# Software Requirements Specification for the Graph Editor

## Document # CSC444-2001-SRS-01B

Revision B

## 14th September 2001

# Table of Contents

# 0 Revision History

## 0.1 12-Sept-2001: CSC444-2001-SRS-01A

Release A contains the basic specification for the graph editing software.

## 0.2 14-Sept-2001: CSC444-2001-SRS-01B

Release B changes:

- Added this revision history.
- Added an appendix with example graphs and their GXL representations.
- Added a reference to the CSC444 course website in section 1.3.
- Numbered the list of CSCs in section 2.2.
- Clarified the list of GXL exclusions in section 2.4.
- Added an explicit assumption in section 2.5 that requirements may be scrubbed by mutual agreement with customers if schedule and resource constraints make this necessary.
- Changed references to “BMS” in section 3.2.1 to “GDO”
- Added a requirement in section 3.2.4 to preserve all GXL information from GXL files opened for input.
- Clarified requirements concerning the provision of a makefile in section 3.5.8
- Corrected various minor punctuation, spelling and formatting problems.
- Changed some verbs from indicative mood, present tense to optative mood, future tense.

# 1 Scope

This specification establishes the functional, performance, and development requirements for Release 1 of a software application for displaying and editing graphs.

## 1.1 Purpose

The Graph Editor Software is an interactive application that allows the user to create, edit, layout, save, and print arbitrary graphs. It uses the GXL graph notation standard for storing graphs to files.

## 1.2 Definitions, Acronyms and Abbreviations

| | |
|---|---|
| CSCI | Computer Software Configuration Item |
| DTD | Document Type Definition |
| **GDO** | **Graph Data Object CSCI** |
| **GEI** | **Graph Editing Interface CSCI** |
| **GLG** | **Graph Layout Generator CSCI** |
| **GSP** | **Graph Save and Print CSCI** |
| GUI | Graphical User Interface |
| GXL | Graph Exchange Language |
| SRS | Software Requirements Specification |
| XML | Extensible Markup Language |

## 1.3 Reference Documents

The following standards apply

| | |
|---|---|
| J-STD-016-1995 | IEEE/EIA Standard for Information Technology, Software Lifecycle Processes, Software Development, Acquirer-Supplier Agreement |
| IEEE-STD-P1063 | IEEE Standard for Software User Documentation |

The following documents describe the course in which this software is to be developed:

| | |
|---|---|
| CSC444-HND-001 | Course Orientation Handout |
| CSC444-HND-002 | Notes on the Software Trading Game |

| | |
|---|---|
| http://www.cs.toronto.edu/~sme/CSC444F | Course website |

The following documents describe aspects of file formats for graph storage:

| | |
|---|---|
| http://www.gupro.de/GXL/ | The Graph Exchange Language Standard and XML DTDs |
| http://www.w3.org/XML/1999/XML-in-10-points | An introduction to XML |

There are many papers available describing graph layout algorithms, and the aesthetic principles underlying them. Try a websearch for phases such as “graph layout algorithm”. The following paper provides a detailed survey:

Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis Tollis, “Annotated Bibliography on Graph Drawing Algorithms” *Computational Geometry: Theory and Applications* 4:235-282 (1994). Available on the web at: http://www.cs.brown.edu/people/rt/papers/gdbiblio.pdf

## 1.4 Document Overview

Section 1 identifies the scope of this document, the purpose of the software, and lists the definitions, acronyms and reference documents. Section 2 provides an overview of the system, and a description of the functional architecture. Section 3 identifies the four main Computer Software Configuration Items (CSCIs) that comprise the system, and gives the functional requirements and constraints for each CSCI. Section 3 also describes the quality requirements for the software. Appendix A contains some sample graphs and their GXL representations.

# 2 Overall Description

## 2.1 System Perspective

Graphs are commonly used throughout software engineering and computer science to present structured information in a form that is easy to read. Many forms of information can readily be presented as a graph, including structural and behavioural views of a software system, such as call graphs, data flow diagrams, state machine models, etc. Briefly, a graph is a collection of *nodes* and connected *edges*. The edges may be directed (in which case they are usually shown using arrowheads) or undirected. For different applications, the nodes may be drawn as circles, dots, boxes, etc, while the edges are nearly always drawn as straight lines or arcs. Nodes and/or edges may be labelled, and for some applications, there are conventions for how the labels are presented.

The software described in this specification consists of an application for creating, viewing, editing, laying out, saving, and printing the kinds of graphs commonly used in software engineering. The application shall input graphs created by other applications, represented in the standard graph language GXL. It shall also support the creation of new graphs through the use of a graphical user interface. In either case, the application shall support automated layout of graphs according to aesthetic principles such as minimizing the crossing of edges. It shall also allow the user to manipulate an automatically-generated layout, by moving nodes and edges interactively. The application shall allow graphs to be saved in GXL format, and shall also support common imaging formats for outputting pictures of graphs.

## 2.2 System Functions

There are essentially four main functional areas, which correspond to the four CSCIs specified in section 3:

1. **Graph Data Object (GDO):** This CSCI shall provide a memory resident data structure to hold the current graph, which shall be used by the other CSCIs. GSP shall initiate this data structure either by getting GSP to read in a description of an existing graph from a GXL file, or by generating a new empty graph for the user to edit. This CSCI shall provide functions to the other CSCIs to traverse and manipulate the graph, but shall not pass the entire graph as a data structure to other CSCIs.
2. **Graph Editing Interface (GEI):** This CSCI shall provide a Graphical User Interface (GUI), to allow the user to view and edit graphs. If the graph to be displayed contains layout information, the GEI shall use this information to arrange the graph on the screen; otherwise GEI shall call GLG to provide a new layout. The GUI shall provide functions to the user to *add* or *delete* nodes and edges, *move* nodes, *edit the labels* on nodes and edges, change the *shape* of nodes, change the *size* of nodes, and change the *style* of edges. It shall also provide access to general commands such as opening a GXL file, creating a new graph, and saving a graph as GXL or as an image. This CSCI shall not store graphs in memory. It shall call GDO to access and update information contained in the current graph. It shall call GSP for opening and saving files.
3. **Graph Layout Generator (GLG):** This CSCI shall lay out an existing graph using the aesthetic conventions of the appropriate application, using a standard graph layout algorithm. If a given graph has partial layout information, GLG shall use this information as a basis, and complete the layout. Otherwise, GLG shall generate a complete layout for the graph. GLG shall call GDO to access information about the current graph, and to store layout information as part of this graph.
4. **Graph Save and Print (GSP):** This CSCI shall be responsible for opening GXL files and for saving the current graph to a file. For opening a file, GSP shall parse the GXL information and call the functions of GDO to store the graph in memory. GSP shall save the file as one of two formats: as GXL, or as an image (for later printing). For a GXL file, all information about the graph (including layout, and size and shape of nodes and edges) shall be stored in the GXL description. For an image file, the current view of the graph shall be stored as an image in one of a number of common image formats. Structure information about the graph may be lost in this case. GSP shall provide error handling for cases where open and save operations are unsuccessful.

The architecture is summarized in Figure 2.1.

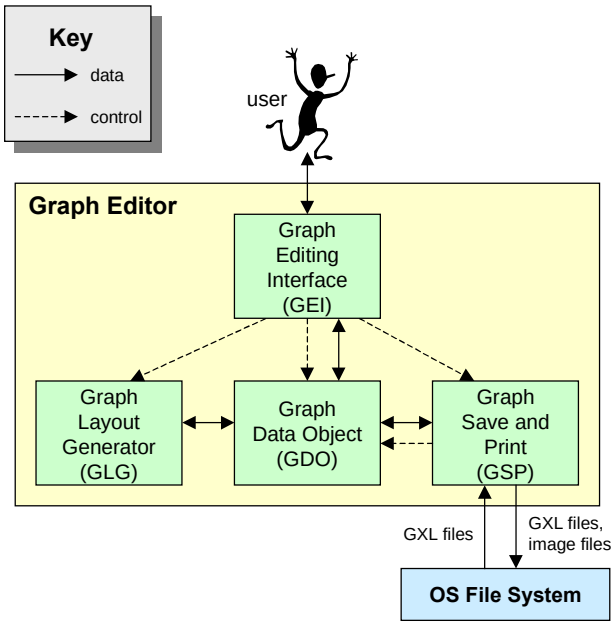



**Figure 2.1**: The System Architecture

## 2.3 User Characteristics

No special knowledge or skills shall be assumed on the part of the users. Users shall not be expected to learn a set of commands in order to start using the application. Users shall not be expected to remember a list of commands while using the software – these shall be provided via menus, tool palettes, or help screens. Users shall be protected from data loss, such as closing a graph without saving it first.

## 2.4 Constraints

For release 1 of the software, a subset of the GXL notation shall be used. Specifically, the software is not required to handle hierarchical graphs (i.e. graphs that contain composite nodes or edges – nodes/edges that contain other nodes or graphs), hypergraphs (i.e. graphs that contain n-ary relationships), and GXL extensions. If a GXL input file contains a graph that uses these features, they shall be ignored by the program, but preserved if the graph is saved to a GXL file.

## 2.5 Assumptions and Dependencies

It is assumed that the requirements described in this document have different levels of priority. Although no specific priorities have been documented, it is assumed that software contractors may need to scrub low priority requirements in the face of schedule and resource pressures. Requirements should only be scrubbed by agreement with the customer, and only if it can be clearly demonstrated that there is no other high priority (non-scrubbed) requirement that depends upon the scrubbed requirements.

No other special assumptions or dependencies have been identified.

# 3 Specific Requirements

## 3.1 External Interface Requirements

### *3.1.1 User Interfaces*

All interaction with the user shall be via the GEI CSCI. The interface provided by the GEI may be graphical or textual. The interface shall always give a view of the current graph. If the current graph is too large to display on the screen, the GEI shall display part of it, and provide controls for scrolling to the hidden parts. The GEI shall also provide a mechanism for zooming out, to allow the user to see the entire graph in a single view (without necessarily seeing all the detail). The GEI shall also provide an interface for loading and saving files. It shall provide a visual reminder if the current graph has been edited but not yet saved, and in this state, shall warn the user if she attempts to close the graph (or open a different one) without saving the current graph first. It shall also provide a warning if the user attempts to overwrite an existing file. For ease of use, the GEI shall allow the user to browse the file system when opening or saving files, and the GEI shall remember the name and location of the last file opened.

### *3.1.2 Hardware Interfaces*

None

### *3.1.3 Software Interfaces*

The system shall make use of the operating system calls to the file management system to store and retrieve files containing graphs and images. Platform independence and portability requirements are described in section 3.5.5.

### *3.1.4 Communication Interfaces*

The system shall be capable of exchanging graph information with other applications that are GXL compliant.

## 3.2 Functional Requirements

This section describes the functional requirements for each of the four CSCIs: Graph Data Object CSCI (GDO), Graph Editing Interface CSCI (GEI), Graph Layout Generator CSCI (GLG), and Graph Save and Print CSCI (GSP).

### *3.2.1 Graph Data Object CSCI (GDO)*

The GDO CSCI shall store the current graph in memory, and provide a functional interface to other CSCIs to access and make changes to the current graph. It shall also be responsible for checking the validity of the current graph.

#### *3.2.1.1 GDO Internal Data*

The main internal data used by the GDO shall be a representation of the current graph. GDO shall hold all data about the current graph in a suitable data structure, and shall provide information about the graph to the other CSCIs. The following data shall be stored:

- For the graph as a whole, GDO shall store the filename that this graph was last saved to (if any), the size of the grid on which it will be laid out, and optional global defaults for the size and appearance of nodes and edges.
- For each node in the graph, GDO shall provide a unique identifier by which other CSCIs may refer to it. Each node may additionally have an optional textual label, an optional location represented as a grid reference, an optional size, an optional shape, and an optional colour.
- For each edge in the graph, GDO shall record which nodes are connected by the edge. Each edge may additionally have an optional textual label, an optional line style (e.g. solid, dashed, dotted, etc), an optional thickness, and optionally, arrowheads on either end.

Note that information about the size, location and appearance of nodes and edges can be stored in GXL either as attributes of nodes and edges, or as type information. In the latter case, a graph schema will typically define a small number of node and edge types, so that appearance information needs to be defined only once. For example, a dataflow diagram may have three different node types (e.g. “process”, “datastore”, and “external entity”), where the

attributes of each type define its appearance, and each node in the graph is declared to be one of these types. In either case, GDO shall store the relevant information in appropriate data structures, so that it can be saved again to a GXL file in the same form as it originally appeared.

GDO shall also perform validity checks on the current graph when requested. For a graph to be valid, there must be no disconnected nodes, and no disconnected subgraphs, all edges must connect to valid nodes, and all grid references representing node locations must lie with the graph's given grid size.

#### *3.2.1.2 GDO External Interface*

The GDO CSCI provides the following functions for external use by other CSCIs:

new_graph(): initializes the internal data structure with a new empty graph.

add_node(nodeID): given a string containing a node identifier that is not already a node in the current graph, creates a new node in the current graph with this identifier. Returns true if the creation was successful, false otherwise.

delete_node(nodeID): given a valid node identifier in the current graph, deletes that node from the current graph. Returns true if the deletion was successful, false otherwise.

add_edge(nodeID, nodeID, edgeID): given two valid node identifiers in the current graph, and a string containing an edge identifier that is not already an edge in the current graph, adds an edge between the two nodes with the given identifier. Returns true if the creation was successful, false otherwise.

delete_edge(edgeID): given a valid edge identifier in the current graph, deletes that edge from the current graph. Returns true if the deletion was successful, false otherwise.

get_label(edgeID|nodeID): given a valid edge identifier or a valid node identifier in the current graph, returns a string containing the textual label for that edge or node.

set_label(edgeID|nodeID, label): given a valid edge identifier or a valid node identifier in the current graph, and a string, set the textual label for that edge or node to the given string. Returns true if the change was successful, false otherwise.

get_position(nodeID): given a valid node identifier in the current graph, returns a pair of coordinates describing the position of that node on the grid.

set_position(nodeID, coordinates): given a valid node identifier in the current graph and a pair of coordinates on the grid, sets the position of the given node to the given coordinates. Returns true if the change was successful, false otherwise.

get_grid(): returns a pair of integers representing the grid size used in the layout of the current graph.

set_grid(X, Y): given two positive integers, sets the grid size of the current graph to the given values.

expand_grid(factor): given a positive integer, expands the size of the current grid by this factor, multiplying all node location coordinates by the same factor (this function effectively increases the resolution of the grid). Returns true if the change was successful, false otherwise.

get_node_style(nodeID): given a valid node identifier in the current graph, returns a record giving appearance parameters for the given node.

set_node_style(nodeID, appearance): given a valid node identifier in the current graph, and a record giving appearance parameters for the given node, sets the appearance of the node to the given parameters. Returns true if the change was successful, false otherwise.

get_edge_style(edgeID): given a valid edge identifier in the current graph, returns a record giving appearance parameters for the given edge.

set_edge_style(edgeID, appearance): given a valid edge identifier in the current graph, and a record giving appearance parameters for the given edge, sets the appearance of the edge to the given parameters. Returns true if the change was successful, false otherwise.

list_neighbours(nodeID): given a valid node identifier in the current graph, returns a list of all nodes that are directly connected to this node via edges.

list_edges(nodeID): given a valid node identifier in the current graph, returns a list of edgeIDs of edges connected to that node.

first_node() *and* next_node(): These functions provide an iterator over all the nodes in the current graph. first_node initializes the iterator, returning the nodeID of the first node (chosen arbitrarily). Each time next_node is called, it returns the nodeID of the next node in the graph (in some arbitrary order). Each node in the current graph is returned at most once in any sequence of calls to next_node() (unless there is an intervening call to first_node()). If there are no more nodes, next_node returns an empty string.

number_of_nodes(): returns the total number of nodes in the current graph.

valid_graph(): checks whether the current graph is valid. Returns true if the graph is valid, false otherwise.

### *3.2.2 Graph Editing Interface CSCI (GEI)*

The GEI CSCI shall provide a user interface for viewing and editing graphs. GEI shall provide a view of the current graph, along with the ability to zoom in and out, and to scroll to any parts of the graph not visible in the current view. GEI shall also provide a set of tools for editing graphs, including the ability to add and delete nodes and edges, and to set various parameters concerning the visual appearance of the graph. All user interaction with the software shall be handled by GEI. Hence GEI shall also provide user commands for opening and saving to files, for navigating the filesystem, and for generating and modifying layouts.

GEI shall not not store any data about the current graph. GEI shall use the GDO to store the current graph, including layout and appearance information, and to check that the current graph is valid. GEI shall use the GLG to generate layouts prior to displaying any graphs that do not have complete layout information. GEI shall also provide user commands for modifying the automatically generated layout (e.g. by moving individual nodes). GEI shall not directly interact with the operating system for access to files; all access to the filesystem shall be via GSP.

#### *3.2.2.1 GEI Internal Data*

GEI shall store information about the past sequence of edit actions, so that it can provide a multi-level undo. GEI shall allow the user to undo an entire sequence of edits, back to the last time the graph was saved to a file. The saved list of edit actions shall be cleared each time the current graph is saved.

GEI shall store a list of recently opened files to allow the user instant access to these files.

GEI shall store user preferences, including the size, position and zoom level of the current view, so that these can be restored if the software is closed and reopened.

#### *3.2.2.2 GEI External Interface*

The GEI CSCI shall use the functions provided by GDO, GLG and GSP to provide information about the current graph, to generate layouts, and to open, save and print the current graph.

The GEI CSCI shall make the following functions available to other CSCIs:

ask_user(message, type): given a string containing a message, and a primitive type (integer, boolean, string, or real), asks the user to input a value of the given type, using the given message as a prompt. Returns the value input by the user.

#### *3.2.2.3 GEI User Commands*

The GEI CSCI shall make the following commands available to the user. These commands shall be provided by whatever interface mechanisms are appropriate, (e.g pull-down or pop-up menus, tool palettes, keyboard shortcuts, etc). A combination of such mechanisms may be used if usability testing suggests they are helpful.

- **new**: initializes the current graph. This command must give the user a warning (and a chance to cancel the operation) if the previous current graph has not been saved since the last edit
- **open**: allows the user to browse the directory structure to select a GXL file to open. This command must give the user a warning (and a chance to cancel the operation) if the previous current graph has not been saved since the last edit.

- **save**: saves the current graph to the most recent GXL file associated with this graph. If the current graph has been initialized and not yet saved to a file, this command functions the same as the "save as" command.
- **save as**: this command shall prompt the user to select a directory and a filename to save the current graph to (as GXL). If the selected filename already exists, this command shall ask the user to confirm whether to overwrite the existing file.
- **export**: this command shall prompt the user to select an image type to export the file as (selected from those image types supported by GSP), along with any necessary parameters (such as image size, resolution, etc). It will then prompt the user to select a directory and a filename to export an image of the current graph to.
- **zoom in**: allows the user to zoom in on the current view, centered on the selected node(s). If no nodes are selected, this command should zoom in on the last edited node. If no nodes are selected, and no nodes have been edited, this command should zoom in on the center of the current view. At least two levels of magnification should be offered.
- **zoom out**: allows the user to zoom out from the current view. Enough levels of magnification shall be provided to allow the user to zoom out such that the entire current graph is visible (even if much of the detail is lost).
- s**croll**: allows the user to scroll to all parts of the graph that are not visible in the current view, without changing the zoom level. If the entire graph is visible, this command shall not be enabled.
- **add node**: allows the user to create a new node and place it on the grid.
- **select node(s)**: allows the user to select one or more nodes in the current graph on which to perform edit actions. This command shall include functions to add nodes to the current selection, select all nodes in the graph, and to clear the selection.
- **delete node**: allows the user to delete the currently selected node(s) from the graph.
- **move node**: allows the user to move the currently selected node(s) on the grid. This command should ensure that edges connected to the moved nodes are redrawn correctly.
- **change node style**: allows the user to change the appearance of the currently selected node(s), by changing their shape, size, colour or other relevant parameters.
- **add edge**: allows the user to add an edge between two nodes. An appropriate interface mechanism should be used to allow the user to indicate which two nodes the edge should connect.
- **select edge(s)**: allows the user to select one or more edges in the current graph on which to perform edit actions. This command shall include functions to add edges to the current selection, select all edges in the graph, and to clear the selection.
- **delete edge**: allows the user to delete the currently selected edge(s) from the graph.
- **change edge style**: allows the user to change the appearance of the currently selected edge(s), by changing their thickness, arrowheads, or other relevant parameters.
- **create new layout**: allows the user to invoke the auto-layout function on demand. If one or more nodes is currently selected, only the selected nodes will be laid out (with all other nodes fixed).
- **undo**: allows the user to undo edit actions. A multi-level undo shall be provided to allow the user to keep undoing edit actions, back to the last saved version of the file.
- **redo**: allows the user to 'undo' an undo command. This command shall be enabled immediately after an undo command has been used, and disabled as soon as other edit actions are performed.
- **repeat**: allows the user to repeat the last edit action on the current selection.
- **revert**: allows the user to revert to the last saved version of the current graph. This command must ask the user to confirm that they wish to discard all edits to the current graph.

### *3.2.3 Graph Layout Generator CSCI (GLG)*

The GLG CSCI shall be responsible for automatically generating a layout for the current graph. If the current graph has location information for some nodes and not for others, GLG shall use the existing layout for those nodes that have locations defined, and layout the remaining nodes. If the current graph has a grid size defined, GLG shall attempt to use this grid. If there is no grid size, or the grid size is smaller than the number of nodes in the current graph, GLG shall set a new, appropriate grid size.

GLG shall use a standard graph layout algorithm to find an aesthetically pleasing layout. Typical aesthetics for graph layouts include minimizing the number of edges that cross one another, keeping the lengths of edges to be similar to each other, and keeping nodes that are connected to one another closer together than nodes that are not. Note that most standard graph layout algorithms have input parameters that affect the way the layout is generated. These parameters should be made available to the user to select via the ask_user command of the GEI. If appropriate, GLG may include more than one layout algorithm. If so, the user should be allowed to select between the available algorithms.

GLG shall not store any data about the current graph. GLG shall use the GDO to store the current graph, including layout information, and to check that the current graph is valid.

GLG shall not interact with the user directly. All user interactions shall be handled by GEI.

#### *3.2.3.1 GLG Internal Data*

The GLG may need to store temporary information about the layout being generated, according to the algorithm(s) used.

GLG shall keep a copy of any layout information for the current graph that it overwrites, so that old layouts can be retrieve by the user via the undo command.

#### *3.2.3.2 GLG External Interface*

The GLG shall provide the following functions for use by other CSCIs:

layout(): completes a layout for the current graph, preserving any location information already defined. Returns true if it successfully generates a new layout, false otherwise.

new_layout(): generates a new layout for the current graph, discarding any existing location information defined. Returns true if it successfully generates a new layout, false otherwise.

partial_layout(list of nodeIDs): given a list of valid node identifiers, generates a new layout for the current graph, by discarding any existing location information for the given nodes, but preserving location information for all nodes not in the given list. Returns true if it successfully generates a new layout, false otherwise.

undo_layout(): restores the previous layout. Returns true if it successfully restores a layout, false otherwise.

### *3.2.4 Graph Save and Print CSCI (GSP)*

The GSP CSCI shall be responsible for opening GXL files, saving the current graph to a GXL file, and for exporting views of the current graph to image files for displaying in other applications and for printing. GSP shall be responsible for all interactions with the file system, including checking whether a file exists, and moving between directories.

GSP shall preserve all information in a GXL file opened by the software and subsequently saved as GXL (whether to the same file or a different file), except where information is explicitly deleted by the user prior to saving. Information the in GXL files that must be preserved includes, but is not limited to: unique identifiers; location information for nodes; type information; all attributes of nodes and edges; links to external documents; GXL extensions; and features of GXL not used by this software.

GSP shall support one image export format chosen from the following list: GIF image, JPEG image, Encapsulated Postscript (EPS), BMP bitmap, TIFF bitmap, or portable document format (PDF). The image saved shall correspond as closely as possible to the appearance of the graph displayed by GEI.

#### *3.2.4.1 GSP Internal Data*

GSP shall not store any information about the current graph. All data read in from GXL files shall be passed to GDO to be stored in memory.

#### *3.2.4.2 GSP External Interface*

The GSP CSCI shall provide the following functions for external use by other CSCIs:

open_graph(filename): given a valid filename in the current directory, if the file contains GXL, the GXL information is passed to GDO to be stored in the appropriate data structures. Returns true if it succeeded in loading a new graph, false otherwise. If the graph could not be loaded for any reason, the previous current graph should be unchanged; otherwise any previous graph data is discarded.

save_graph(): saves the current graph to the GXL file from which it was read. Returns true if the save operation was completed successfully, false otherwise. If the current graph is not associated with a file, save_graph returns false.

save_graph_as(filename): given a string, saves the current graph as a GXL file with the given name in the current directory. Returns true if the save operation was completed successfully, false otherwise.

save_image(filename): given a string, saves the current graph as an image file with the given name in the current directory. Returns true if the save operation was completed successfully, false otherwise.

exists(filename): given a string, returns true if the given string is the filename of an existing file in the current directory, false otherwise.

change_dir(dirname): given a string, changes the current directory to the given directory if it exists. Returns true if the directory change was completed successfully, false otherwise.

list_dir(): returns a list of the files and subdirectories in the current directory as a list of strings.

## 3.3 Performance Requirements

The system shall respond to each user input within 2 seconds. The system shall be able to handle large graphs (thousands of nodes), as would be generated from typical reverse engineering tools. When laying out large graphs, speed is important. If the layout will take more than a few seconds, the user shall be warned of this, and given a progress indicator. There are no other performance requirements.

## 3.4 Design Constraints

### *3.4.1 Standards Compliance*

The input and output file formats should conform to the GXL standard.

All language used in the software (including manuals and documentation) should comply with current University of Toronto guidelines for decency and equal opportunities.

## 3.5 Software System Attributes

### *3.5.1 Reliability*

The system shall never crash or hang, other than as the result of an operating system error. The system shall provide graceful degradation in the face of network delays and failures, or when handling large graphs.

### *3.5.2 Availability*

There are no specific availability requirements.

### *3.5.3 Security and Privacy*

There are no specific security and privacy requirements, other than those generally governing use of student login accounts on University of Toronto computer equipment.

### *3.5.4 Maintainability*

All code shall be fully documented. Each function shall be commented with pre- and post-conditions. All program files shall include comments concerning authorship and date of last change.

The code shall be modular to permit future modifications.

### *3.5.5 Portability*

The software shall be designed to run on *at least one* of the following four platforms:

- Microsoft Windows (Windows implementations shall be portable to all versions of Windows up to and including Windows XP)
- UNIX (Unix implementation shall be portable to any version of Unix that supports the user interface libraries used)
- Apple Macintosh OS (Apple Mac implementations shall be portable to all current versions of the MacOS)
- Linux (Linux implementations shall run on at least one version of Linux)

No other specific portability requirements have been identified.

### *3.5.6 Safety*

No safety requirements have been identified.

### *3.5.7 Training-related Requirements*

No specific training should be necessary for a user to begin using this application.

### *3.5.8 Packaging Requirements*

The system shall be packaged along with source code and all documentation, and shall be available for electronic transfer as a single compressed file. The uncompressed set of files shall include a README file containing a minimal guidance for installing and running the software, including recompilation if needed. If recompilation is necessary during installation, the system shall include a makefile.

### *3.5.9 Legal Requirements*

Copyright laws and license agreements must be respected for any third party software used in the creation of this system.

## 3.6 Other Requirements

There are no other requirements.

# Appendix A: Example graphs

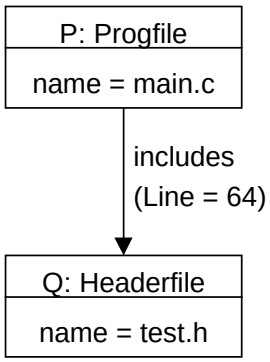
## Example 1

A dependency graph generated from a simple C program. Note that the type information is linked to a separate GXL file that contains a schema for this type of graph. The schema is not shown here.

**GXL file**

```
<gxl>
  <graph edgemode = "directed">
    <node id = "P" >
      <type xlink:href = "schema.gxl#Progfile">
      <attr name = "name">
        <string> main.c </string>
      </attr>
    </node>
    <node id = "Q" >
      <type xlink:href = "schema.gxl#Headerfile">
      <attr name = "File">
       <string> test.h </string>
      </attr>
    </node>
    <edge id = "r1" from = "P" to = "Q">
      <type xlink:href = "schema.gxl#includes">
      <attr name = "Line">
        <int> 64 </int>
      </attr>
    </edge>
  </graph>
</gxl>
```

**Possible graphic representation**



## Example 2

A simple state machine model of the states involved in making a phone call.

**GXL file**

```
<gxl>
 <graph ID='caller' edgemode='directed'>
   <node ID='idle'>
     <attr name='OFFHOOK'>     <bool> False </bool> </attr>
     <attr name='CALLEE_SEL'>  <bool> False </bool> </attr>
     <attr name='CALLEE_FREE'> <bool> True </bool>  </attr>
     <attr name='CONNECTED'>   <bool> False </bool> </attr>
   </node>
   <node ID='ringtone'>
     <attr name='OFFHOOK'>     <bool> True </bool>  </attr>
     <attr name='CALLEE_SEL'>  <bool> True </bool>  </attr>
     <attr name='CALLEE_FREE'> <bool> True </bool>  </attr>
     <attr name='CONNECTED'>   <bool> False </bool> </attr>
   </node>
   <node ID='dialtone'>
     <attr name='OFFHOOK'>     <bool> True </bool>  </attr>
     <attr name='CALLEE_SEL'>  <bool> False </bool> </attr>
     <attr name='CALLEE_FREE'> <bool> True </bool>  </attr>
     <attr name='CONNECTED'>   <bool> False </bool> </attr>
   </node>
   <node ID='busytone'>
     <attr name='OFFHOOK'>     <bool> True </bool>  </attr>
     <attr name='CALLEE_SEL'>  <bool> True </bool>  </attr>
     <attr name='CALLEE_FREE'> <bool> False </bool> </attr>
     <attr name='CONNECTED'>   <bool> False </bool> </attr>
   </node>
   <node ID='connected'>
     <attr name='OFFHOOK'>     <bool> True </bool>  </attr>
     <attr name='CALLEE_SEL'>  <bool> True </bool>  </attr>
     <attr name='CALLEE_FREE'> <bool> False </bool> </attr>
     <attr name='CONNECTED'>   <bool> True </bool>  </attr>
   </node>
   <edge from='idle' to='dialtone'>
     <attr name='label'>     <string> pickup </string></attr>
   </edge>
   <edge from='dialtone' to='idle'>
     <attr name='label'>     <string> putdown </string></attr>
   </edge>
   <edge from='dialtone' to='ringtone'>
     <attr name='label'>     <string> isfree </string></attr>
   </edge>
   <edge from='dialtone' to='busytone'>
     <attr name='label'>     <string> isbusy </string></attr>
   </edge>
   <edge from='ringtone' to='idle'>
     <attr name='label'>     <string> putdown </string></attr>
   </edge>
   <edge from='ringtone' to='connected'>
     <attr name='label'>     <string> answered </string></attr>
   </edge>
   <edge from='busytone' to='idle'>
     <attr name='label'>     <string> putdown </string></attr>
   </edge>
   <edge from='connected' to='idle'>
     <attr name='label'>     <string> putdown </string></attr>
   </edge>
   <edge from='connected' to='dialtone'>
     <attr name='label'>     <string> cutoff </string></attr>
   </edge>
 </graph>
</gxl>
```

**Possible graphic representation**