One	`T or null`	Output type is `T` if input 1 less than input 2; otherwise `null`
`lessthan_1_0`	Two	`T`	One	`1 or 0`	Output referent is 1 if input 1 less than input 2; otherwise 0
`minus`	Two	`Number`	One	`Number`	Output number is concept 1 minus concept 2.
`multiply`	Two	`Number`	One	`Number`	Output number is product of the two inputs. Commutative.
`notequal`	Two	`T`	One	`T or null`	Output type is `T` if inputs are strictly not equal; otherwise `null`
`notequal_1_0`	Two	`T`	One	`1 or 0`	Output referent is 1 if inputs are strictly not equal; otherwise 0
`plus`	Two	`Number`	One	`Number`	Output number is sum of the two inputs. Commutative.

There is as yet no actor-definition mechanism; that is a future enhancement. (Suggestions for appropriate mechanisms are welcome.)

An actor plug-in interface has been developed as of CharGer 2.6b or later. See the technical reference below for details on how to write your own actors for ***CharGer***.

Database Linking

One of ***CharGer***'s features is its ability to use an external "database" that actor <lookup> can use. This ability is built into ***CharGer***'s actor definitions. For ***CharGer***'s purposes (as of the current version) a database file is a tab-separated tabular text file. For example, suppose the file DBElement.txt contains the following tab-separated values:

Number	Element	Symbol
1	Hydrogen	H
2	Helium	He
3	Lithium	Li

CharGer's **`<dbfind>`** or **`<lookup>`** actor is designed to illustrate ***CharGer***'s interface to a database. The following graph shows how such a database would be used in a conceptual graph with actors:

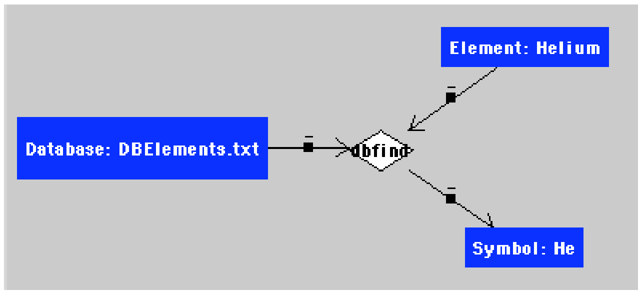


Figure 0-1. Illustration of the <dbfind> and <lookup> actor.

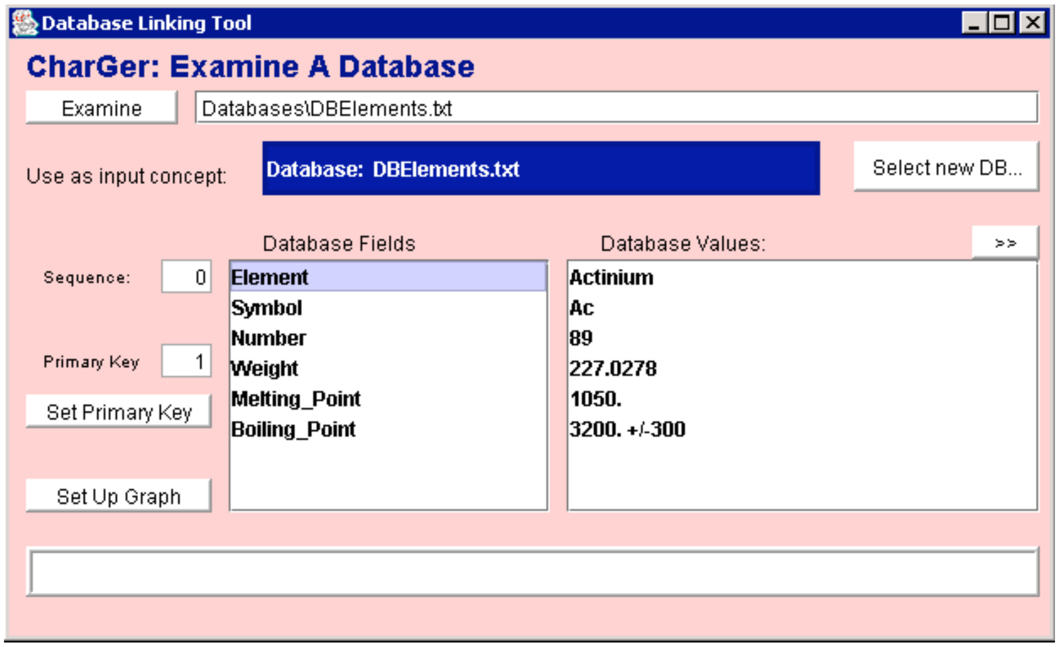
To see how the actor works, use the "T" tool to change "Helium" to "Lithium". Note how the Symbol and Number concepts now have new referents! Another example is to change the Element referent to "Earth" (which is not an element) and note how both Symbol and Number become null (which is equivalent to ⊥ in conceptual graph terms).

There are a few important guidelines for how the **`lookup`** actor works. First, it requires an input concept of type **`Database`** whose referent is a real file name. Second, another input must have a type which exactly matches some field type (i.e., a header name from the file). Third, there must be a single output concept, whose type also matches some field type. (It's permitted for the input and output types to be the same; it's a good way to see whether the input concept's referent value is actually found in the database.) Use the **Database Linking Tool** to see what type names are valid in a given database.

A database with no tab characters will be treated as a one-column table whose entire first line is considered the only valid field name.

Restrictions: At present all databases must be in the same folder; the default is the **`Databases`** folder under the "top-level" folder; but this may be changed in the Preferences panel. Other restrictions probably exist regarding spaces in field names and things like that. There is also no type checking performed with actual values from the database; i.e., if the value of field **`Number`** is not a number, **`lookup`** won't care.

To make it easier for users to create usable graphs with database actors, the **`Database Linking Tool`** window has been provided. It is accessed through the **`Tool`** menu in the Main Window. When activated, the window looks something like Figure 0-2:



Databases\DBElements.txt

Database: DBElements.txt

Sequence: 0

Primary Key 1

Database Fields	Database Values:
Element	Actinium
Symbol	Ac
Number	89
Weight	227.0278
Melting_Point	1050.
Boiling_Point	3200. +/-300

Figure 0-2. Database Linking Tool Window.

This window helps the user determine what are valid inputs and outputs to the database lookup actor(s). The window can also be used to set up a template graph for defining a database's semantics (with or without a primary key).

Examine: will open the "database" file whose name is shown. The first line of the file must be a header consisting of a series of two or more tab-separated strings, each of which is the field name corresponding to the subsequent lines in the file. At present, ***CharGer*** only works with text-only, tab-separated lines.

Select new DB… will open a file dialog giving the user a chance to pick a database file. The database file must reside in the chosen Databases folder; ***CharGer***'s default, or a folder selected by the user. This is a known limitation to be rectified in the future.

Note: Regardless of the folder one has reached through the file dialog, it will be ignored and the folder "Database" substituted.

Use as input concept indicates the valid input concept representing the database, to be used as an input to a <lookup> actor for effective lookup operations using this database.

Database Fields contains the exact names of the columns in the database file. These names are to be used as type-labels for the input concepts to the lookup actor. Referent values for such actors can be any value for the type. For example, [Number: 5] would be a valid input concept for the fields given in the example screen. Selecting one of the database fields puts its sequence number into the **Sequence** box.

Database Values contains the values, for a given record (line) in the database file. The "**>>**" button skips to the next line. This lets the user confirm that the database is well formed and that the field names correspond to the desired values. There is no way to skip backward.

Set Primary Key makes the selected field name into the database file's primary key. When one of the field names is selected, show index will give its sequence number in the list. This is the same sequence number that will be used for the primary key sequence number.

Set Up Graph creates a new graph, in an editing window, with the database concept as input to a set of <lookup> actors, a primary key set up as the other input (if a primary key has been selected), so that the user can begin a graph already connected to a database. This is a handy tool for describing database semantics in a conceptual graph, and checking the semantics of the graph using actual values in a database.

CRAFT Subsystem

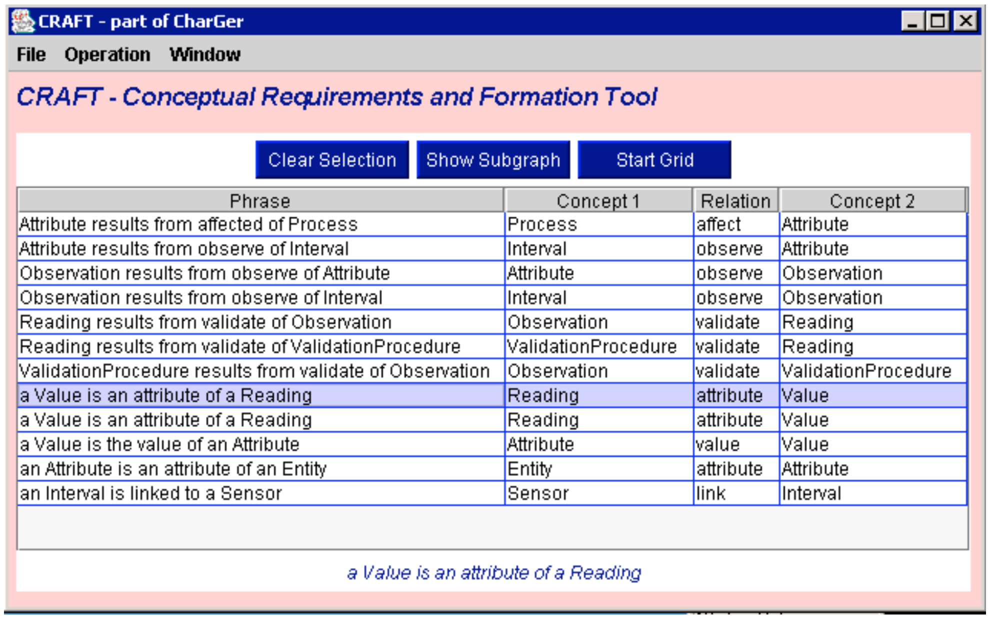
As of CharGer 3.3b, CharGer has been extended to include a repertory grid acquisition interface called the Conceptual Requirements Acquisition and Formation Tool (CRAFT). This tool allows repertory grids to be acquired based on an underlying conceptual graph. The workflow supported is as follows:

- Create one or more conceptual graphs with generic concepts representing parts of the domain you want to model.
- Make sure the graphs you want are opened in CharGer.
- If you want to include a type hierarchy, have exactly one CharGer window with type labels in it. These can be included with other graphs if you want.
- Invoke the CRAFT main window. It should display a list of phrases with their associated concepts and relationships.
- Choose a line and invoke "Start Grid". A new blank grid will open and a simple acquisition dialog will proceed.

There are two relevant windows and a glossary dialog box associated with CRAFT. They are explained in this section.

CRAFT main window

The CRAFT main window looks like this:



Phrase	Concept 1	Relation	Concept 2
Attribute results from affected of Process	Process	affect	Attribute
Attribute results from observe of Interval	Interval	observe	Attribute
Observation results from observe of Attribute	Attribute	observe	Observation
Observation results from observe of Interval	Interval	observe	Observation
Reading results from validate of Observation	Observation	validate	Reading
Reading results from validate of ValidationProcedure	ValidationProcedure	validate	Reading
ValidationProcedure results from validate of Observation	Observation	validate	ValidationProcedure
a Value is an attribute of a Reading	Reading	attribute	Value
a Value is an attribute of a Reading	Reading	attribute	Value
a Value is the value of an Attribute	Attribute	value	Value
an Attribute is an attribute of an Entity	Entity	attribute	Attribute
an Interval is linked to a Sensor	Sensor	link	Interval

a Value is an attribute of a Reading

The main window has the following buttons. Buttons are not shown unless they can be invoked.

Clear Selection

If a row is selected, then un-select it. Selecting more than one row is not recommended and probably will have no positive effect.

Show Subgraph

Each phrase is based on a relationship in a conceptual graph in an open window. This button reveals the concepts and relations for the selected phrase by showing a selection box around them wherever they are present.

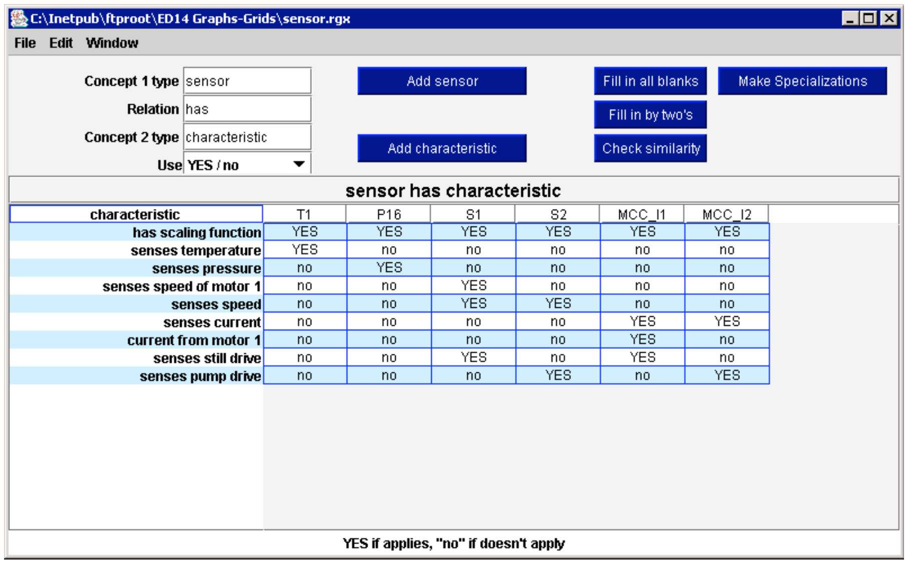
Start Grid

Based on the selection, start a new repertory grid and begin acquiring rows and columns for the grid using the Grid Window.

The Grid Window

The repertory grid window is used to acquire and summarize knowledge through a primitive repertory grid elicitation process. Consider this an experimental capability. The acquisition facility will acquire instances and attributes of whatever relation was chosen in the CRAFT window (see above). Choosing a relation in the CRAFT window selects a concept-relation-concept triple that will form the basis for the grid. **`Show Subgraph`** will remind the user of what relationship he/she chose.

Note that not all buttons will appear on the window at any given time; the buttons are set to appear only when they are applicable.



Concept 1 type: sensor
Relation: has
Concept 2 type: characteristic
Use: YES / no

sensor has characteristic

characteristic	T1	P16	S1	S2	MCC_I1	MCC_I2
has scaling function	YES	YES	YES	YES	YES	YES
senses temperature	YES	no	no	no	no	no
senses pressure	no	YES	no	no	no	no
senses speed of motor 1	no	no	YES	no	no	no
senses speed	no	no	YES	YES	no	no
senses current	no	no	no	no	YES	YES
current from motor 1	no	no	no	no	YES	no
senses still drive	no	no	YES	no	YES	no
senses pump drive	no	no	no	YES	no	YES

YES if applies, "no" if doesn't apply

Concept 1 type

This is the type label for the first concept in the relation. This label is filled in from the original concept-relation-concept triple.

Relation

This is the relation label for the first concept in the relation. This label is filled in from the original concept-relation-concept triple.

Concept 2 type

This is the type label for the second concept in the relation. This label is filled in from the original concept-relation-concept triple.

Add (instance of concept 1)

Using the type label of concept 1, this button will allow you to add an instance of that concept to the grid as a column label. You may change it later by double-clicking on the column label.

Show Subgraph

Takes the user back to the original conceptual graph and highlights the concept-relation-concept subgraph which forms the basis for this grid.

Add (instance of concept 2)

Using the type label of concept 2, this button will allow you to add an instance of that concept to the grid as a column label. You may change it later by double-clicking on the column label.

Fill in all blanks

If there are any blank entries in the grid (cells), then this button will start a filling-in process that will consider each of the one by one, asking a grid question that should assist the user in providing an answer.

Fill in by two's

Use a simplified dyadic elicitation technique to acquire new instances and attributes.

Check similarity

Look for columns with the same set of attributes. This will automatically invoke an acquisition process to acquire some new attribute(s) where the instances in those columns are different.

Make specializations

Known Bugs and Restrictions

Most of these are meant to be handled in future versions. In the meantime, I hope you find ***CharGer*** to be useful in some way. Please let me know (delugach@cs.uah.edu) about other bugs.

Known Bugs

- Undo does not work within a text field, although an entire editing operation can be undone once it is completed.
- Some CGIF files may cause a non-fatal, but perplexing Java parser error.

Restrictions

- Moving (or attempting to move) parts of a graph out of the current drawing area is not supported. Auto-scrolling is not supported.
- Actors do not yet operate with contexts as input or output, although such input and output contexts can be drawn. Inputs and outputs for actors must be simple concepts. The concepts themselves may be nested in contexts.
- The **`lookup`** actor uses databases that must all be located in the same folder; this is the folder named **`Databases`** within the top-level CharGerXX folder, or you may select a different one in the Actor Settings panel of the Preferences window. There is no pathname in the **`Database`** input concept to **`lookup`**. It is possible to work around this with aliases, links or shortcuts. It may be possible to play around with the actual referent of a **`Database`** concept to include a path, but I wouldn't rely on it, especially if graphs are to be used on more than one platform.
- The bottom symbol, ⊥ is represented by the type or referent **`null`** in referents or concepts. There is no provision yet for a ∀ or ∃ symbol in a referent. The vertical bar "|" or space " " is illegal in a type or referent.
- In general, text in ***CharGer*** is case-sensitive, meaning that strings are compared "as is". The only exception is that when dealing with filenames, case sensitivity may depend on the conventions of the underlying platform operating system.
- Actors can only be activated by editing one of their input or output concept referents, or the actor name itself. There is no explicit activation of the actor itself.
- Lambda expressions are not yet supported.
- Arithmetic with real numbers lacks precision; ***CharGer*** is not a reliable calculator for real numbers.
- When moving a context, all its contents go with it logically; if the move causes additional graph elements to appear enclosed visually, those additional elements are not logically part of the context! This issue should be addressed in a future release.
- Type and relation labels (e.g., in a hierarchy) are NOT exported in the CGIF form except insofar as they are passed as type labels of concepts, contexts and relations.

- A concept can safely be a member of only one coreference set. It is not yet clear to this author (and others) how to interpret the semantics of a concept that's a member of more than one coreference set.
- When joining or matching forms a new graph, elements of the new graph may overlap, since each set of elements is derived from a separate graph. The graph is stored internally in its correct form (as would be displayed by the CGIF format).
- Matching is not completely implemented. That is, several combinations and/or matching parameters will either not work at all or produce unpredictable results. The matching in ***CharGer*** is provided by Notio, which is currently undergoing additional development.
- Graph modality labels are for convenience only; no operational difference occurs in ***CharGer***. (They can be turned on and off in the Compatibility panel of the Preferences window.)
- LINUX only: running ***CharGer*** will often show errors such as "cannot convert string …. To type VirtualBinding". These are really Motif errors and can be safely ignored. If you are interested in the cause of the errors, see Google's newsgroups and search on "cannot convert string" VirtualBinding and you'll find out more.

Frequently Asked Questions

What does ■ mean on the linking lines?

Linking lines in conceptual graphs can be labeled. Generally relation arrows are to be numbered, although in many cases that's not necessary, since the types which are linked will serve to distinguish the arrows. The ■ is a selection handle merely for convenience in selecting a line for editing its label or deleting. If you click on the dot when deleting, then the arrow is deleted. If you click on the dot when you are editing text, you can change the relation label. The handle does not appear when printing.

To make the handles invisible (and make all lines unable to be selected!) uncheck the **Show line selection handle** option on the Preferences Panel.

How do I select a context?

Click somewhere on the *border* of the context. This is also how you select the context's label for editing. This allows moving the context, deleting the context or editing its name, just as you would any other graph component.

How do I change the text label in a context?

Double-click somewhere on the *border* of the context. The context name should appear in an editing field and you can change it.

How do I delete an arrow?

Choose the Delete Tool and then click on the arrow's handle. If there is no handle showing, be sure to check the **Show line selection handle** option on the Preferences Panel.

How do I delete a concept/relation without deleting its linking arrows?

You can't. ***CharGer*** cannot have a line unless there is an element at both of its ends. Deleting or moving a concept/relation also deletes or moves its lines.

How do I re-size a concept/actor/relation node?

There is no explicit way to just re-size a node. ***CharGer*** chooses the size of a node automatically, based on its text label, fonts and some additional cosmetic considerations. A context is expanded to enclose its contents. The "+" and "–" keyboard hotkeys will enlarge or reduce one or more selected nodes one pixel at a time. Holding down the SHIFT key while pressing "+" or "–" will enlarge or reduce the selected nodes more quickly. The "Shrink Selection" menu item will also change the size of nodes and contexts. A context can be expanded by moving its contents out toward its edges – the context border will expand to fully enclose them.

I opened a CGIF/CGX graph file and all the objects are crowded into the top left corner. Why?

The saved CGIF/CGX graph did not have embedded ***CharGer*** layout information in its CGIF comments. To activate this feature, see the Preferences Panel "Include ***CharGer*** Info in CGIF". Without this information, ***CharGer*** is unable to draw the graph on the screen, although its contents (semantics) should be preserved.

How do I change a regular (non-negated) context to a “cut” and vice versa?

There is no way to do this directly. The only way to reverse the sense is to un-make the context/cut and then re-make it as a cut/context.

Is it possible to select all the concepts in a drawing and change the text color in all of them at once?

There is no way to select just the concepts or just the relations in a graph, but you can select everything. Do Edit->Select All and then Edit->Change Color->Text. This brings up a color palette that will let you choose the new color. Note that if the fill and text colors are the same, the text will seem to “disappear”.

Technical Reference

CharGer XML File Format (version 3.1b and later)

As of version 3.1b, CharGer’s stored graphs are in an XML format. This is a preliminary format, which unfortunately may change, but it serves as a starting point for others to develop compatible applications. Graphs in this form are suffixed with “.cgx” to distinguish them from the earlier versions.

The syntax of XML is beyond the scope of this manual. A DTD may be forthcoming, but in the meantime, this section describes the XML syntax.

The meaning of a “graph” in CharGer is a set of (possible unconnected) CharGer nodes, some of which may have links between them. The links are stored separately in the XML file.

The formal grammar for the file is as follows. Note that there must be space between characters unless they are special characters and that all parameter values within a tag must have double quotes around them. See general XML documentation for the syntax

```
CharGerXMLfile ::= xmlheader "<conceptualgraph" cgparmList ">"
                   cgElementList "</conceptualgraph>"
xmlheader ::= "<?xml version="1.0" encoding="UTF-8"?>"
cgparmList ::= empty | cgparm | cgparm cgparmList
cgparm ::= "creator=" string | "version=" string | "created=" datestring
cgElementList ::= empty | cgElement | cgElement cgElementList
cgElement ::= cgNode | cgGraph | cgEdge
cgGraph ::= "<graph" objectParmList ">" objectTagList cgElementList
"</graph>"
cgNode ::= "<" cgNodeName objectParmList ">" objectTagList "</"cgNodeName ">"
cgLine ::= "<" cgLineName objectParmList edgeParmList ">" objectTagList
                                    "</"cgLineName ">"
cgNodeName ::= "concept" | "relation" | "actor" | "typelabel" |
                                "relationlabel" |
cgEdgeName ::= "arrow" | "genspeclink" | "coref"
objectParmList ::= empty | objectParm | objectParm  objectParmList
objectParm ::= "id=" idstring | "owner=" idstring | "label=" string |
"negated=" trueorfalse
trueorfalse ::= "true" | "false"
edgeParmList ::= empty | edgeParm | edgeParm edgeParmList
edgeParm ::= "from=" idstring | "to=" idstring
idstring ::= string | "0"
objectTagList ::= empty | objectTag | objectTag objectTagList
objectTag ::= typeTag | referentTag | layoutTag
typeTag ::= "<type>" typePartList "</type>"
typePartList ::= typePart | typePart typePartList
typePart ::= label | typeDescriptor
referentTag ::= "<referent>" referentPartList "</referent>"
referentPartList::= referentPart | referentPart referentPartList
referentPart::= label | referentDescriptor
referentDescriptor ::= string   /* Note: currently undefined */
label ::= "<label>" string "</label>"
typeDescriptorPartList ::= typeDescriptorPart | typeDescriptorPart
typeDescriptorPartList
typeDescriptor ::= WordnetDescriptor | GenericDescriptor
WordnetDescriptor ::= "<wordnet" wordnetParms "/>"
WordnetParms ::= "version=" version "pos=" partofspeech "offset="
uniqueoffset
partofspeech ::= "noun" | "verb" | "adjective" | "adverb"
uniqueoffset ::= positiveNumber
genericDescriptor ::= "<generic" genericParms ">" typeDescriptorPartList
"</generic>"
genericParms ::= "pos=" partofspeech "definition=" string
layoutTag ::= "<layout>" layoutPartList "</layout>"
layoutPartList ::= empty | layoutPart | layoutPart layoutPartList
layoutPart ::= rectangle | color
```

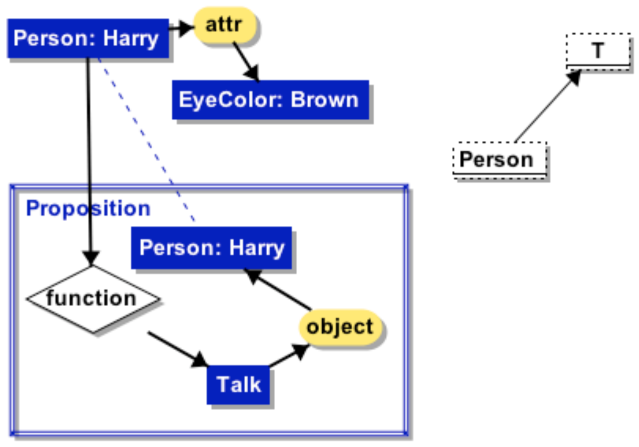
```
rectangle ::= "<rectangle" "x=" int "y=" int "width=" int "height=" int "/>"
color ::= "<color" "foreground=" rgb "background=" rgb "/>"
rgb ::= int "," int "," int
```

There are certain constraints to make a well-formed graph for the CharGer parser, mostly due to the fact that it's a one-pass parser.

- All owner objects must appear before any "owner=" parameters.
- An edge must appear after both of its endpoint objects have been listed (this can be a bit tricky with respect to contexts and lines of identity)
- If a parameter appears more than once, the last one will generally be accepted

Here is an example graph, whose file is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<conceptualgraph creator="CharGer"
      version="3.2b" created="Jun 19, 2003 11:06:20 PM">
<graph id="109" owner="0" >
    <type>
        <label>Proposition</label>
    </type>
    <layout>
        <rectangle x="0" y="0" width="50" height="30"/>
        <color foreground="0,0,175" background="0,0,175"/>
    </layout>
    <relation id="114" owner="109" >
        <type>
            <label>attr</label>
        </type>
        <layout>
            <rectangle x="118" y="66" width="40" height="23"/>
            <color foreground="0,0,0" background="255,231,100"/>
        </layout>
    </relation>
    <concept id="113" owner="109" >
        <type>
            <label>Person</label>
        </type>
        <referent>
            <label>Harry</label>
        </referent>
        <layout>
            <rectangle x="-1" y="74" width="104" height="25"/>
            <color foreground="255,255,255" background="0,0,175"/>
        </layout>
    </concept>
    <concept id="112" owner="109" >
        <type>
            <label>EyeColor</label>
        </type>
        <referent>
            <label>Brown</label>
        </referent>
        <layout>
            <rectangle x="105" y="113" width="125" height="25"/>
            <color foreground="255,255,255" background="0,0,175"/>
        </layout>
    </concept>
    <typelabel id="111" owner="109" >
        <type>
            <label>T</label>
        </type>
        <layout>
            <rectangle x="355" y="83" width="40" height="23"/>
            <color foreground="0,0,0" background="255,255,255"/>
        </layout>
    </typelabel>
    <typelabel id="110" owner="109" >
        <type>
            <label>Person</label>
        </type>
        <layout>
            <rectangle x="283" y="152" width="59" height="23"/>
            <color foreground="0,0,0" background="255,255,255"/>
        </layout>
    </typelabel>
    <graph id="115" owner="109" >
        <type>
            <label>Proposition</label>
        </type>
        <layout>
            <rectangle x="3" y="180" width="250" height="157"/>
            <color foreground="0,0,175" background="0,0,175"/>
        </layout>
        <relation id="119" owner="115" >
            <type>
                <label>object</label>
            </type>
            <layout>
                <rectangle x="185" y="258" width="53" height="23"/>
                <color foreground="0,0,0" background="255,231,100"/>
            </layout>
        </relation>
        <actor id="118" owner="115" >
            <type>
                <label>function</label>
            </type>
            <layout>
                <rectangle x="12" y="230" width="85" height="43"/>
                <color foreground="0,0,0" background="255,255,255"/>
            </layout>
        </actor>
        <concept id="117" owner="115" >
            <type>
                <label>Person</label>
            </type>
            <referent>
                <label>Harry</label>
            </referent>
            <layout>
                <rectangle x="79" y="206" width="102" height="27"/>
                <color foreground="255,255,255" background="0,0,175"/>
            </layout>
        </concept>
        <concept id="116" owner="115" >
            <type>
                <label>Talk</label>
            </type>
            <layout>
                <rectangle x="127" y="294" width="40" height="25"/>
                <color foreground="255,255,255" background="0,0,175"/>
            </layout>
        </concept>
        <arrow id="120" owner="115" label="-" from="119" to="117">
            <layout>
                <rectangle x="169" y="243" width="6" height="6"/>
                <color foreground="0,0,0" background="255,255,255"/>
            </layout>
        </arrow>
        <arrow id="122" owner="115" label="-" from="116" to="119">
            <layout>
                <rectangle x="176" y="285" width="6" height="6"/>
                <color foreground="0,0,0" background="255,255,255"/>
            </layout>
        </arrow>
        <arrow id="121" owner="115" label="-" from="118" to="116">
            <layout>
                <rectangle x="106" y="281" width="6" height="6"/>
                <color foreground="0,0,0" background="255,255,255"/>
            </layout>
        </arrow>
    </graph>
    <arrow id="127" owner="109" label="-" from="113" to="118">
        <layout>
            <rectangle x="49" y="162" width="6" height="6"/>
            <color foreground="0,0,0" background="255,255,255"/>
        </layout>
    </arrow>
    <arrow id="126" owner="109" label="-" from="114" to="112">
        <layout>
            <rectangle x="149" y="98" width="6" height="6"/>
            <color foreground="0,0,0" background="255,255,255"/>
        </layout>
    </arrow>
    <arrow id="125" owner="109" label="-" from="113" to="114">
        <layout>
            <rectangle x="108" y="77" width="6" height="6"/>
            <color foreground="0,0,0" background="255,255,255"/>
        </layout>
    </arrow>
    <genspeclink id="124" owner="109" label="-" from="110" to="111">
        <layout>
            <rectangle x="340" y="126" width="6" height="6"/>
            <color foreground="0,0,0" background="255,255,255"/>
        </layout>
    </genspeclink>
    <coref id="123" owner="109" label="-" from="113" to="117">
        <layout>
            <rectangle x="87" y="150" width="6" height="6"/>
            <color foreground="0,0,175" background="0,0,175"/>
        </layout>
    </coref>
</graph>
</conceptualgraph>
```



Repertory Grid Proprietary File Format (v. 3.2b and later)

```
RepGridXMLfile ::= xmlheader "<repertorygrid>" rgheader rgattrList
                  rgelemList rgcellList "</repertorygrid >"
xmlheader ::= "<?xml version="1.0" encoding="UTF-8"?>"
rgheader ::= "<header>" headerLabelList "</header>"
headerLabelList ::= empty | headerLabel | headerLabel headerLabelList
headerLabel ::= "<headerlabel>"
cgparmList ::= empty | cgparm | cgparm cgparmList
cgparm ::= "creator=" string | "version=" string | "created=" datestring
cgElementList ::= empty | cgElement | cgElement cgElementList
cgElement ::= cgNode | cgGraph | cgEdge
```

CharGer Proprietary File Format (version 3.0b only)

See the .cg files to see what they look like. Edit them at your own risk! Here is a brief specification of the format. An example file is “catonmat.fact.cg” which looks like this:

```
CharGer|version=3.0b|creation=Feb 25, 2003 7:42:06 PM
Graph|17,0|Proposition|0,0,1000,1007|0,0,175|0,0,175
Concept|22,17|SIT|312,123,35,25|255,255,255|0,0,175
Relation|21,17|agent|203,224,55,25|0,0,0|255,231,100
Concept|20,17|Cat: Axel|63,123,113,25|255,255,255|0,0,175
Relation|19,17|location|385,243,80,25|0,0,0|255,231,100
Concept|23,17|MAT|488,123,45,25|255,255,255|0,0,175
Arrow|27,17|-|374,193,6,6|0,0,0|0,0,0|22,19
Arrow|26,17|-|171,183,6,6|0,0,0|0,0,0|21,20
Arrow|25,17|-|276,183,6,6|0,0,0|0,0,0|22,21
Arrow|24,17|-|464,193,6,6|0,0,0|0,0,0|19,23
\\
```

The formal grammar of the file is as follows:

The first line is of the form **`CharGer|version=<versionnumber>|creation=<date>`**

Succeeding lines are of the form

`<ObjectType>|<ID>,<EnclosingID>|<TextLabel>|<Rectangle>|<textColor>|<fillColor>|<fromtoID>`

- <ObjectType> Valid object types are: **`Graph`**, **`Concept`**, **`Relation`**, **`Actor`**, **`TypeLabel`**, **`RelationLabel`**, **`Arrow`**, **`Coref`**, and **`GenSpecLink`**. Spelling and capitalization must match exactly.
- <ID>,<EnclosingID> The first number <ID> is the unique identifier for this graph object. No two objects can have the same one in the same file. The second number <EnclosingID> is the unique identifier number for that object's "owner". In the 5th line of the example, Concept number 20, which is [Cat:Axel], is enclosed by Graph numbered 17. Note that Graph 17 itself has owner zero; that indicates this is the top-level graph in the file. If a context appears, it will have the type "Graph" and its owner will be whatever graph logically encloses it. There is no explicit provision for a graph to be a referent; its name constitutes its referent designator.
- <TextLabel> the text label for the object. For the top-level graph, it is often convenient to use the file's name, although that is not required. Otherwise any text (except for a vertical bar and a newline) may appear in this term. Connecting lines may have a label, if so, it appears here.
- <Rectangle> the object's display rectangle. Four numbers are required. They denote in order the upperleft corner's x-coordinate, upper-left corner's y-coordinate, width, and height. The coordinate system used is Java's, where x increases going to the right and y increases going down.
- <textColor> a set of three integers representiing the RGB color value of this object's text. No alpha channel is supported for colors.
- <fillColor> a set of three integers representiing the RGB color value of this object's background fill. No alpha channel is supported for colors.
- For nodes in a graph (Graph, Concept, Relation, Actor and CGType) only five terms are present. For connecting lines in a graph (Arrow, Coref and GenSpecLink), a sixth term is present. It consists of two numbers, the first denoting the unique identifier of the source node, the second denoting the destination node. In the example, Arrow with ID 26 goes from ID 20 to ID 21, which means the arrow goes from the Relation "agent" to the Concept "Cat: Axel".

The file must be terminated by two backslashes "\\" on its own line.

CharGer Proprietary File Format (versions 2.6b or earlier)

CharGer can still read the old files (setting colors to the current default color scheme), but it can no longer save them in that form. All .cg files will be saved in the new format (see above). See the (old) .cg files to see what they look like. Edit them at your own risk! Here is a brief specification of the format. An example file is "catonmat.fact.cg" which looks like this:

```
Graph|12,0|catonmat.fact.cg|0,0,1000,1000|0,0,1000,1000
Concept|18,12|Cat: Albert|81,158,100,25|81,158,100,25
Concept|17,12|SIT|305,158,40,25|305,158,40,25
Concept|16,12|MAT|467,158,40,25|467,158,40,25
Relation|15,12|agent|191,263,56,18|191,263,56,18
Relation|14,12|location|357,263,77,18|357,263,77,18
Arrow|20,12|-|403,183,72,80|436,220,6,6|14,16
Arrow|19,12|-|141,183,70,80|173,220,6,6|15,18
```

```
Arrow|22,12|-|228,183,84,80|267,220,6,6|17,15
Arrow|21,12|-|333,183,55,80|357,220,6,6|17,14
\\
```

Each object in the graph is specified by a single line. Terms on each line are separated by a vertical bar.

- The first term is the graph object type. Valid types are: **`Graph`**, **`Concept`**, **`Relation`**, **`Actor`**, **`CGType`**, **`Arrow`**, **`Coref`**, and **`GenSpecLink`**. Spelling and capitalization must match exactly.
- The second term consists of a pair of numbers. The first number is the unique identifier for this graph object. No two objects can have the same one in the same file. The second number is the unique identifier number for that object's "owner". In the 2nd line of the example, Concept number 18, which is [Cat:Albert], is owned by Graph 12. Note that Graph 12 has owner zero; that indicates this is the top-level graph in the file. If a context appears, it will have the type "Graph" and its owner will be whatever graph logically encloses it. There is no explicit provision for a graph to be a referent; its name constitutes its referent designator.
- The third term is the text label for the object. For the top-level graph, it is often convenient to use the file's name, although that is not required. Otherwise any text (except for a vertical bar and a newline) may appear in this term. Connecting lines may have a label, if so, it appears here.
- The fourth term is the object's display rectangle. Four numbers are required. They denote in order the upperleft corner's x-coordinate, upper-left corner's y-coordinate, width, and height. The coordinate system used is Java's, where x increases going to the right and y increases going down.
- The fifth term is a relative rectangle, currently not used. Make it the same as the fourth term.
- For nodes in a graph (Graph, Concept, Relation, Actor and CGType) only five terms are present. For connecting lines in a graph (Arrow, Coref and GenSpecLink), a sixth term is present. It consists of two numbers, the first denoting the unique identifier of the source node, the second denoting the destination node. In the example, there is an arrow which goes from ID 15 to ID 18, which means the arrow goes from the Relation "agent" to the Concept "Cat: Albert".

The file must be terminated by two backslashes "\\" on their own line

Development Details

For any developers who are interested, the editor up to version 2.6b was developed under Metrowerks' CodeWarrior for the Macintosh. Since then, development has proceeded under Mac OS X using Project Builder and now Xcode. CharGer 3.4b consists of 120 Java classes, which make up approximately 47,000 lines of code (including comments). A description of these classes can be found at http://www.cs.uah.edu/~delugach/CharGer/Docs.

The Java console (i.e., the command line window) may display messages from time to time. In general, users can safely ignore them. If errors occur, the console may have information that can be useful in figuring out that there's a bug to tell me about.

Invoking *CharGer* from an application

If you have your own Java application (or possibly some other language's application) you may invoke ***CharGer*** quite easily, I think. The steps should be as follows:

- Make sure that the `charger` package and the `notio` packages are included in your Java project or environment. This will depend on the programming environment you use. In Metrowerks' CodeWarrior, you should add the `CharGer.jar` file to your project.
- In your Java program, invoke the following call.

  ```
  charger.Hub.setup();
  ```
- In the places where the driver needs to exit, the call

  ```
  charger.Hub.closeOutAll();
  ```

That's all I had to do in my own main class. You may encounter problems; if so, report them and I'll do my best to figure them out.

In the future, interface calls will be provided to allow users to construct graphs in other programs and pass them into ***CharGer*** for editing.

Actor Plugin Interface

Starting with CharGer 2.6b, there's an actor plugin architecture whereby you can write your own actors that ***CharGer*** will incorporate. Of course, you're responsible for the reliability, etc. of the actors you write. If you want me to include them in the release (once you've tested them thoroughly!) send them to me for consideration.

The plugin interface provides a mechanism for creating external actors that can be incorporated into ***CharGer***. Java classes that implement this interface are allowed to "plug in" to ***CharGer*** and show up in the actor list just as the primitive actors are. Classes implementing the ActorPlugin interface must appear in a package named "`plugin`". Responsibility for extracting referents from the `charger.Concept` arguments lies with the class implementing the interface; some convenience methods are found in `GraphUpdater`.

```
package charger;

public interface ActorPlugin
{
/**
    @return name by which the actor will be known throughout the system; this is the
string
        that will be used as the label on an actor in CharGer.
    */
    abstract public  String getPluginActorName();

/**
    Assumption about input and output vectors is that inputs are numbered 1..n and
outputs
        are numbered n+1 .. m.
    @return List of input concepts (or graphs), each with a constraining type (or "T" )
*/
    abstract public  Vector getPluginActorInputConceptVector();

/**
    Assumption about input and output vectors is that inputs are numbered 1..n and
outputs
        are numbered n+1 .. m.
```

```
   @return List of output concepts (or graphs), each with a constraining type (or "T"
)
*/
   abstract public Vector getPluginActorOutputConceptVector();

/**
   @return Vector of objects, usually in string form, indicating other actor
constraints.
      Currently only "executable" and "commutative" are supported. Attributes must
include
      "executable" if you want CharGer to activate the actor.
*/
   abstract public Vector getPluginActorAttributes();


/**
   Perform the actor's function. Called by GraphUpdater when input or output concepts
change.
      @param inputs Vector of charger.Concept
      @return outputs Vector of charger.Concept
*/
    abstract public void performActorFunction( Vector inputs, Vector outputs );

/**
   Perform any clean-up required by the actor when it is deleted or its graph
      is de-activated.
*/
   abstract public void stopActor();

/**
   Give a string identifying the author, version, and email address of this plugin
*/
   abstract public String getSourceInfo();
}
```

Bug Fixes

A number of bugs get fixed all the time (just as new ones are introduced!). Here are some recent ones:

Fixed/Changed in v1.8b

- The graph modality “gen” from previous versions has been changed to “generic”. This will mean changing the names of any generic graphs created under previous versions named xxx.gen.cg to xxx.generic.cg.
- The database interface is much improved, although still quite limited. It works only for text databases, i.e., tab-separated text files. See above.
- No part of the system pretends to be an applet anymore.
- Some safeguards are in place to prevent losing an un-saved graph.

Fixed/Changed in 1.9b

- Menu names have been made more consistent with other menu interfaces.
- Editing of arc labels (e.g., “1”, “2”) is handled correctly, particularly with actors.
- Release is no longer in a .jar file, but in classes packaged by folder.

- Some actor activation errors were fixed, and better fault alerts to the user.

Fixed/Changed in 2.0

- Editing a context's name no longer causes the new name to first appear in the center, instead of in the corner where it belongs.
- Simplified the installation by ensuring that the jar file is runnable under JDK1.2.2 without having to unzip all the class files.
- Improved performance on graph re-drawing by using double buffering
- The text edit box for changing labels goes away when you're through with it.
- A preferences panel for setting some preferences within a session.
- Improved appearance in the user interface.
- Ability to set the font in displaying graphs
- Cut/Copy/Paste and Undo have been implemented for both graphs and text.

Fixed/Changed in 2.1

- Improved performance in moving/cutting/copying graph elements
- Both Save and Save As… options for menus
- A brief summary of a graph's contents.
- An indicator on the editor window indicating whether the graph has been changed since its last save.
- Some problems with cross-platform file names were fixed in the main window's graph list.
- An indicator (on the main window) of the memory used and available during execution.
- Arrow-keys are enabled to make small changes in objects' locations on the drawing area.
- Keyboard hotkeys are provided for changing the size of objects.
- Hotkeys for switching between tools.
- Clicking on a window's close button will work correctly under JDK1.2; i.e., "Cancel" (or dismiss window) will abort the close, "Save" will automatically save the graph under its current name, and "Don't Save" will close the graph, discarding any changes since its last save.
- Some preferences can be altered by the user and saved between sessions.

Fixed/Changed in 2.2

- "Make Generic" feature added, to remove referents from selected concepts and contexts.
- Open CGIF capability added. If CGIF file was saved from ***CharGer*** with the ***CharGer*** information embedded in its comments, then ***CharGer*** can re-create the graph from its original, including element positioning.
- Type labels for concepts, relations and actors now conform to the ANSI standard; most invalid characters are converted to "_" (underscore).
- Command line invocation used shorter words.
- Many internal changes made to accommodate keeping a Notio representation internally.

Fixed/Changed in 2.3

- A natural language paraphrase feature, to paraphrase a graph in English (other languages on the way)
- CG operations: match, join on selected concept
- "Unlimited" undo/redo levels
- Adjustable parameters for the matching scheme applied to joins and matching

Fixed/Changed in 2.4

- Coreferent links are correctly imported from CGIF (2.4a04)
- Drawing area automatically enlarges to enclose graph (2.4a02)
- Relation labels are defined similarly to type labels.
- Relation hierarchies use relation labels, rather than tied to actual relations (2.4a03)
- Coreferent links are shown correctly as dashed lines (2.4a03)

Fixed/Changed in 2.5

- Printing has been rewritten and should be more robust and more useful.
- A page setup dialog is now available.
- The “Lib” folder has been removed; its contents are now kept with the Java classes, either in the .jar file or in a class hierarchy if you decide to unpack the .jar file.

Fixed/Changed in 3.0

- The editing window’s appearance has changed.
- Many internal changes (especially bringing ***CharGer*** more into “swing”) have been made.
- The graph “modality” (e.g., schema, definition, etc.) labels are now OFF by default; they can be turned on through the Preferences. The graph modality itself can now be set through a menu in the Edit menu, rather than through its own pop-up menu.
- Added a feature to save a graph in PNG (Portable Network Graphics) format.
- Added a feature to provide actor plugins
- Added a built-in plugin that will display a numeric value in a visual display window by itself.
- Added a “Save All to CGIF” button in the main window that converts all listed graphs to CGIF format, with filenames ending in “.cgif”.
- Added custom colors for all graph objects, and the ability to set one’s own default color scheme.

Fixed/Changed in 3.0.2

- Zoom in and out have been implemented.
- Editing of node labels has changed somewhat to accommodate zooming; editing is now done inside a small independent dialog window on top of the canvas.
- A problem with opening multiple windows was fixed by adding a dialog note telling the user how many files were opened.

Fixed/Changed in 3.3

- The copying of co-referents is now only enabled if actors are enabled in Preferences
- Negation via "cuts" are now supported.
- Improved button icons in the toolbox on the editing window.
- Added the ability to draw "cuts" (i.e., negated contexts)
- Improved robustness when saving graph files
- Concept type should have option of multiple wordnet senses; e.g., "workflow_log"
- Zooming scale shows as a percentage in the top border of the drawing window.
- Handle graphs in knowledge summary.
- A "summarize what is known" option has been implemented, to form a crude English version of everything in open graphs and grids.
- Types can be gathered from graphs and repertory grids and then added automatically to a type hierarchy.
- A Wordnet synset (word-sense) entry can be added to any node (if Wordnet is available)
- RepGrid : if a term (or sub-term) has already had a meaning chosen in a previous query, then suggest it a the default.
- If rep grid element duplicates an existing one, should just use the previous one.
- CharGer can now auto-detect the kind of file being read, whether CG or CGX
- Repertory Grid interface installed through the CRAFT tool window.
    - Repertory grid can be edited and diadic elicitation is supported.
    - Repertory grid can be exported to Burmeister (*.CXT) file
    - Repertory grid can be printed
- Footer option (on by default) for printing graphs and grids.
- "Duplicate" command added to graph edit menu, bypassing clipboard.
- CGX (conceptual graphs in XML) now being used as an internal storage format.
- Trigger and autonomous actors can now be constructed, either through the plugin interface (see manual) or as built-ins.
- Find and find-again commands for node labels has been implemented.
- GraphUpdater updates referents across co-referent links when concept referents change
- Added a color option to set a default black-and-white color scheme.
- Added a display option to draw outlines for all nodes (useful when doing black and white)
- Changed cursor appearance when it's over a context border, since they're so hard to select.

Fixed/Changed in 3.4

- Removed preferences item to set kind of image file being copy/pasted, since it never worked anyway.
- Changed the set of graphics formats available for export (thanks to freehep.org)
    - Vector graphics formats: EPS, PDF, SVG and EMF
    - Raster (bitmap) graphics formats: GIF and PNG
- "lookup" actor (and dbfind) can have any number of output concepts associated with a single lookup key.

- Changed preferences files' suffixes to .PREF so they won't conflict with Windows profile (.PRF) files
- Improved stability during editing and file saving.
- Some built-in graph metrics can be displayed in a text window.
- CharGer now requires Java 1.5 or greater.
- Restricts undo/redo to 10 levels only, due to its memory consumption.
- Handles certain XML-special characters correctly.
- Copy/Paste now lets you paste a graph into another application as an image.

Fixed/Changed in 3.5

- Improved memory efficiency and plugged many memory leaks (thanks to David Phillips!)
- Converted CharGer's graphics to Graphics2D so that positioning now uses floating point valued points instead of int.
- Improved appearance making things a bit sharper and drawing actors and relations better with arcs.
- FindClippingPoint for an actor now takes into account that actors aren't rectangles.
- Provided options for pre-set black-and-white and also grayscale (both in "Change Color.." and on the Preferences panel)
- Changed activation mechanism to account for actors with varying numbers of arguments

Fixed/Changed in 3.6

- CGIF export now compliant with ISO/IEC 24707:2007 Annex B
- Sourceforge web address included, clickable for browsing.