User's Guide

CHARGER Conceptual Graphs

A Conceptual Graph Editor by Harry Delugach

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 deliberately and explicitly a research tool meant for conceptual graph researchers to explore implementation issues in conceptual graph interfaces. For known bugs and limitations, read the list starting on page 28.

Using the software will require 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://www.cs.uah.edu/~delugach/CG.

Features currently supported

These features are currently supported (as of version 2.4b, dated 2000-09-05):

- Save and retrieve graphs from files, in a proprietary (i.e., non-standard) text format. These graphs have the extension .cg and are portable across all platforms.
- Save and retrieve graphs from files in CGIF standard interchange format. These graphs have the extension .CGF and are portable across all platforms.
- Display a graph's CGIF version or a graph's natural language paraphrase, or copy it to the clipboard as text, or save it to a text file.
- 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.
- Auto-save safeguards to prevent losing a graph. Other modifications to prevent losing data.
- Copy/paste of graphs using an internal clipboard
- 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 1.2)
- 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.
- Capability to automatically create a skeleton graph from a database suitable for providing database semantics.
- Hotkeys 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)
- Adjustable parameters for the matching scheme applied to joins and matching

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 28.

- Validation facilities (except for enforcing CG formation rules)
- Ability for user-defined actors.
- Type and relation hierarchies in CGIF form (awaiting final CGIF standard).

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 image 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, at the 1.1.8 level or higher.
- A color monitor, 800 x 600 resolution or higher.

Software

The installation procedure is as follows. You must have a Java package installed to run the editor. A Java package at the JDK1.2 level 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.

To install the software, there are these two steps:

- Obtain the **`CharGer24b.zip`** file.
- Un-zip the **`CharGer24b.zip`** file. It should create three sub folders, **`Lib, Databases`** and **`Graphs`**, which you'll need to run ***CharGer***.

Folder Structure

A simple folder structure is expected when running ***CharGer***. The top-level installation folder CharGerxx will be given the name "**`CharGer24b`**". After installation, the structure should look like Figure 1:

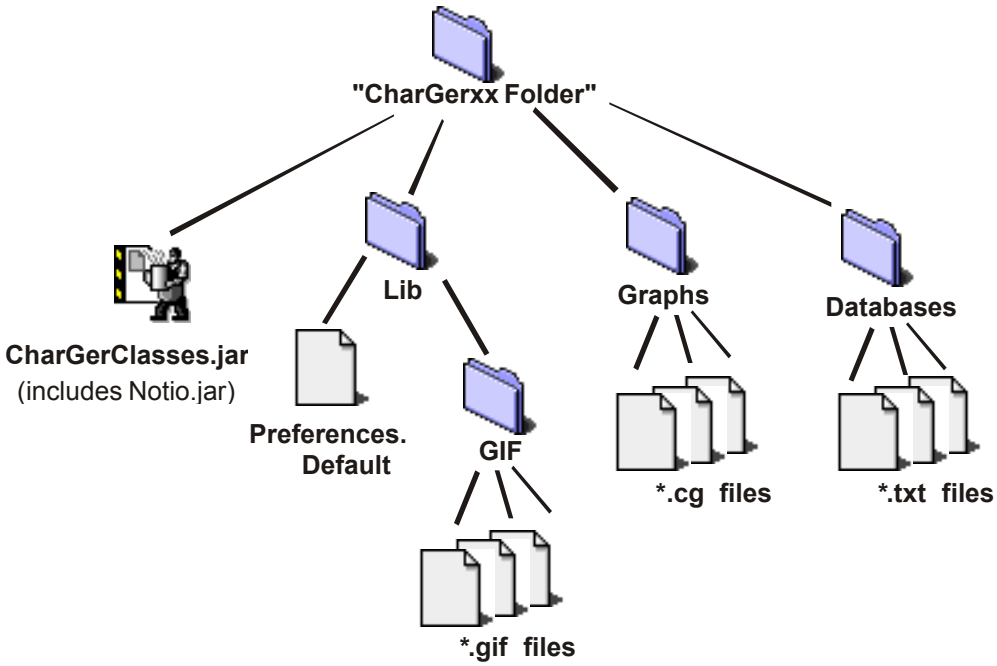


Figure 1. CharGer File and Folder Structure.

A brief explanation of these is as follows:

`CharGerClasses.jar`
Java archive (jar) containing the classes needed to execute ***CharGer***; these are platform independent at the JDK 1. 2 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 the Notio package whose Web site is **`http://backtrack.uwaterloo.ca/CG/projects/notio`**, courtesy of Finnegan Southey (Thanks Finn!)

`Lib/Preferences.Default`
A file containing a number of parameters and options to ***CharGer***.

`Lib/GIF/*.gif` All the pictures and icons ***CharGer*** uses.

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

`Databases` Where ***CharGer*** expects to find its tab-separated text databases for the <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 will have to copy your old graph files to the Graphs folder under the new CharGerxxb folder.

Preferences File

You will probably not need to make too many changes to the Preferences file. A text file called **`Preferences.Default`** must be available in the **`Lib`** folder under the top-level folder. It consists 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 //=

The list of parameters is long and tedious. Look in the **`Preferences.Default`** file released with this prototype for a list and description.

No apologies for the very restrictive syntax!

If a **`Preferences.Default`** file is not found, ***CharGer*** may generate an exception, but otherwise most things will still operate normally on their defaults. A blank line will probably cause an exception too.

The Preferences file should normally be created in its correct location when you unzip the **`CharGer24b.zip`** file in the top-level folder.

CharGer's preferences panel allows the user to save their own set of preferences. These are saved in the **`Lib`** folder as the **`Preferences.User`** file. It has the same format as the default file.

GIF Files

There are a few **`.gif`** images that are used in the banner screen and as tool button images. They need to be in the folder **`GIF`** under the folder **`Lib`** under the top-level folder.

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:

```
...\CHARGER24b> java -cp "CharGer.jar" CGMain
```

This should invoke the application and bring up the ***CharGer*** Hub 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.

A convenience file called **`cg.bat`** for Windows users has been provided. It invokes the above command, assuming that the **`java`** command is accessible (i.e., its directory is found in your PATH settings) in the particular command line window in which it is running.

Documentation

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

General Information

Bear in mind that the ***CharGer*** editor is currently a **beta test** 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 Hub, which looks something like Figure 2. The version number displayed on your Hub may be different; yours reflects the actual version you're using.

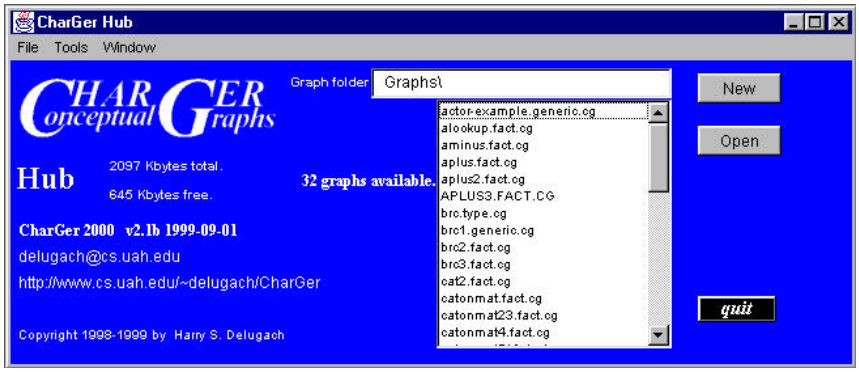


Figure 2. *CharGer* Hub Window.

From this window, one or more editing windows may be opened. An editing window looks something like Figure 3. Details of its areas and buttons are given below.

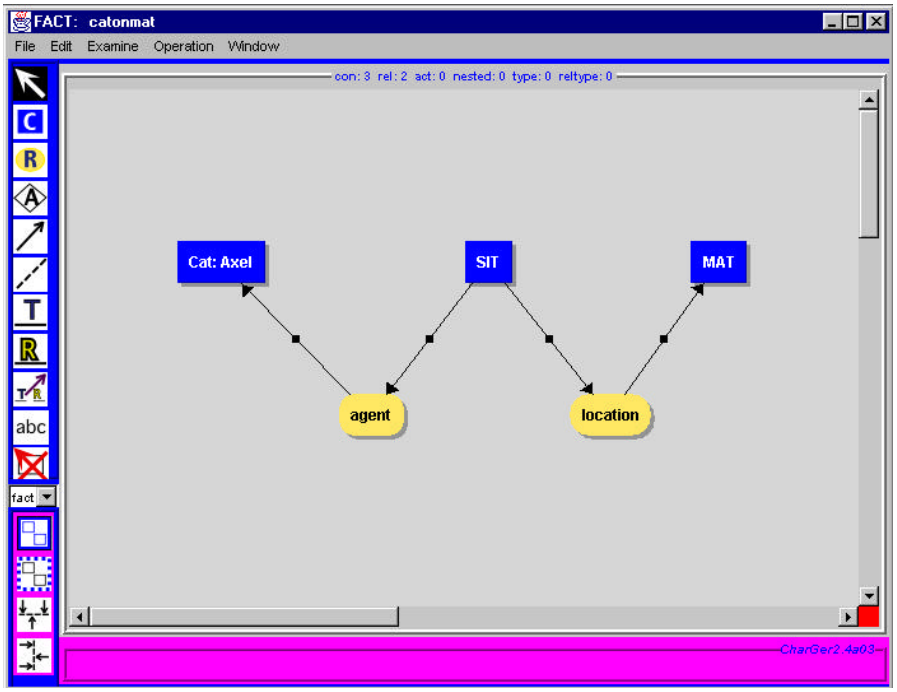


Figure 3. Editing Window.

Another window, the Driver Window, is described below. Although not strictly part of ***CharGer***, it appears in this prototype and therefore deserves some mention. 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 can still be called a graph.

We have introduced a new graphical element not previously defined in conceptual graph display environments, namely, a **typelabel** 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.

"PURPOSE" OF A GRAPH

Several authors have made reference to a graph's *intent*, that is, whether the graph is intended as a query, an empirical observation, a rule, a schema, a definition, etc. We consider this to be an important aspect of the content of a graph and 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 elements that is already defined, then the system will 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.

"Splash" Screen

The splash screen can be safely ignored, although if you want to quit, its button works. This screen's driver application illustrates how developers may build a driver that sits on top of ***CharGer***, which may even be written in any another language that can call a Java class. See the Technical Reference topic entitled Invoking ***CharGer*** from an application. Just so you'll know, the splash screen looks something like Figure 4:

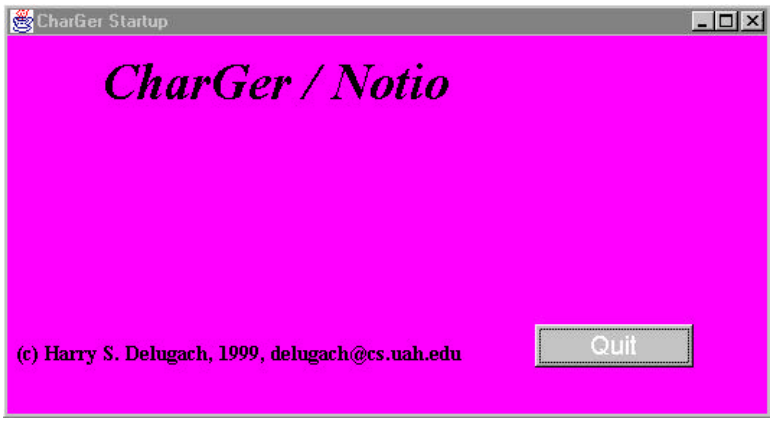


Figure 4. Splash Screen

Hub Window

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

<GraphName> . <purpose> . cg

where <GraphName> is string, <purpose> is one of the labels shown under Preferences Panel, and .cg indicates a graph file type according to its purpose. The entire notion of purpose can be turned off by an entry in the Preferences file.

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

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

Graphs are currently not saved in a standard form. The ***CharGer*** 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.

Buttons

The following buttons in the Hub are supported:

NEW

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

OPEN

Opens the selected graph in the scrolling list. Double-clicking a selection will do the same thing.

QUIT

Aborts the entire ***CharGer*** thread. Gives user a chance to save any un-saved graphs.

Menus

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

File Menu

NEW

Creates a new editing window. Operated the same way as the New button.

OPEN …

Lets the user open a graph as chosen through a standard file selection dialog window. Operated the same way as the Open button.

OPEN CGIF …

Lets the user open a CGIF (*.CGF) 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 is only one tool: the **`Database Linking Tool`** (described in Database Linking 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 5 is a summary. Each tool is described below.

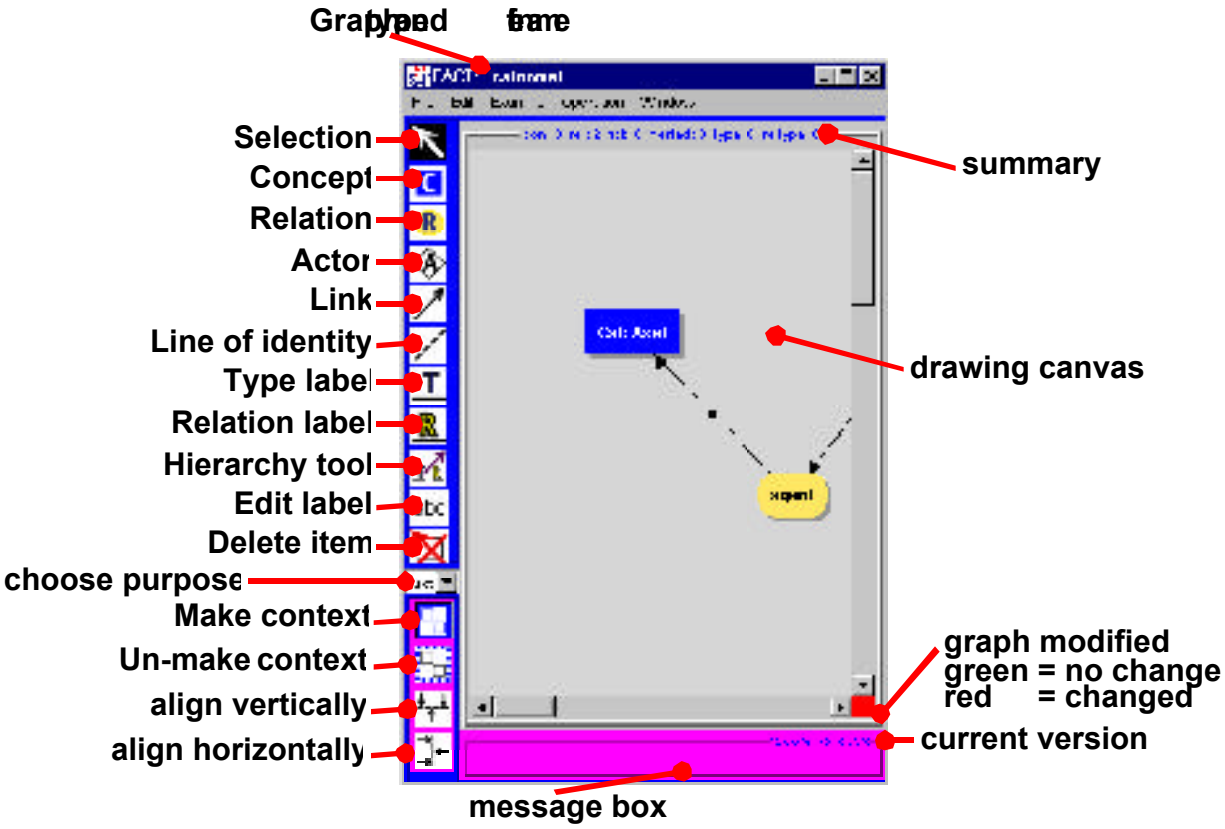
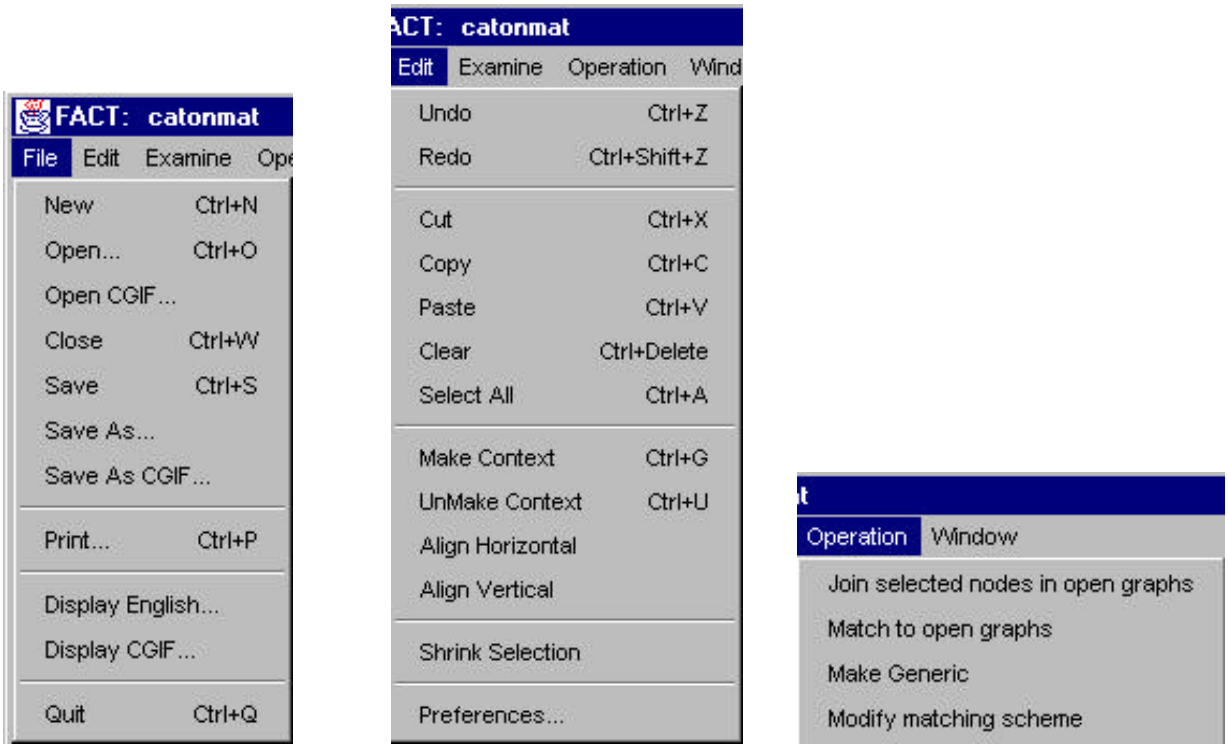


Figure 5. Basic Editing Features.

There are four menus on the Editing Window, three of which are shown in Figure 6.



(a) (b) (c)

Figure 6. 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.

OPEN CGIF ...

Invokes a file open dialog for locating a CGIF (*.CGF) file to be loaded. The file will be loaded into a new window; the current graph and its window are unaffected.

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 "Export CharGer Info" preference is checked in the Preferences Panel.

PRINT...

Prints the display form of the graph on the currently selected printer. It is recommended that you select landscape mode when printing out wide graphs.

DISPLAY 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 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).

DISPLAY CGIF…

Displays the CGIF form of the graph in a separate window. The window looks like . The display looks like Figure 7. The window's Edit menu allows you to copy part or all of the contents.

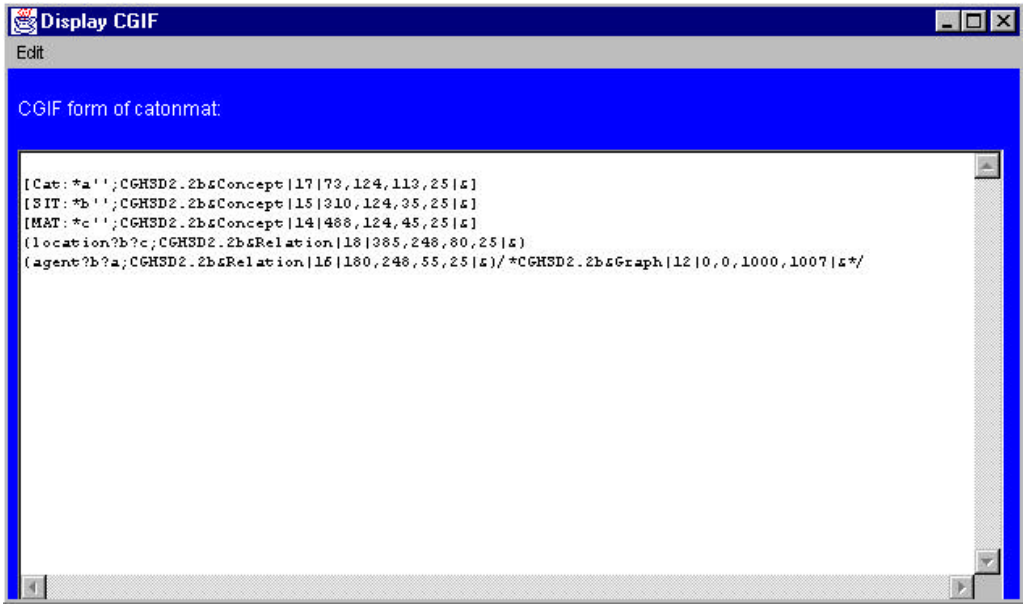


Figure 7. Display CGIF window.

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:

- Arrows, co-referent links, etc. are in the selection if they are clicked on, shift-clicked on, or included in a dragged selection rectangle.
- 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.

CUT

Copies the selection to the (internal) clipboard and removes 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.

COPY

Copies the selection to the (internal) clipboard, leaving it in 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 copies to the clipboard.

PASTE

Inserts the contents of the clipboard. If graph objects were cut/copied in ***CharGer***, then they are pasted into the current editing window. **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.

CLEAR

Erases all the selected objects, without copying them to the clipboard for later use. **Undo** restores them.

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.

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 Horizontal

Same as Align Horizontal command button.

Align Vertical

Same as Align Vertical command button.

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.

Operations 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.

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.

Match to Open Graphs

Matches the current graph to any other graphs in an open window. If a match is found

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.

Modify Matching Scheme

Allows the user to adjust the parameters of matching and joining. The command brings up a window like Figure 8.

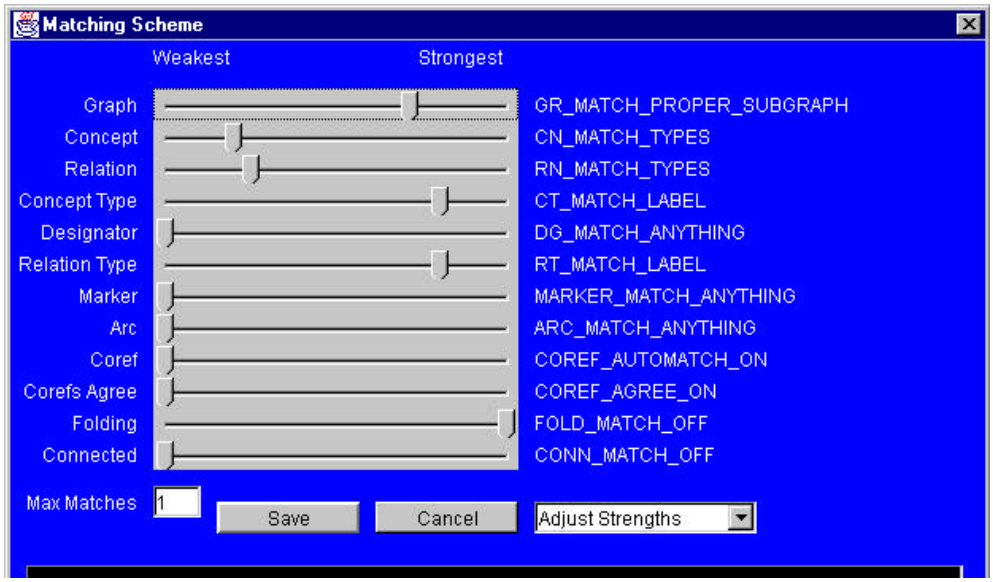


Figure 8. Matching Scheme Parameters.

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.

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 HUB

Allows convenient return to the Hub.

Tools

Selection Tool

The selection tool is the basic way to make a selection or to move graph objects. In general, the arc 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 within 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. 18) 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. 18) 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

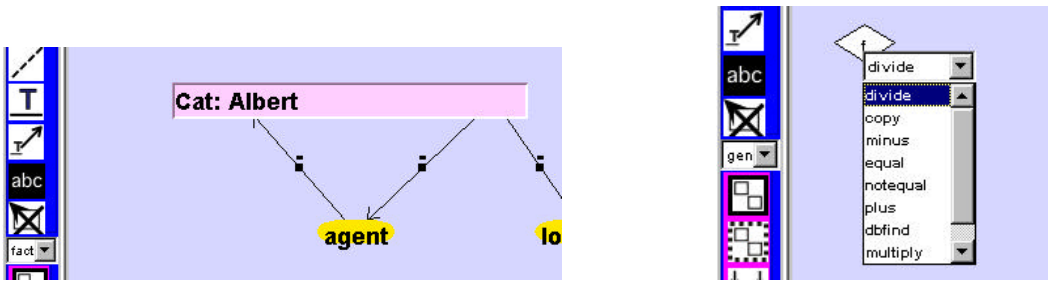
The relation type label tool allows you to insert relation types (usually in a hierarchy) onto the graph drawing area. The relation type label tool can also be chosen by pressing the "L" key on the keyboard.

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

abc 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 9(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 9(b). The user may pick an actor/relation name from the menu or create a new (undefined) actor by picking the name "Other…" and then editing it again, inserting the desired name. To edit the label on an arrow, click on the dot (■) and edit its label.



(a) (b)

Figure 9. Editing Node Names.

Text labels for concepts, relations and contexts can be changed through use of a Text Edit Box. (Actors' labels are edited by a different process; see p. 17) 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, position the cursor in the Text Edit Box and press return or enter.

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.

HotKeys

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.

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 Panel

There is a preferences panel available that allows the user to adjust some settings for their own use. These settings will override those in the **`Preferences.Default`** 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 **`Preferences.User`** file. No changes are made by ***CharGer*** to its own **`Preferences.Default`** file; those changes must be made through a regular text edit 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 groups, Appearance, Compatibility, and Actor Settings

Appearance

Show line selection handle

Display the small box at the midpoint of each line; automatically disabled for printing.

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.

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.

Context Border Width

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

Set 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 10.

Figure 10. 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.

Compatibility

Include CharGer info with CGIF

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.

Activate graph purpose 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

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). Users should probably leave it unchecked.

Enforce standard relation arguments

Enforce the standard relation rule such that a relation has at most one output argument. Never enforced for actors.

Graph Folder

The default folder from which the graph list in the Hub Frame 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…** button allows the user to change this default folder.

Actor Settings

Allow null actor arguments

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

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.

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 button allows the user to change this default folder.

Animate actor firing

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. Ignored unless actor firing is animated.

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 Preferences.User file (or creates it, if the file does not already exist).

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.

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 11 before proceeding:

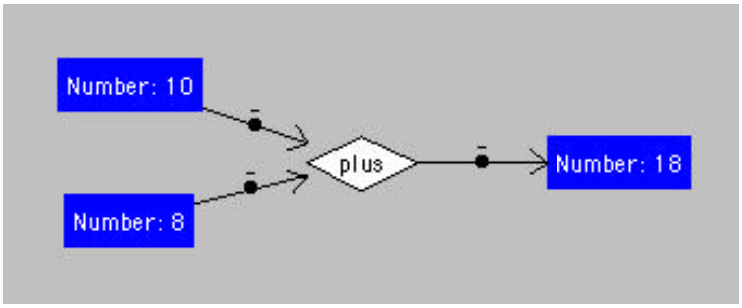


Figure 11. Actor example.

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 12:

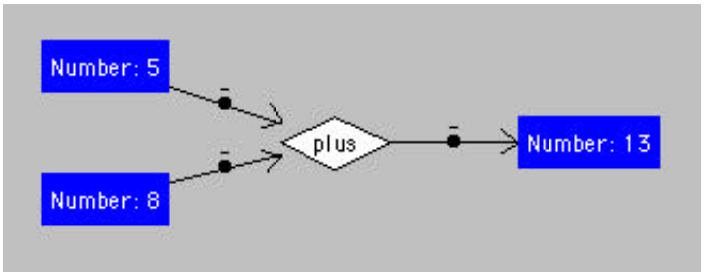


Figure 12. Changed actor graph.

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).

	Input Concepts		Output Concepts		
Actor Name	**Quantity**	**Type(s)**	**Quantity**	**Type(s)**	**Semantics**
plus	Two	**Number**	One	**Number**	Output number is sum of the two inputs. Commutative.
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.
divide	Two	**Number**	One	**Number**	Output number is concept 1 divided by concept 2.
dbfind	Two	**Database** **T**	One	**Number**	Output concept is the value associated with **T**'s referent in the file denoted by the **Database** concept.
equal	Two	**T**	One	**T or null**	Output type is **T** if inputs are strictly equal; otherwise **null**
notequal	Two	**T**	One	**T or null**	Output type is **T** if inputs are strictly not equal; otherwise **null**
copy	One	**T**	One	**T**	Input (referent only) is copied to output concept.
greaterthan	Two	**T**	One	**T or null**	Output type is **T** if input 1 greater than input 2; otherwise **null**
greaterequal	Two	**T**	One	**T or null**	Output type is **T** if input 1 greater than or equal to input 2; otherwise **null**
lessthan	Two	**T**	One	**T or null**	Output type is **T** if input 1 less than input 2; otherwise **null**
lessequal	Two	**T**	One	**T or null**	Output type is **T** if input 1 less than or equal to input 2; otherwise **null**

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

Database Linking

One of ***CharGer***'s features is its ability to use an external "database" that actor <dbfind> 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>** 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:

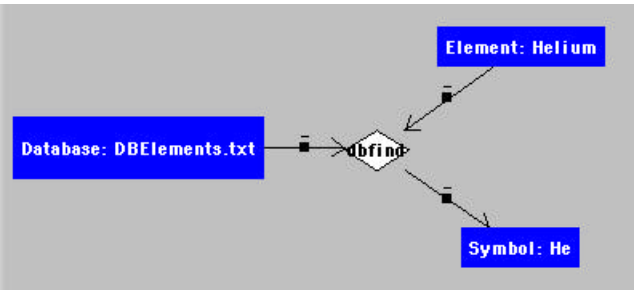


Figure 13. Illustration of the <dbfind> actor.

To see how the actor works, use the "abc" 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 **`dbfind`** 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, **`dbfind`** 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 Hub. When activated, the window looks something like Figure 14:

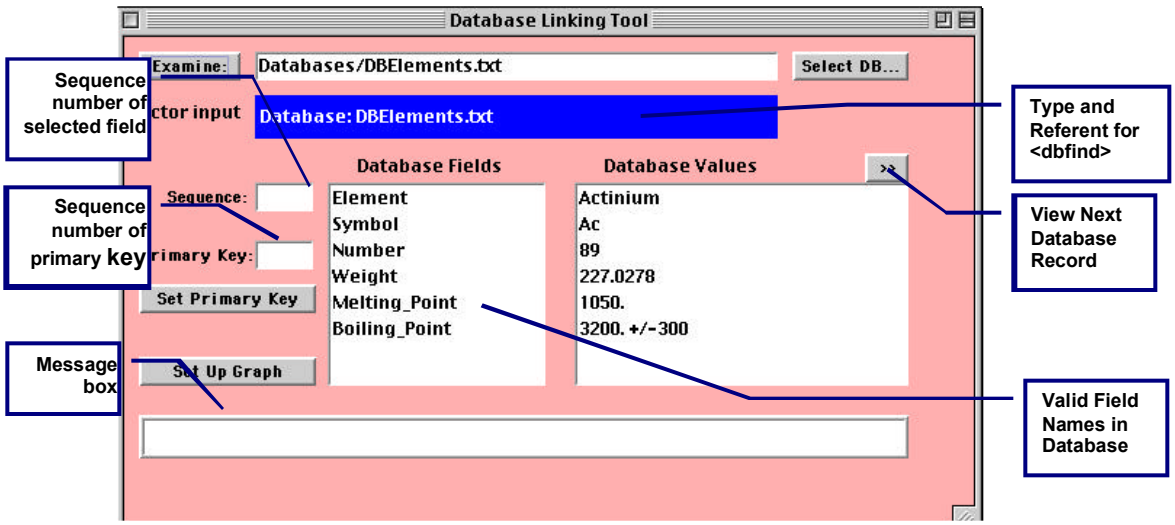


Figure 14. Database Linking Tool Window.

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 provided. 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 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.

`Input Concept` indicates the valid input concept representing the database, to be used as an input to a <dbfind> actor.

`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.

`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 <dbfind> 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.

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

- Printing on some systems may not work; please report errors and we'll see what we can do about it.
- Undo does not work within a text field, although the entire text 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.
- New primitive (built-in) actors must be coded in Java, according to the procedures outlined in comments to the **`GraphUpdater`** class.
- 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 **`dbfind`** 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 Preferences panel. There is no pathname in the **`Database`** input concept to **`dbfind`**. 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 doubt it.
- 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. They will be
- 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.
- Lambda expressions are not yet supported.
- Arithmetic with real numbers lacks precision; ***CharGer*** is not yet meant as a reliable calculator for real numbers.
- When moving a context, all it 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 will be addressed in a future release.
- Type labels 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.

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 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?

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 "–" 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 graph file and all the objects are crowded into the top left corner. Why?

The saved CGIF graph did not have embedded ***CharGer*** 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.

Technical Reference

CharGer Proprietary File Format

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:

```
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 was developed entirely under Metrowerks' CodeWarrior for the Macintosh. CharGer 2.2b consists of 45 Java classes, which make up approximately 14,300 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, it may show 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.

Bug Fixes

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

Fixed/Changed in v1.8b

- The graph purpose "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 Hub's graph list.
- An indicator (on the Hub) 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)