Mattias Werner

daVinci - Online Documentation V2.1

Bremen Institute of Safe Systems (BISS)
Center for Computing Technology (TZI)

University of Bremen, 1998

Overview

daVinci - Online Documentation V2.1

Overview

General Information

User Interface

Tutorial

Questions & Answers

New Features in *daVinci* V2.1.x

Reference

***daVinci* in the Internet**

daVinci V2.1 Online Documentation - Page update: June 15, 1998

Overview -> General Information

daVinci - Online Documentation V2.1

General Information

This chapter covers general information about the interactive graph visualization system *daVinci*. The following subtitles are available:

- **Introduction to *daVinci***
- **Using the Online-Documentation**
- **Platform Requirements and Installation**
- **License Agreement**
- ***daVinci's* Concepts**

daVinci V2.1 Online Documentation - Page update: June 15, 1998

Overview -> General Information -> Introduction to *daVinci*

daVinci - Online Documentation V2.1

Introduction

Welcome to the interactive graph visualization system *daVinci*. You can always go back to this page by using menu Help/Introduction in *daVinci*.

daVinci is a universal, generic visualization system for automatic generation of high-quality drawings of directed graphs. Graphs are frequently used in computer applications as a general data structure to represent objects and relationships between them. They are used to implement hierarchies, dependency structures, networks, configurations, dataflows, and so on. In despite of this manifold demand, techniques to visualize such graphs are not common in today's computer applications. So frequently a user has to deal with uncomfortable textual interfaces or poor ad-hoc drawings of graphs, because high-quality graph layout is difficult to implement and reusable tools for graph visualization are often hard to find.

This fact was the motivation for developing the interactive graph visualization system *daVinci*. The primary objective is to provide a universal layout tool for directed graphs. This way any application program can be equiped with a comfortable *daVinci* graph user interface in a very short time, and the application programmer does not need to deal with layout algorithms and computer graphics. Instead, the application is able to send its graphs to *daVinci's* API (Application Programmer Interface) and let *daVinci* do the visualization.

daVinci is developed since the end of 1992 at the Computer Science Department of Universität Bremen, Germany. The developers of *daVinci* are Michael Fröhlich (until November 1997) and Mattias Werner, with the help of Sebastian Mangels (User-Interface of *daVinci* V2.x). More than 6600 unique sites have already fetched *daVinci* from the ftp server at Universität Bremen, not counting numerous ftp mirror servers in Europe, America, Asia and Australia. We have been informed about more than 3000 *daVinci* installations at educational and commercial sites all over the world by e-mail feedback. The estimated total number of sites using *daVinci* seems to be much higher. On the *daVinci* WWW server, we provide more information about selected *daVinci* sites and their applications. *daVinci* is free available for non-profit use (read the license agreement) and can be downloaded by anonymous ftp. Papers about *daVinci* and references about related work can be found in the bibliography.

What you should read next:

- It is recommended to start by reading the Using the Online Documentation page.
- Be sure to take a look at the *daVinci* Internet Resources, e.g. WWW server, mailing list and bug-report.
- For getting an overview continue by reading about *daVinci's* concepts.
- Finally take a look at the Tutorial for a guided tour through *daVinci* V2.1.x.

daVinci V2.1 Online Documentation - Page update: June 15, 1998

daVinci - Online Documentation V2.1

Using the Online Documentation

The *daVinci* online documentation is the set of HTML pages you are currently reading. For best results the documentation should be loaded in the Netscape Navigator WWW browser. The search engine of the documentation is written in Java which is available in Navigator 2.0 or higher. Further, the *daVinci* help system communicates with Netscape to display particular pages of the online documentation when the user requests for help, e.g. by pressing the help button in a dialog window. The help system does not work with other browsers. Refer to Questions & Answers for a PostScript version of the documentation (Warning: information is lost in the Postscript documentation because hyperlinks are not present there!). The HTML hypertext approach has several advantages:

- Documentation is immediately available at any time.
- Clickable hyperlinks can be used for browsing the documentation.
- Easy access to *daVinci* Internet resources.
- Full integration of the help system.

HTML is a format for hypertexts and can be loaded in a WWW browser. By clicking on one of the blue underlined links or buttons, you can go to another page where more information is provided about the link. Most browsers offer the opportunity to go back and forth according to the followed links and to see the history of previously visited pages. These navigation features are very important to avoid getting lost in the *daVinci* online documentation. Consult the manual of your browser for details about navigation.

Using the Online Documentation together with daVinci

Of course the online documentation can be read "stand-alone" with any kind of WWW browser by loading file *overviewF.html*, the topmost page of the documentation. If you have installed the Netscape Navigator browser, version 1.1 or higher, *daVinci* will be able to communicate with the browser to display particular HTML pages when the user requests for help. This communication only works with Netscape Navigator at the moment, so it is strongly recommended to use this software together with *daVinci*. Product informations about Netscape Navigator are available in WWW.

After you have successfully installed Netscape, *daVinci* should be able to communicate with the browser by default. This is true if:

1. You are able to start Netscape by typing *netscape* in your UNIX shell. This means that the Netscape binary must have its default name and can be found in one of the directories specified by your environment variable $PATH.
   Note: If *netscape* is a shell script at your site, make sure that the script passes the commandline arguments to the Netscape binary (by using *$**), otherwise it is not possible to use *daVinci's* help

system.

2. You have set environment variable $DAVINCIHOME to the directory where the *daVinci* binary is inside (i.e the topmost directory of the *daVinci* V2.x distribution). Further, this directory must contain the online documentation in the *docs* directory (this is default).

If these two conditions are **not** true, you can specify both the location and name of the Netscape binary and the *daVinci* online documentation by using the Options/General Settings... dialog window in *daVinci* and saving the options with menu Options/Save Options afterwards. By starting *daVinci* for the first time, the system will try to launch Netscape to present the introduction page of the online documentation. So this way you can easily find out if communication with the WWW browser works.

Technically, *daVinci* does not start the browser each time as a new process. Instead, Netscape is called with the *-remote* option. This command looks for an already open browser and gives instruction to load a particular URL. Otherwise, if no Netscape process is currently running, a new browser will be started.

How to get help in daVinci

First of all, *daVinci* must be able to communicate with Netscape. Read above for details about this issue. There are three different ways to access the online documentation and help system:

- You can choose particular topics of the documentation by using the Help menu in *daVinci*.
- You can start with the overview page and browse through the documentation by clicking on the links.
- You can get the corresponding help page of any dialog window by pressing the help button inside this window.

Note: It may take some time after calling the help system to display the particular page in Netscape, so please be patient and do not press a help button more than one time.

Roadmap to the Online Documentation

The entry to the online documentation is the overview page. There you can find links to the seven chapters of the documentation:

1. **General Information**:
   Everything a starter needs to know about *daVinci*: Installation, licence, introduction, and concepts.
2. **User Interface**
   This chapter explains how to work with *daVinci's* Motif user interface. The help system refers to these pages, so usually there is no need to read the whole chapter. Instead, read the tutorial and use this chapter as a reference when questions or problems occur.
3. **Tutorial**
   The tutorial is a guided tour through *daVinci* V2.1.x to learn about all features.

4. **Questions & Answers**
   This chapter is a summary of solutions for frequently asked questions.

5. **New Features in *daVinci* V2.1.x**
   A list of modifications compared to previous release *daVinci* V2.0.x.

6. **Reference**
   Detailed information about particular aspects of *daVinci*: Term representation (*daVinci's* graph format), application programmer interface (API), PostScript interface, program options, tools available in the distribution, bibliography.

7. ***daVinci* in the Internet**
   Survey about important *daVinci* resources in the Internet: WWW-server, e-mail address, mailing list, bugreport, etc. Do not miss to read it!

Features of the Online Documentation

Each page of the documentation has a header with seven buttons which can be selected for direct access to frequently used pages. Click on these buttons to go to the corresponding pages:

Underneath these buttons, you can see the location of the particular page in the online documentation. By clicking on the links, you have direct access to parent pages. For example, here is the location of this page (also displayed at the header of this page above):

Overview -> General Information -> Using the Online-Documentation

Furthermore, the overview page has some additional functionality by clicking on the folder icons:

Click it to open the folder. This shows the content of the corresponding chapter.

Click it to close an opened folder.

Click it to open all folders in one step. Another click on this icon undos this operation by closing all folders.

The index page features a search engine to find any keyword in the online documentation. The search engine is written in Java and requires a Java compliant WWW browser such as Netscape 2.0 or higher. Instructions for using the search engine can be found on the index page.

---

daVinci V2.1 Online Documentation - Page update: June 15, 1998

Overview | Intro | Reference | Tutorial | Questions & Answers | New Features | Index

Overview -> General Information -> *daVinci's* concepts

daVinci's Concepts

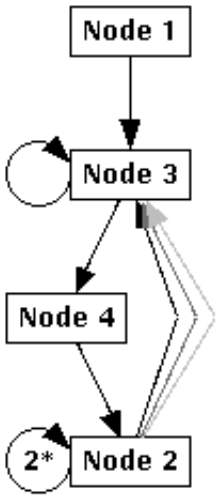
This document introduces to basic concepts and technical background informations:

- Graphs
- Applications using *daVinci's* API
- Graph Layout
- Visualization definition
- Fine Tuning of a Layout
- Graph Status
- Interactions
- Multi-Graph / Multi-View
- Drawing Technology and Pixmap Caching
- Drag and Drop

Graphs

daVinci is an interactive tool to visualize directed graphs. A graph is a structure with a number of objects (*nodes*) and relationships between them (*edges*). For directed graphs, all edges have a direction, i.e. for each edge there is a parent- (the source) and a child node (the target). The graph layout in *daVinci* reflects these hierarchical relationships by arranging the nodes at horizontal levels such that all parent nodes are above their child nodes and all edges point downwards (in a top-down layout). Further, the direction of an edge is usually visualized with an arrow pointing to the child node. This kind of representation is called *hierarchical visualization of a directed graph*.

Although hierarchical visualization is only possible for non-cyclic graphs, *daVinci* is also able to handle cyclic graphs with a trick: A cycle analysis inverts one edge of each cycle internally by exchanging parent and child node (i.e. the parent becomes the child of the edge and vice versa). This way, a cyclic graph is transformed into an acyclic one which can be handled by the hierarchical layout algorithm. Afterwards, the arrows of inverted edges will be drawn at the opposite "wrong" side (i.e. they point to the parent) to invert the direction for a second time such that it will point to the original direction in the visualization.



Hierarchical visualization of a directed cyclic graph with self- and multi-edges.

Graphs are loaded in *daVinci* using a format called *term representation* which is explained in the reference. **Note:** Files containing a *daVinci* term representation should always have suffix *.daVinci*. The term representation format supports all kind of directed graphs: cyclic or acyclic graphs, empty graphs, graphs with only one level (a list of nodes without any edges), multi-edges (two or more edges between two nodes) or even self-edges (edges where the parent and child node are the same). The term representation is a plain text ASCII format, so *daVinci* graphs can even be created with an arbitrary text editor. But normally, one will not do this by hand. Instead, graphs are usually generated automatically by some application program.

Applications using daVinci's *API*

When there is a need to edit a graph by hand, users can connect an external graph editor application to do this interactively based on the visualization of a graph. The grapheditor is an external program that communicates with *daVinci* by using the API (Application Programmer Interface). The editor (or any other *daVinci* application) is responsible for exclusively controlling and manipulating the structure of a graph.

At start time, the editor adds some menus to *daVinci's* user interface, e.g. to insert or delete nodes and egdes or to change their attributes. Application menus are attached to the Edit menu, but there is no function behind these menus in *daVinci*. Instead, the API sends menu events to the graph editor to inform about interactions. For example, when the user selects a node (1. event) and chooses the previously attached menu 'Edit/Delete Node' (2. event), then the editor knows by these two events that the user would like to remove the specified node. The editor has its own graph data structure which needs to be modified by the editor to reflect the events. After modification, the editor will send the actual graph back to *daVinci* to update the visualization.

This principle can be used by any application program that is connected to *daVinci's* API. Note that the application is exclusively responsible for controlling the graph structure. *daVinci* is not able to modify the graph on its own. This is important when *daVinci* is the graph user interface on top of an arbitrary application program. On the other side, by separating the representation layer (*daVinci*) from the control layer (application connected to the API), the application programmer is relieved from dealing with graph layout algorithms and computer graphics, so he/she can concentrate on his/her true work.

Graph Layout

A graph is immediately displayed in a window after loading a term representation from file. For a hierarchical layout, nodes of the same hierarchy layer are assigned to one horizontal level such that all edges point downwards. This is even true for cyclic graphs (see above). Edges passing more than one level (*long span edges*) are filled up with *dummy nodes* at each passed level. Dummy nodes are not part of the term representation, they are introduced automatically. Usually, they are invisible, but highlighted with small bullets when the corresponding edge is selected.

After loading a graph, a simple layout algorithm will be applied without processing an edge crossing minimization. That means all child nodes of a graph are drawn in the initial order found in the term representation. An edge crossing minimization is an expensive, lengthy operation and therefore need to be started by the user with menu Layout/Improve All. This operation calculates a new graph layout with (probably) fewer edge crossings by rearranging the order of nodes at each level. Afterwards, a bend minimization phase will reduce the number of edge bends. The graph layout algorithm of *daVinci* is based on the heuristical approach of Sugiyama et al. ([STT81], [SM91]) and was refined by a non-deterministic component to get better results (discussed in [FW94b], [Frö97]). Due to this random component, two layout calculations of the same graph may yield different results. The advantage of this technique is that the average number of crossings is usually smaller.

For all trees (i.e. graphs where all nodes have only one parent), there is a more efficent tree layout algorithm implemented in *daVinci*. This algorithm with linear complexity is based on the work of Juutistenaho ([Juu94]) and was refined to pass the tree only one time. *daVinci* automatically selects the appropriate layout algorithm, depending on whether the graph is a tree or not.

In *daVinci* V2.x, it is possible to set the orientation of a graph layout. The default orientation is *top-down* to get horizontal levels and edges pointing downwards. The other three layout orientations *left-to-right*, *right-to-left* and *bottom-up* can be set with submenu Layout/Orientation.

After calculating the layout, the graph will be drawn in a window. The visualization of nodes and edges (e.g font, color, shape, icon, line pattern, etc) is determined by their attributes in the term representation. A string, taken from attribute **OBJECT**, is placed inside each node, whereby the nodes size is adapted to match the actual extension of the text. Multi-line text is also supported. For the layout, the minimal distance between nodes at the same level and the distance between levels (called gap width and height) can be set with menu Options/Layout Dimensions....

If a node is selected by the user with the mouse, it will be drawn with a shadow. For visualizations in a reduced scale, selected nodes are only drawn inverted. Many nodes may be selected at a time. Edges can be selected, too, but only one at a time (i.e. multiple selection for edges is currently not possible). Edge selection is a useful trick to trace the route and find the parent/child node of very long edges.

Incremental Graph Layout

If the layout algorithm is started by selecting the menu Layout/Improve All, *daVinci* will try to find the "best" (please remember, that the layout algorithm is a heuristic) layout for the graph. This may result in major layout changes after small changes in the structure of the graph (i.e. graph updates sent via the API). Therefore Michael Fröhlich introduced incremental graph layout in his PhD ([Frö97]). All results of this work are integrated in *daVinci* V2.1.

Graph updates sent to the daVinci API are now inserted into the existing graph layout incrementally instead of forcing a completely new layout. So the layout before the update is preserved as much as possible in order to support layout stability. Of course, after doing some incremental updates, the graph layout may be getting worse, so a new global graph layout might be required. The user can do this at any time with menu Layout/Improve All.

This new feature isn't only used for API updates, but also after executing or undoing abstraction operations, for vertical fine tuning, or for the new functions Layout/Improve Edges and Layout/Improve Selected Nodes.

Visualization definition

The rules of a visualization definition define the visual appearance of a family of graphs. For example, the user can specify a rule to draw all nodes of some type X with a red circle, instead of using attributes for each individual node of the graph. The modification of a rule affects all elements of the same node type in a graph class at the same time. So, the central visualization definition is a comfortable method to define a universal visualization for a family of graphs. It is possible to define visualization rules which are the elements of a visualization definition for each node and edge type and it is even possible to overwrite the *daVinci* default values for attributes. Please refer to API command category Visual for technical informations.

Note: In future versions of *daVinci* these visualization definitions should be loadable from a file. In the current version V2.1 visualization definitions can only be set and changed by API commands of category Visual.

Attribute evaluation order

If no visualization definition is present, an attribute is either set for each node or the default value is taken. If a visualization definition is present for the current graph, the attribute evaluation order is modified. First the value of the attribute of the individual node or edge is considered, if this is not set, the corresponding rule for the nodes or edges type is retrieved. If a rule is found and the attribute is defined in the rule, its value is taken, otherwise the default value is used.

If someone has many graphs with attributes defined for each node and edge from former versions of *daVinci*, it would be impossible to use the new visualization definitions with these graphs. Therefore the attribute evaluation order can be controlled by setting API command set(rules_first(true)). In this case the visualization rules are considered before the individual node or edge attributes and before the default values.

Fine Tuning of a Layout

An automatically generated layout of a graph can never be perfect. To overcome this deficite in *daVinci*, a user is able to improve the visualization of a graph with fine tuning operations. Fine tuning is available at any time, before or after reducing edge crossings with the layout algorithm (menu Layout/Improve All). Fine tuning is even useful to remove obvious edge crossings, left by the layout algorithm. Afterwards, the layout algorithm can be started again to try eliminating even more crossings. The following description is written for fine tuning in a *top-down* layout, for other orientations you have to exchange "horizontal" and "vertical" as necessary. Fine tuning for other orientations than *top-down* has been introduced in *daVinci* V2.1.

Fine tuning is done by selecting a node with the left mouse button. The button need to be pressed while the mouse is moved to a new position (*dragging*). Both horizontal and vertical fine tuning is implemented in the current release. Regular nodes can be moved in both directions, but the invisible dummy nodes can only be moved horizontally on a level. Fine tuning in *daVinci* is more than just trivial adjustment of a node's position. Moving a node may have an immediate influence on the rest of the graph layout. For example, if a node is moved horizontally and will touch its left or right neighbour node, then either both nodes have to exchange their position (*modify node order*) or the neighbour node has to be moved as well (*preserve node order*), depending on the current fine tuning mode (see below). Furthermore, if a node is moved to the next or previous level, then the child- or parent subgraphs may need to be moved as well to keep the layout hierarchically (i.e. all edges have to point downwards).

For horizontal fine tuning (i.e. moving a node on a level to the left or to the right), there are three modes that can be set with menu Options/Layout Algorithm.... A more comfortable way to set the fine tuning mode is by using the last three icons of *daVinci's* icon bar. The fine tuning mode defines the behaviour when a moved node touches its left or right neighbour node at the same level. A neighbour node is touched when the minimal node distance is reached by the mouse pointer.

- In default mode *Preserve Node Order*, the touched node is moved as well to keep the order of nodes on a level.
- In mode *Modify Node Order*, the moved node and the touched node will exchange their position as soon as there is enough space to do so. This mode is useful to modify the order of nodes on a level, e.g. to remove edge crossing by hand.
- In the third mode *Fixed Neighbour Nodes*, fine tuning will stop as soon as a neighbour node is touched. This is useful to protect the position of all neighbour nodes.

Vertical fine tuning was first introduced as a new feature in *daVinci* V2.0. This operation allows the user to move a node to a different level. Vertical fine tuning only works with regular nodes, dummy nodes cannot be moved vertically. A dragged node is moved to the next or previous level as soon as the level coordinate (i.e. the center of all nodes at this level) is reached by the mouse pointer. At this time, dynamic layout may automatically start to rearrange the layout. This is needed to keep a layout hierarchically.

Important notes about fine tuning:

- After starting the layout algorithm, the x-coordinates and order of all nodes will be recalculated, but their current level will be kept. This preserves the vertical fine tuning, but eliminates the horizontal. If you want to place the nodes back to their default level, you have to deselect checkbox **Keep Nodes at Levels** in the Options/Layout Algorithm... dialog.
- Modifying the layout dimensions with the Options/Layout Dimensions... dialog eliminates previous horizontal fine tuning operations. So, the final dimensions should be set before starting with extensive fine tuning.
- In *daVinci* V2.1 you'll find no more left and right borders for fine tuning, as in previous versions, where you couldn't enlarge the visualization during fine tuning. But please note, that unnecessary space to the left and the right of the visualization is removed immediately after you've released the mouse button.

Graph Status

A graph status is the alternative external format for graphs in *daVinci* (the other one is the term representation, see above). Files containing a *daVinci* status should always have suffix *.status*. Beside the pure graph structure of a term representation, a status also contains the graph layout informations and mostly all current settings of the user interface. So, a user is able to restore a previous session by loading a status file. A status file contains informations about one graph only (i.e. the graph of the base window where the status was saved). Multi-graph and multi-view are not considered in a graph status.

Note: A status file contains information that is specific for a particular *daVinci* release. Usually, the data stored in status files is extended with each new release, to consider new features. This means that a particular *daVinci* release, for example V1.4.x, cannot read a status file that was generated by a future release, e.g. V2.1.x. But the status files of earlier releases can always be read in future releases by using a compatibility mode. You can determine the release of a status file by looking at the beginning of the file with the *more* command.

Interactions

daVinci offers some very useful interactive operations to work with a graph visualization. Some of them, namely selection and fine tuning, have already been discussed above.

Scaling (menu View/Scale...) can be used to get an overview of a graph that does not fit in a window. The scale of a window can be set to any rate between 1% and 100%. In reduced scales smaller than 100%, some details are not shown (e.g. the text of nodes), but all operations are still available. There is a very efficient implementation for scaling in *daVinci*. Further, scaling does not affect the layout, so do not hesitate to change the scale frequently in a session.

Abstraction operations (menu Abstraction) are implemented to hide uninteresting or confusing details of a graph for some time. At the moment, two abstractions are available: Subgraph hiding and edge hiding. By hiding a subgraph, all child nodes are removed from the visualization that can only be reached (via edges) from the selected node. The other abstraction, edge hiding, removes all incoming and outgoing edges of a node from the visualization. Both abstractions can be undone with appropriated menu functions.

Some useful *navigation* and *find* features are available in menu Navigation. With the navigator, a user can browse in a graph by moving to the nearest neighbour- or relative node of a currently selected node. Selection functions show the parents, siblings or children of a selected node. Finally, a find function can be used to search in a graph for nodes with a particular text.

Multi-Graph / Multi-View

daVinci V2.x has built-in support for *multi-view* and *multi-graph*. These features offer the opportunity to work on the same graph in different coupled windows (views) or to load many graphs in several uncoupled windows at the same time.

The user is able to open an additional view of a graph from any base window by using menu View/Open New View. This opens a second coupled window showing the same graph. Interactions in one view of the graph (e.g. fine tuning) manipulate the other views, too. One exception is the scale, so each view may have its own scale. The current fine tuning mode is also set independently for each view.

A special view with additional functionality is the *survey view* which can be opened from any base window (called *detail view*) with menu View/Open Survey View. A survey view shows the graph of the detail view in a reduced scale such that it is completely visible, even if the window size is changed for the survey view. The visible area of the graph in the detail view is visualized in the survey view with a continuously updated blue rectangle. When the user selects a node in the survey view, *daVinci* will scroll to this node in the detail view using animation. This behaviour can be switched off in dialog Options/General Settings... by deselecting checkbox **On selection in Survey View**.

A second, independent graph can be loaded with menu File/Open... in each view. The new graph is only displayed in the view where the load operation was triggered and not in the other views. So technically, a view of a graph is disconnected from the other views of the same graph as soon as a new graph is loaded there. For example, when you have only one window (i.e. no views) and would like to visualize a second graph simultaneously, then open a view to get a second window and load the new graph there. The maximal number of opened base windows in *daVinci* for multi-view and multi-graph is 64.

Drawing Technology and Pixmap Caching

The drawing functions of *daVinci* V2.x are designed to reuse already drawn parts of a graph visualization. Therefore the canvas of a graph is partitioned into a grid with segments of 1000x1000 pixel. For each grid segment there is one pixmap for drawing the nodes and the edges. When a graph segment becomes visible for the first time, the corresponding nodes and edges will be drawn on the segment's pixmap. Afterwards, the graph will be displayed by copying the pixmap or parts of it into the window. By drawing and allocating pixmaps dynamically on demand, *daVinci* is able to handle very large graphs with superior performance.

The pixmaps of all drawn segments are kept in a memory cache as long as the scale of the visualization is not modified and the graph layout does not need to be recalculated. Pixmap caching accelerates the scrolling performance when the user scrolls back into previously visible areas of the graph. The pixmap cache can be switched off with dialog Options/General Settings.... This should be done when problems with not enough memory occur (*daVinci* will crash with a *X Error: BadAlloc* message as soon as no more memory is available).

Drag and Drop

For a better support of user interaction we have added the drag and drop functionality to the *daVinci* API. With the new commands of API command category Drag and Drop you can set the API into drag and drop mode. This switches off the two dimensional scrolling feature, you achieve normally by pressing the middle mouse button (BTransfer). Then you can click with the middle mouse button somewhere in the *daVinci* window and the API is informed (create node), or you can select a node with the middle mouse button and drag an edge from this node to another node. If the mouse button is over a node during dragging, the node is inverted to inform the user about the secondary selection. The application is informed if you release the button over a node (create edge) or over the free area (create node and edge). If you select a node with the middle mouse button and the Shift-Key pressed, you can drag a node and drop it somewhere over another node, even in another window or context (drop node). Please refer to API command category Drag and Drop for technical informations. You can test this feature by using the example application grapheditor contained in the *daVinci* distribution. Please follow the steps in the grapheditor tutorial.

daVinci V2.1 Online Documentation - Page update: June 15, 1998

daVinci V2.1.x License

IMPORTANT:
Please read this license carefully before using this software. By using, copying or redistributing *daVinci*, you accept and agree to become bound by the terms of this license. Please do not use this software, if you do not accept the terms of this license.

The interactive graph visualization system *daVinci* V2.1.x (hereinafter refered to as "*daVinci*") has been developed by the *daVinci* project at Bremen Institute for Safe and Secure Systems, Universität Bremen, Germany (hereinafter refered to as "*daVinci* project"). The *daVinci* project grants a "right to use" license of *daVinci* to any user of this software (hereinafter refered to as "licensee") for use under the following conditions:

RIGHT TO USE LICENSE

Permission to copy the system, to redistribute it, and to use it for any purpose is granted, subject to the following restrictions and understandings:

1. **Grant of License**
   daVinci is licensed free of charge for non-profit or internal use. Any commercial exploitation of *daVinci*, as a standalone program, as part of a more comprehensive system or in any other constellation, is prohibitted by this license. Please contact the *daVinci* project if you are interested in a commercial license of *daVinci* to use it in case of for-profit applications.
2. **Redistribution**
   daVinci can only be redistributed for free or as part of a free, non-profit software. Any copy made of *daVinci* must cover the complete distribution, especially this license in full, the complete documentation, all of the tools and examples.
3. **Restrictions**
   This license does not permit the licensee to alter, modify, improve, decompile, disassemble or otherwise reverse-engineer the obtained object code binaries of *daVinci*. Especially, the licensee shall not remove any proprietary copyright legend from *daVinci* or any part thereof.
4. **Obligations**
   The licensee agree to make his/her best efforts to inform the *daVinci* project about bugs and noteworthy applications of this system.
5. **Acknowledgement**
   All materials, developed as a consequence of *daVinci*, shall duly acknowledge such use, in accordance with the usual standards of acknowledging credit in academic research.
6. **Disclaimer of Warranty**
   The licensee expressly acknowledge and agree that use of *daVinci* is at the sole risk of the licensee. *daVinci* and related documentation are provided "as-is". The *daVinci* project makes no warranty or representation, either express or implied, for *daVinci*, including, but not limited to, any implied warranties of merchantability and fitness for a particular purpose. The *daVinci*

project does not warrant that *daVinci* will meet the licensees requirements, or that the operation of *daVinci* will run uninterrupted or error-free, or that defects in *daVinci* will be corrected. The entire risk as to the functioning, faultless operation, quality, performance and effect on hardware or other software of *daVinci* is with the licensee. Should *daVinci* prove defective, the licensee assumes the cost of all necessary servicing and repair of correction.

7. **Limitation of Liability**
   Under no circumstances, including negligence, unless required by applicable law, shall the *daVinci* project be liable for any direct, indirect, special, incidental, or consequential damages arising out of the use of or inability to use *daVinci*, even if the *daVinci* project has been advised of the possibility of such damages. Specifically, the *daVinci* project is not responsible for any costs including, but not limited to, those incurred as a result of lost profits, revenue or other economic loss, loss of use of the product, loss of data, the costs of recovering such software or data, the cost of substitute software or data, claims by third parties or for other similar costs.
8. **Disclaimer of Support**
   The *daVinci* project is under no obligation to provide any services, by way of maintenance, consultation, update, or otherwise.
9. **Non-Exclusiveness**
   The licensee recognises that *daVinci* is released on a non-exclusive basis and the *daVinci* project shall have the exclusive right to grant licenses to others or to make such other use of *daVinci* as it shall desire.
10. **Effectiveness and Termination**
    This license is taking effect after using, installing, copying or redistributing *daVinci* by the licensee. By using, installing, copying or redistributing *daVinci*, the licensee accepts and agrees to become bound by all terms of this license. The license is effective until terminated. The licensee may terminate this license at any time by destroying all components of *daVinci*, related documentations and all copies thereof. This license will terminate immediately without notice from the *daVinci* project if the licensee fail to comply with any provision of this license. Upon termination, the licensee must destroy all components of *daVinci*, related documentations and all copies thereof.

Bremen, July 10th, 1998 - The *daVinci* project

---

daVinci V2.1 Online Documentation - Page update: July 10, 1998

Platform Requirements and Installation

Platform Requirements

daVinci V2.1.x is available for UNIX platforms which are supported by X-Window (X11R5 or higher) and the Tcl/Tk toolkit. At the moment there are binary distributions for the following operating systems:

- Sun Solaris 2.x SPARC
- SunOS 4.1.x
- HP-UX
- IBM AIX 4.2
- SGI IRIX
- Linux (SuSe and RedHat4, RedHat5)
- DEC Alpha OSF/1

Please read the distribution notes for detailed informations about the particular *daVinci* distributions. If you miss a UNIX platform for which *daVinci* is not available at the moment, feel free to contact the developers for asking about a port. The source code of *daVinci* is not available.

The minimal hardware requirement for working seriously with *daVinci* is a UNIX workstation with a performance comparable to a Sun SPARCstation 10 or better (i.e. at least 1 SPECint95) and 32MB RAM. For PC's, an Intel 486 and 16MB or better is recommened. It is possible that *daVinci* will work on slower configurations or with fewer memory, but in this case the performance may be disappointing. If you have problems with not enough memory (*daVinci* will crash with a *X Error: BadAlloc* message as soon as no more memory is available), you should try to switch off the pixmap cache by starting *daVinci* with option -nocache.

The software requirement for *daVinci* is X11 Release 5 (X11R5) or higher. The user interface is implemented in Tcl/Tk, but it is not required to install Tcl/Tk in order to use *daVinci*. The system has been tested with the following X11 window managers:

- dtwm (CDE)
- mwm (Motif)
- olwm/olvwm (OpenWindows)
- fvwm/fvwm2
- ctwm
- twm

Installation

Overview:

1. Get *daVinci* by ftp for a particular platform
2. Use gunzip, tar -xvf
3. Set $DAVINCIHOME to the *daVinci* directory
4. Be sure to have Netscape 2.0 or higher for the online documentation
5. Start *daVinci*
6. Read and accept the license

Detail:

After downloading the *daVinci* distribution for the particular UNIX platform by ftp, the retrieved file need to be uncompressed with command *gunzip <file.tar.gz>* and afterwards extracted by using command *tar -xvf <file.tar>*. This will create a directory *daVinci_V2.1* in your current directory. To place *daVinci* at another location, you need to move the *daVinci_V2.1* directory completly to some other directory of your filesystem.

Before starting *daVinci*, you have to set the environment variable $DAVINCIHOME to the absolute file name of the *daVinci_V2.1* directory on your computer. *daVinci* needs to know this location to find resources, e.g. the online documentation or the graph editor. Be careful with changing the location of the *daVinci* distribution after users of *daVinci* have saved their options, read the note in chapter Reference -> Command-Line Options -> $DAVINCIHOME for details. For example, if you have installed *daVinci* in your home directory */home/me*, then the environment variable can be set using the following command:

For csh users:

```
%setenv DAVINCIHOME /home/me/daVinci_V2.1
```

For sh users:

```
%DAVINCIHOME=/home/me/daVinci_V2.1; export DAVINCIHOME
```

You should make these settings permanent by putting this command into file *~/.cshrc* (for csh users) or *~/.profile* (for sh users). Finally, make sure that you have installed Netscape 1.1 or higher to work with the online documentation. For the search engine, written in Java, you need to have at least version Netscape 2.0 of the Netscape WWW browser.

After moving to the *daVinci_V2.1* directory with *cd*, you can start the system by entering command *daVinci* in the shell. For the first time a user is starting *daVinci*, a window with the license agreement will appear. Please read the terms of the agreement carefully. You can get the full text of the license by pressing the **Show License** button. The license text is also available in file *LICENSE* in the *daVinci_V2.1* directory. By confirming the license window, you accept and agree to become bound by the terms of this license. Please press the **Quit** button in the license agreement window, if you do not agree to this license.

By confirming the license agreement, a notification window will appear to ask the user about sending an e-mail to the *daVinci* project. This notification is optional. Afterwards, daVinci can be used and Netscape will be started "automagically" to show the introduction page of the online documentation.

daVinci V2.1 Online Documentation - Page update: June 15, 1998

Overview -> User Interface

daVinci - Online Documentation V2.1

User Interface

This chapter covers information about the *daVinci* user interface. The following subtitles are available:

- **Using the Mouse**
- **Using the Keyboard**
- **Base Window**
- **Pop-Up Menu**

- **File Menu**
- **Edit Menu**
- **View Menu**
- **Navigation Menu**
- **Abstraction Menu**
- **Layout Menu**
- **Options Menu**
- **Help Menu**

daVinci V2.1 Online Documentation - Page update: June 15, 1998

Using the Mouse

In *daVinci* V2.x, the three buttons of the mouse are used as follows:

- The *left* mouse button (**BSelect**) is used for selection.
- The *middle* mouse button (**BTransfer**) is used for 2D-scrolling and Drag&Drop mode.
- The *right* mouse button (**BMenu**) is used for menu operations.

(The **B**-names are the OSF/Motif notation for mouse buttons.)

BSelect (Left Mouse Button)

In Motif applications, this mouse button is used for selection, activation, and setting the location cursor. In a *daVinci* graph area, this button can also be used to select nodes and edges. Many nodes may be selected at the same time, but only one edge. After making a new selection, the old one is lost. Many nodes can be selected in one step by using rubber-banding. To do so, click on empty space in the graph area with *BSelect* and hold down this mouse button. Then move the mouse by still pressing *BSelect*. You can see a moving rectangular area on the screen (*rubber-band*). After releasing the mouse button, all nodes are selected that lie completely inside this rectangle.

A user can extend or reduce the current selection by using the *Shift*-Key and *BSelect* simultaneously. By holding down *Shift* and selecting a node with *BSelect*, you add or remove this node from the set of currently selected nodes, depending on whether or not the node was selected before. To say it in another word: With *Shift*, the old selection will not be lost after a new selection. Shift-selection also works in combination with rubber banding.

Note for users familar with *daVinci* V1.x: Extension and reduction of a current node selection was implemented in *daVinci* V1.x to be used with the middle mouse button instead of the *Shift* key. This has been changed for *daVinci* V2.x due to transition from OPEN LOOK to OSF/Motif.

Fine-tuning of a graph layout also works with *BSelect*. Dragging a node (i.e. holding down *BSelect* while moving the mouse) can be used to move it to a new position to improve the overall layout.

BTransfer (Middle Mouse Button)

In Motif applications, this mouse button is used for moving and copying elements. Due to its fundamental concepts, *daVinci* does not offer any built-in capability for editing, so copy operations does not make sense. This way, *BTransfer* is used to move (i.e. scroll) the visible part of a graph area in a base window. By pressing *BTransfer* anywhere in *daVinci's* graph area, the window will scroll in both dimensions at the same time (*2D-scrolling*). When the mouse is moved while pressing *BTransfer*, the visible area of the window will directly follow these movements. 2D-scrolling is also available in the directory and file views of the file browser window.

An application can switch on Drag and Drop mode. In this case, the 2D-scrolling is switched off and the button *BTransfer* is used for Drag and Drop of nodes and edges. This can only happen, if an application is connected. It is recommended, that the application indicates the mode change in an appropriate manner.

BMenu (Right Mouse Button)

In Motif applications, this mouse button is used for poping up menus. With *BMenu*, the user is able to select from the pull down menus of *daVinci's* base window by dragging through menus and submenus and releasing the mouse button above the desired menu entry.

If the mouse pointer is over a node or an edge and *BMenu* is pressed, the value of the node pop-up menu or the edge pop-up menu attribute is considered and if there is a menu defined, the corresponding popup-menu is displayed. If none is defined, the regular pop-up menu is displayed.

BMenu, pressed in the graph area, shows the pop-up menu for direct access to frequently used menu functions.

---

daVinci - Online Documentation V2.1

Using the Keyboard

Input from the keyboard (i.e. mouseless operation) is both supported in the base window and in all *daVinci* dialog windows, e.g. the file browser. In a base window, the keyboard can be used for quick access to menu operations. In dialog windows, the keyboard is used to manipulate the dialog elements of the window.

Keyboard Usage in the Base Window

In an active *daVinci* base window (to get an active window, you either need to select a window or place the mouse pointer above a window), there are two different ways to call menu operations with the keyboard instead of using the mouse: Mnemonics and accelerators. All menus have mnemonics, but only for the frequently used menus there are accelerators.

A **mnemonic** is the underlined character of each menu. For example, menu *File* has an underlined mnemonic **F**. The menu entry '*Open...*' in the *File* menu has mnemonic **O**. So, to open a new graph, you can type shortcut **Alt+F O** in a base window instead of selecting menu '*File/Open...*' with the mouse. **Alt+F O** means: Press the **Alt** and **f** key simultaneously, then release both keys and press **o**. Do not use upper case **F** and **O** with the **Shift** key! So the **Alt** key is only used to enter a menu, **Alt** is not needed for the entries in a menu. After entering a menu, you can also navigate through the menus by using the four cursor keys of the keyboard.

Mnemonic Keys to access Menus:

Alt+F:	Enter File Menu.
Alt+E:	Enter Edit Menu.
Alt+V:	Enter View Menu.
Alt+N:	Enter Navigation Menu.
Alt+A:	Enter Abstraction Menu.
Alt+L:	Enter Layout Menu.
Alt+O:	Enter Options Menu.
Alt+H:	Enter Help Menu.
F10:	Enter first menu (File Menu).
Escape:	Leave menus (cancel).

Some frequently used menu operations have an **accelerator** key which calls the particular function immediately without going through the menus. E.g. to open a file, press the **Ctrl** key and the **o** key (lower case!) simultaneously.

Accelerator Keys to call Menu Operations directly

Ctrl+O: Open a graph or status file.
Ctrl+S: Save a graph or status file.
Ctrl+P: Print a graph.
Ctrl+C: Close a base window.
Ctrl+X: Exit *daVinci*.
Ctrl+N: Open New View to a graph.
Ctrl+L: Full Scale.
Ctrl+T: Fit Scale to Window.
Ctrl+F: Find a node.
Ctrl+A: Find Again for a previous Find.
Ctrl+I: Improve All, start layout algorithm.

daVinci's Navigator for browsing a graph can also be accessed from the keyboard using the four cursor keys:

Cursor Keys to navigate in a Graph

CursorLeft: Go to left neighbour node (see Navigator).
CursorRight: Go to right neighbour node.
CursorUp: Go to nearest node at previous level.
CursorDown: Go to nearest node at next level.
Shift+CursorLeft: Go to nearest sibling node at left side.
Shift+CursorRight: Go to nearest sibling node at right side.
Shift+CursorUp: Go to nearest parent node.
Shift+CursorDown: Go to nearest child node.

Keyboard Usage in Dialog Windows

In an active *daVinci* dialog window (to get an active window, you either need to select a window or place the mouse pointer above a window), all dialog items can be manipulated from the keyboard instead of using the mouse.

Each dialog window has an active item which is the one currently manipulated with keyboard operations and highlighted with a black frame. After opening a dialog window, the active item is usually the leftmost push button. One can go to the next and previous item by using **Tab** and **Shift+Tab**.

Choosing the actual Item:

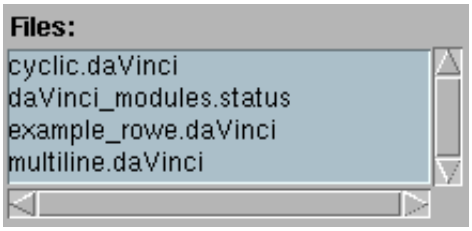
Tab: Go to the next item.
Shift+Tab: Go to the previous item.
Alt+F4: Close the dialog window (cancel).

To manipulate the different kinds of items in dialog windows, the following keyboard operations are supported:

Text Entry Fields:

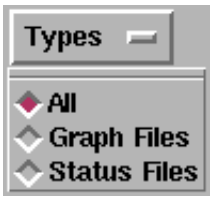
CursorRight:	Move cursor one character to the right.
CursorLeft:	Move cursor one character to the left.
Home:	Move cursor to the beginning of the text.
End:	Move cursor to the end of the text.
Shift+CursorRight:	Extend text selection to the right.
Shift+CursorLeft:	Extend text selection to the left.
Shift+Home:	Extend text selection to the beginning of the text.
Shift+End:	Extend text selection to the end of the text.

Lists:



CursorDown:	Select next list element.
CursorUp:	Select previous list element.
Ctrl+Home:	Select first list element.
Ctrl+End:	Select last list element.
PageDown:	Scroll downwards.
PageUp:	Scroll upwards.

Option Buttons

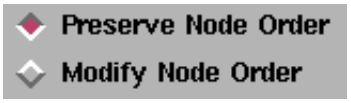


Return:	Open or close menu (by using current item).
Space:	Same as above.
CursorDown:	Go to next menu item.
CursorUp:	Go to previous menu item.
Escape:	Leave menu (cancel).

Sliders

CursorRight:	Increase slider value by one step.
CursorLeft:	Decreased slider value by one step.
Ctrl+CursorRight:	Increase slider value by several steps.
Ctrl+CursorLeft:	Decreased slider value by several steps.
Home:	Set slider to minimal value.
End:	Set slider to maximal value.

Radio Buttons



Return: Select or deselect button.
Space: Same as above.

Check Buttons

Return: Select or deselect button.
Space: Same as above.

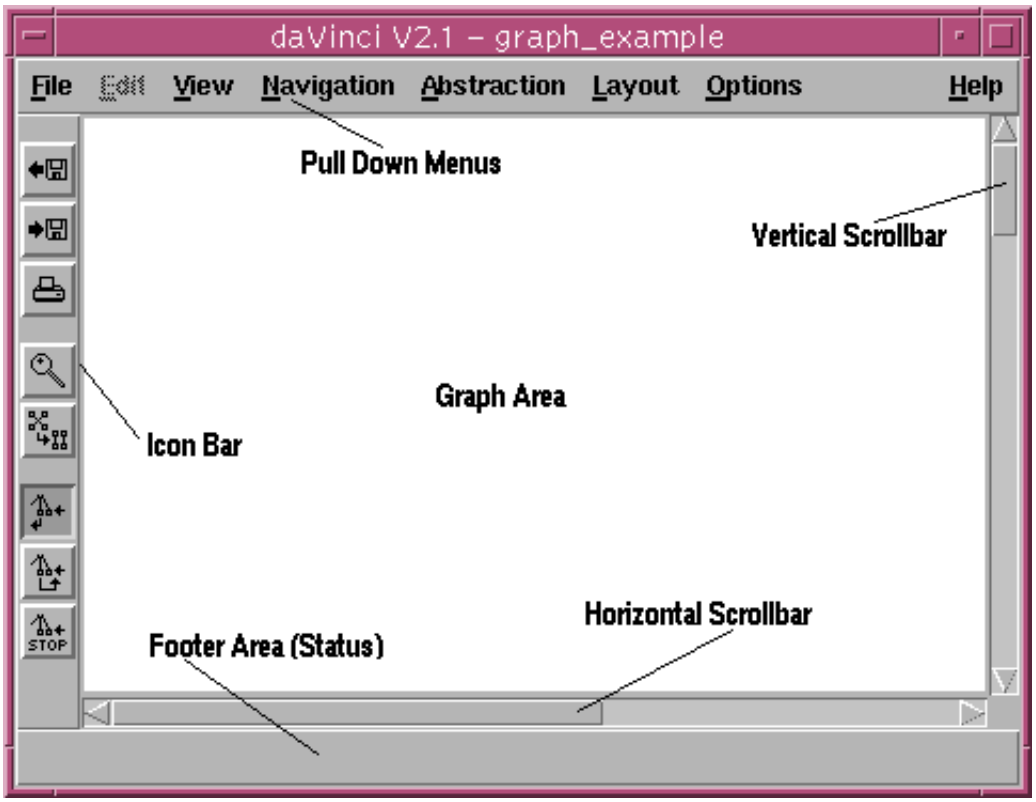
Push Buttons

Return: Select button.
Space: Same as above.

---

daVinci V2.1 Online Documentation - Page update: June 15, 1998

Base Window



The base window is the central place where graphs are visualized. *daVinci* is a multi-graph and multi-view environment, so in one base window, you can open a new view of the current graph (i.e. another base window) by using menu View/Open New View. When two or more base windows are open at a time, you can close one of them with menu File/Close without terminating *daVinci*. To exit from any base window, select menu File/Exit.

After loading a graph from file, it will be displayed in the **graph area** of the base window. If the graph exceeds the size of the window, the **horizontal** and **vertical scrollbar** can be used to adjust the currently visible area (2D-scrolling is possible by dragging with BTransfer, the middle mouse button). In the **footer area**, system messages are displayed, e.g. the status of an operation.

The **pull down menus** are used to start operations in *daVinci*. Menus can either be selected with the mouse or triggered by using the keyboard. The following menus are available in *daVinci* V2.x:

- File Menu
- Edit Menu
- View Menu
- Navigation Menu

- Abstraction Menu
- Layout Menu
- Options Menu
- Help Menu

The **icon bar** at the left side of each base window contains icons for direct access to some frequently used menu operations. *daVinci* V2.x offers the following icons:

File/Open...
File/Save As...
File/Print...
View/Scale...
Layout/Improve All
Options/Layout Algorithm.../Preserve Node Order
Options/Layout Algorithm.../Modify Node Order
Options/Layout Algorithm.../Fixed Neighbour Nodes

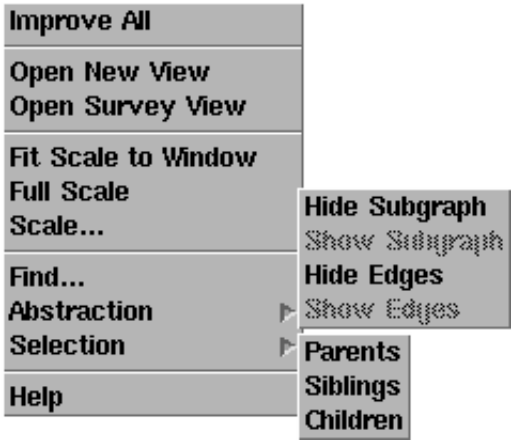
Beside these icons, the icon bar may have a second plane if an external application program is connected with *daVinci's* API. A *daVinci* application is able to add its own icons to the icon bar. Of course, these icons are specific for a particular application and cannot be discussed in this document. Application icons are shown on a second plane of the icon bar. The user can switch between the *daVinci* plane (with the icons shown above) and the application plane by selecting the small arrows (,) at the lower side of the icon bar.

Note: You can also use the keyboard in the base window for mouseless operation.

---

daVinci V2.1 Online Documentation - Page update: June 15, 1998

Pop up Menu

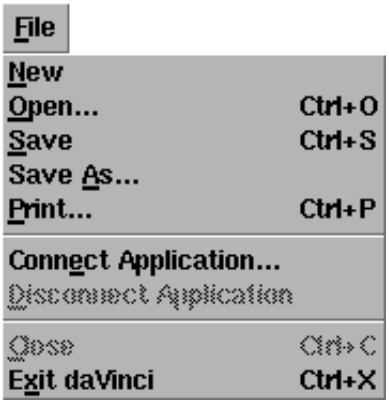


A pop-up menu is displayed if the mouse pointer is inside the graph area of a base window and BMenu (the right mouse button) is pressed. This menu is only displayed, if the mouse pointer is not over a node or edge or if the node pop-up menu or the edge pop-up menu attribute for this node or edge is undefined. Otherwise a context-sensitive popup menu is displayed. The general pop-up menu contains some frequently used menu operations for direct access in the graph area. You need to load a graph before getting a pop-up menu. The following operations are available in the pop-up menu. Click on the links to read about the corresponding operations which are also available in the pull-down menus of the base window.

- Improve All
- Open New View
- Open Survey View
- Fit Scale to Window
- Full Scale
- Scale...
- Find...
- Abstraction
  - Hide Subgraph
  - Show Subgraph
  - Hide Edges
  - Show Edges
- Selection
  - Parents (same as menu Navigation/Select Parents)
  - Sibling (same as menu Navigation/Select Sibling)
  - Children (same as menu Navigation/Select Children)
- Help (same as menu Help/Overview)

daVinci V2.1 Online Documentation - Page update: June 15, 1998

File Menu



The File menu contains operations for performing actions on files, such as opening, saving, and printing. It also contains operations for connecting an application to the API and for quitting.

Note: Menus can also be used mouseless with keyboard commands.

File and Printer Management

- **New**
  Clearing the graph area, i.e. removing the graph in the base window. In multi-view mode, when there are different views (base windows) to one graph, this operation only affects the base window where this operation was triggered and not the other views of the same graph Afterwards, the cleared view will be disconnected from the other views to act independently.

- **Open...**
  Loading a *daVinci* graph or graph status from file. After selecting this menu entry, a file browser will appear to specify the filename. In multi-view mode, loading a graph in one view will display the new graph only in the base window where this operation was triggered. Afterwards, this view will be disconnected from the other views of the former graph to act independently. This is called multi-graph mode, because this way different graphs are visualized at the same time.

- **Save**
  Saving the current graph status to a file for loading it in a later session. This stores the status directly without showing a file browser. The filename for the status is generated automatically, e.g. when graph *my_graph.daVinci* is loaded, the filename will be *my_graph.status* in the same directory. **Note:** This operation overwrites a file with the same name without asking the user to confirm.

- **Save As...**
  Saving the current graph or graph status to a file. After selecting this menu entry, a file browser will appear to specify the filename. Saving a status is used to restore the current session later, especially to preserve the layout of a graph.

- **Print...**
  Sending the current graph in PostScript format to a printer or storing it in a PostScript file. After selecting this menu entry, a print dialog window will appear to specify the parameters.

Application Management

- **Connect Application...**
  Connecting a *daVinci* application to the API. After selecting this menu entry, a file browser will appear to specify the filename of the application binary and the options. Read more about applications in the concept document. Such an external application program can use *daVinci* as a graphical user interface for graphs. One application is the graph editor which is part of the *daVinci* distribution.

- **Disconnect Application**
  Disconnecting a *daVinci* application which is currently connected to the API. Afterwards, the user will be able to connect another application. Disconnecting is only possible if the application supports this feature (i.e. it has to terminate after receiving answer disconnect from the API).

Closing Actions

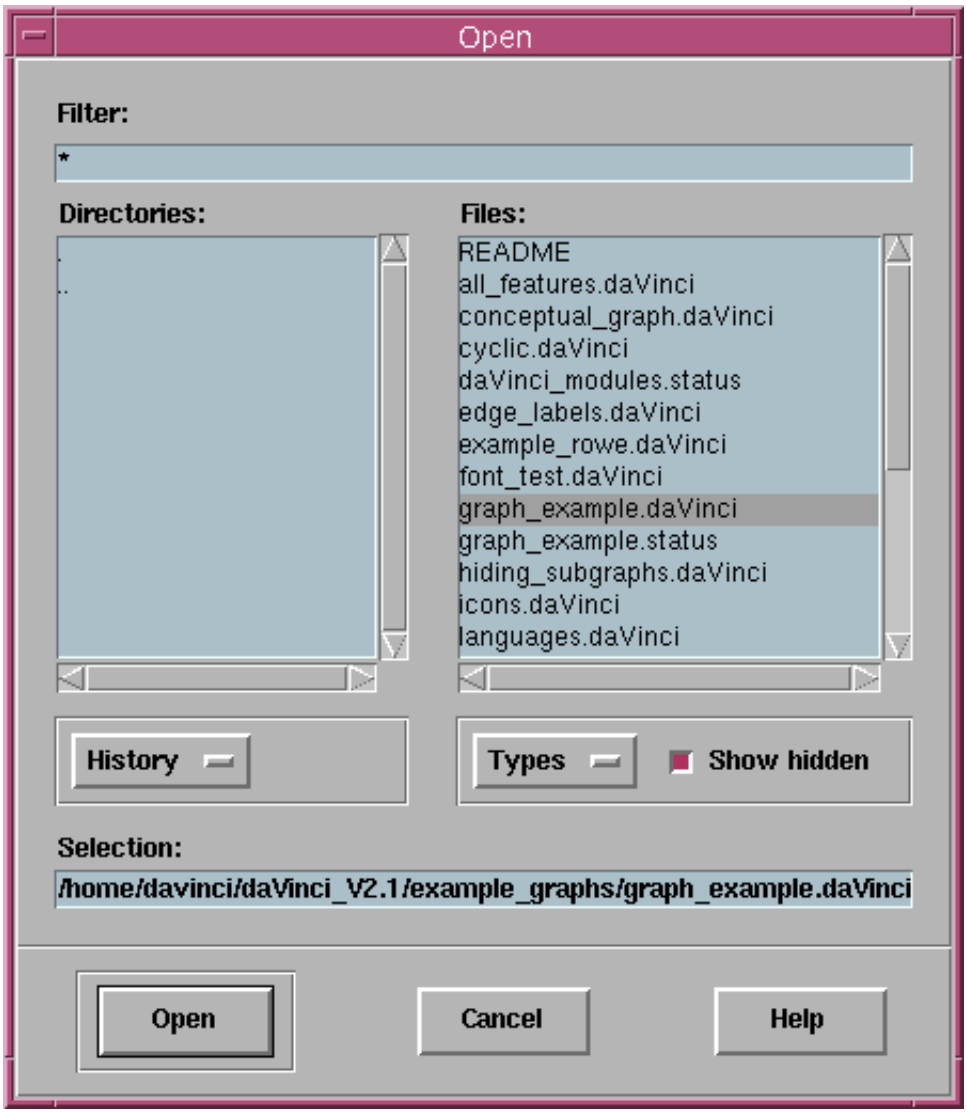
- **Close**
  Closing a base window, i.e. remove it from screen. In multi-view mode, when there are different views (base windows) to one graph, this operation only affects the base window where this operation was triggered and not the other views of the same graph. This operations is only available when more than two base windows are opened in a session. The last base window cannot be closed. So, this operation will never quit *daVinci*, but it may close the last view of a graph.

- **Exit**
  Exiting a *daVinci* session. **Note:** The user is not asked to confirm before quitting, so be sure to save the status for all graphs if you want to keep the layout for a later session.

Overview | Intro | Reference | Tutorial | Questions & Answers | New Features | Index

Overview -> User Interface -> File Menu -> Open...

File Browser to open Files



This file browser can be used to locate a graph or graph status file which shall be loaded and visualized in *daVinci*. Graph files are generated by the user or an application program (e.g. the graph editor) and should have the suffix *.daVinci*. They contain a term representation with the structure of a graph, but without layout informations (e.g. node coordinates). On the other side, a status file is always saved by *daVinci* itself and should have the suffix *.status*. A status contains the graph, the layout informations with the position of graph nodes and edges and most of the current user interface settings.

The browser has two views, one for the **Directories** and one for the **Files** in the current directory. The current directory is the one shown in the **Selection** field. To navigate through the directories, double-click on the elements in the **Directories** view. By clicking on '..', you move to the parent directory. To choose a file, either select it in the **Files** view and press the **Open** push button or simply double-click it. The browser beeps when you try to access a file or directory for which you do not have permissions, or when the specified file does not exist.

The file browser window can also be used mouseless with keyboard operations.

Dialog Items of the File Browser

- **Filter**
  Shows the filter expression currently applied to the **Files** view. The filter can either be selected with the **Types** option button or by manually typing an arbitrary shell expression here.

- **Directories**
  A list to show the directories inside the current directory (which is displayed in the **Selection** field). Double-click on the list elements to navigate through the directories. '..' moves to the parent directory.

- **Files**
  A list to show the files inside the current directory (which is displayed in the **Selection** field). Select one of the list elements and press the **Open** push button to load a particular *daVinci* graph- or status file. You can accelerate this by double-clicking a file.

- **History**
  Used to go back to a previously visited directory or to the home directory of the user.

- **Types**
  Used to reduce the number of elements in the **Files** view. You can choose to see all files (default), only the graph files or only the status files.

- **Show hidden**
  Select this check button to see hidden files and directories.

- **Selection**
  This text entry field shows the current selection which is modified when you browse through the directories. The browser usually suggests an appropriate initial selection. You can also edit the selection field by hand: If the entered selection is a directory, the browser will move to that directory, otherwise an open action is performed with the specified file.

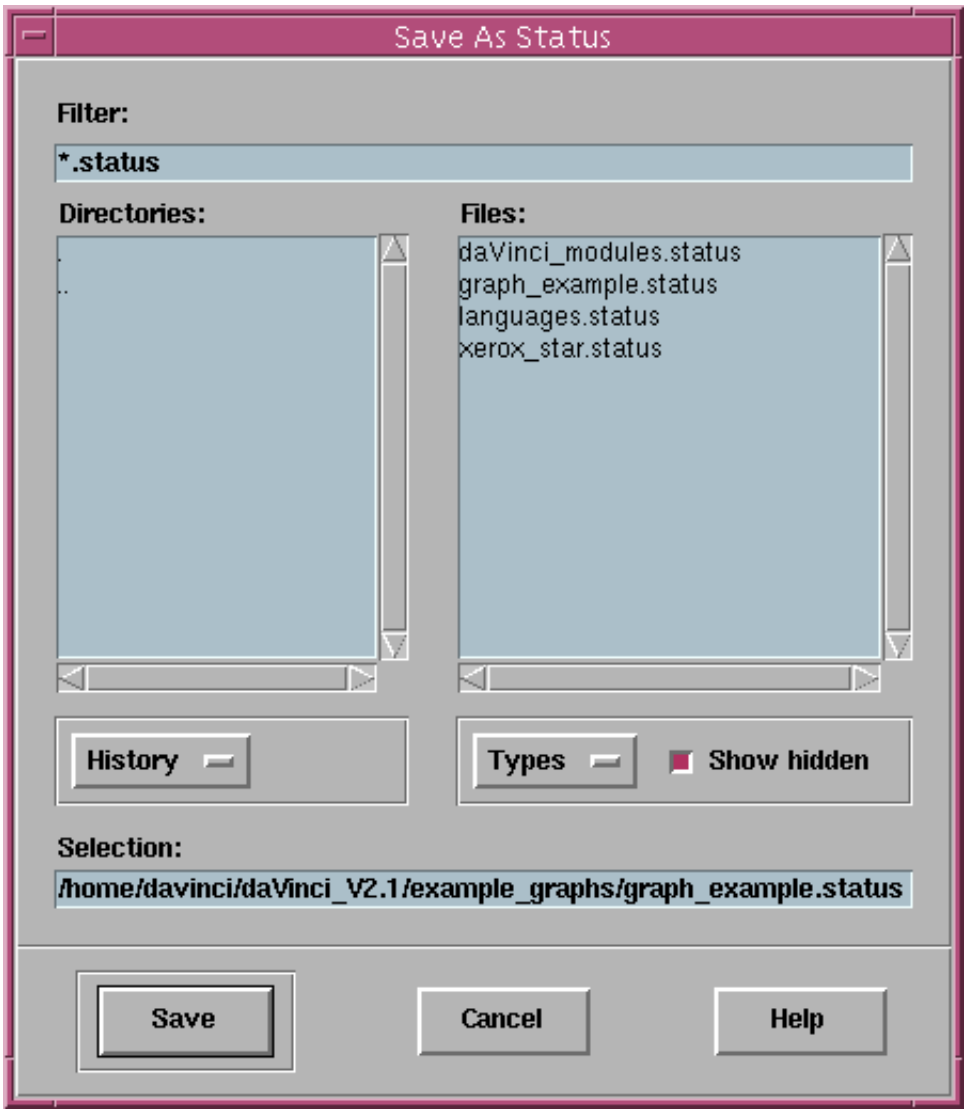
- **Open**
  Push button to load the choosen file specified in the **Selection** field. The file must exist to proceed with the load operation, otherwise the browser beeps.

- **Cancel**
  Push button to close the file browser without performing any action.

- **Help**
  Push button to start the online help system with this page.

daVinci V2.1 Online Documentation - Page update: June 15, 1998

File Browser to save Files



This file browser can be used to locate a graph or graph status file for saving. Graph files contain a term representation with the structure of a graph, but without layout informations (e.g. node coordinates). They should have the suffix *.daVinci*. On the other side, a status file contains the graph, the layout informations with the position of graph nodes and edges and most of the current user interface settings. Status files can be used to restore a previous *daVinci* session. They should always have the suffix *.status*. Before saving a graph, you need to specify the format using the **Type** option button. The default format is 'status file'.

The browser has two views, one for the **Directories** and one for the **Files** in the current directory. The current directory is the one shown in the **Selection** field. To navigate through the directories, double-click on the elements in the **Directories** view. By clicking on '..', you move to the parent directory. To choose a file, either select it in the **Files** view and press the **Save** push button or simply double-click it. To save a new file, you need to enter the filename in the **Selection** field. The browser beeps when you try to access a file or directory for which you do not have permissions. Before overwriting an existing file, the browser asks the user to confirm.

The file browser window can also be used mouseless with keyboard operations.

Dialog Items of the File Browser

- **Filter**
  Shows the filter expression currently applied to the **Files** view. The filter can either be selected with the **Types** option button or by manually typing an arbitrary shell expression here.

- **Directories**
  A list to show the directories inside the current directory (which is displayed in the **Selection** field). Double-click on the list elements to navigate through the directories. '..' moves to the parent directory.

- **Files**
  A list to show the files inside the current directory (which is displayed in the **Selection** field). Select one of the list elements and press the **Save** push button to save a *daVinci* graph- or status to a particular file. You can accelerate this by double-clicking a file. To save a new file, you need to enter the filename in the **Selection** field.

- **History**
  Used to go back to a previously visited directory or to the home directory of the user.

- **Types**
  Used to specify the saving format, either graph or status. This also reduces the number of elements in the **Files** view by applying a filter expression.

- **Show hidden**
  Select this check button to see hidden files and directories.

- **Selection**
  This text entry field shows the current selection which is modified when you browse through the directories. The browser usually suggests an appropriate initial selection. You can also edit the selection field by hand: If the entered selection is a directory, the browser will move to that directory, otherwise a save action is performed with the specified file.

- **Save**
  Push button to save the choosen file specified in the **Selection** field. If the file already exists, the browser will ask the user to confirm overwriting.

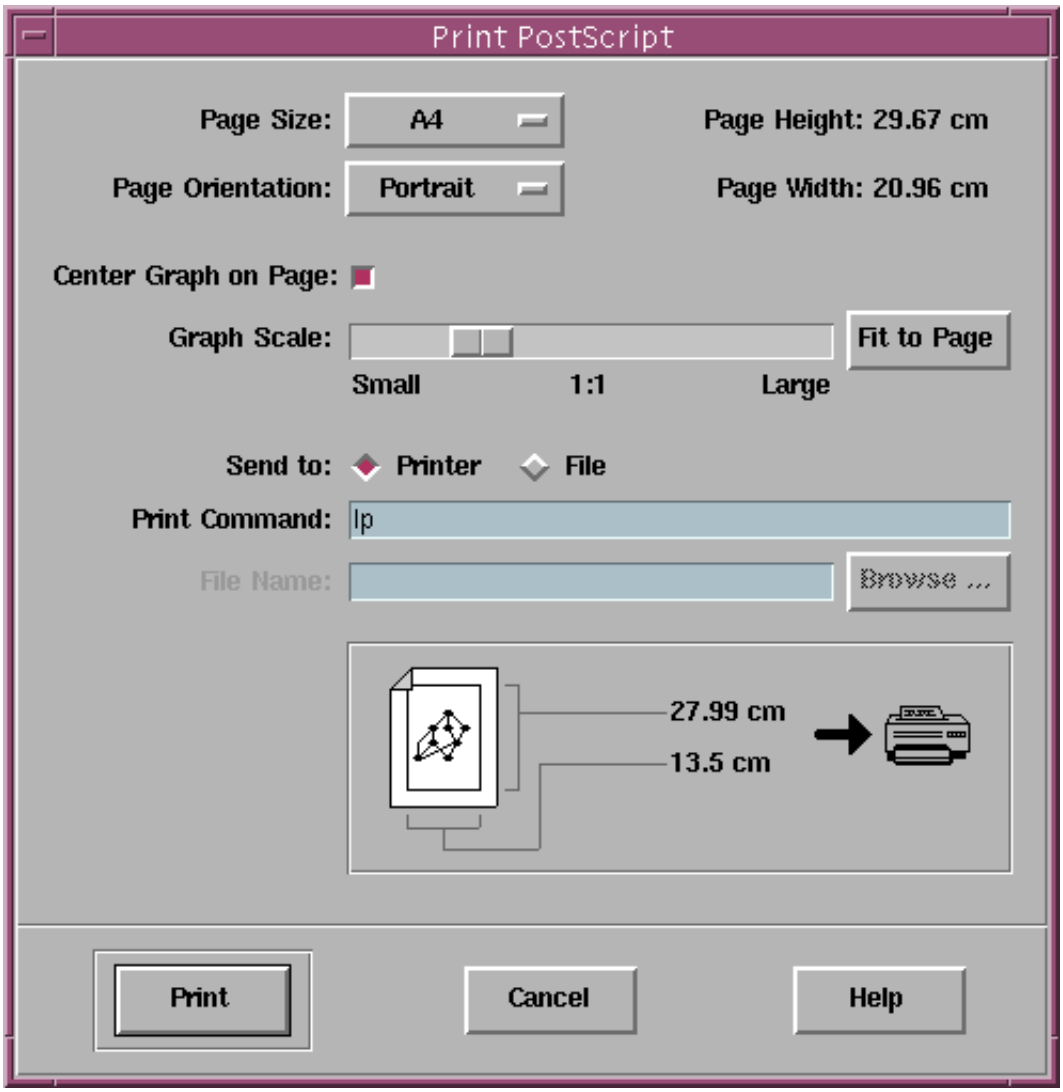
- **Cancel**
  Push button to close the file browser without performing any action.

- **Help**
  Push button to start the online help system with this page.

---

daVinci V2.1 Online Documentation - Page update: June 15, 1998

Print Dialog Window



This dialog window can be used to set options and target before printing the current graph in PostScript format. The graph can either be sent to a PostScript printer or saved in a file for import in another application. Informations about the *daVinci* PostScript interface are available in the reference chapter.

Initially, the options in this dialog are set such that the graph is sent to a printer, fits on the page according to the specified size and is centered on the page. Before printing, the user is able to modify these settings. Beside using the windows dialog items, this can also be done with direct manipulation of the "paper picture" in the lower part of the window:

- Click on the symbolized graph (on the paper) to switch the **Center Graph on Page** check button.
- Click on the symbolized paper (i.e. on the border around the graph) to switch the **Page Orientation** option button.
- Click on the symbolized printer resp. disk icon to switch the **Send to** radio buttons.

The measure unit (either cm or inch) and the print command can be set using dialog Options/General Settings.... The print dialog window can also be used mouseless with keyboard operations.

Dialog Items of the Print Window

- **Page Size**
  Specifies the size of the page used for printing the graph, either *A3*, *A4*, *A5*, *Letter* or *Legal*. Look at your printer to choose the appropriate value. The resulting page size (in cm or inch) is displayed in the window. This size is used to center the graph correctly on a page and has no effect when the **Center Graph on Page** check button is not selected.

- **Page Orientation**
  Specifies the orientation of the graph on the page, either *Landscape* or *Portrait*. The resulting page width and height (in cm or inch) is displayed in the window. The initial orientation is set automatically to get a maximal scale (e.g. *Landscape*, if the graph height is smaller than the width).

- **Center Graph on Page**
  Select this check button to center the graph on the page. You have to choose the right **Page Size** of your printer to get proper results. Usually, centering is desired when a graph is printed on paper, but not when a PostScript file is imported in another application. So, if the user switches the **Send to** radio buttons, then this check button is automatically set (centered for printers, not centered for files). Of course, this automatic setting can be manipulated afterwards.

- **Graph Scale**
  This slider sets the scale in the PostScript representation of the graph. The **Fit to Page** button calculates the maximal scale where the graph is completely visible on the page (but not larger than a scale of **1:1**). This is also the scale used as initial value. After manipulating the scale, the actual width and height of the PostScript graph will be displayed (in cm or inch) in the picture in the lower part of the window. Width and height are drawn in red when the graph exceeds the current dimensions of the page. A special scale is **1:1** where one inch on the screen is one inch in the generated PostScript file (for 72dpi screens). To get this scale, simply click on the **1:1** label.

  Note: The *daVinci* PostScript interface has a minimal scale with a magnification rate of 0.1x which is the **Small** value of the scale slider. No smaller scales are supported, so very large graphs may not fit on the page of the selected size. The **Fit to Page** button does not reduce the scale if the scale slider is already at the minimal value. Choose a larger page size or do not use the **Center Graph on Page** option in this case.

- **Send to**
  Used to select the target of the PostScript graph, either a printer specified by the **Print Command** field or a file specified by the **File Name** field. Switching these radio buttons also manipulates the **Center Graph on Page** check button for more convenience (see above).

- **Print Command**
  This specifies the UNIX command used for printing the graph on a PostScript printer after the **Print** push button is pressed. The required syntax is the same used in a UNIX shell for printing files, but without the filename. Options to the print command may also be typed here.

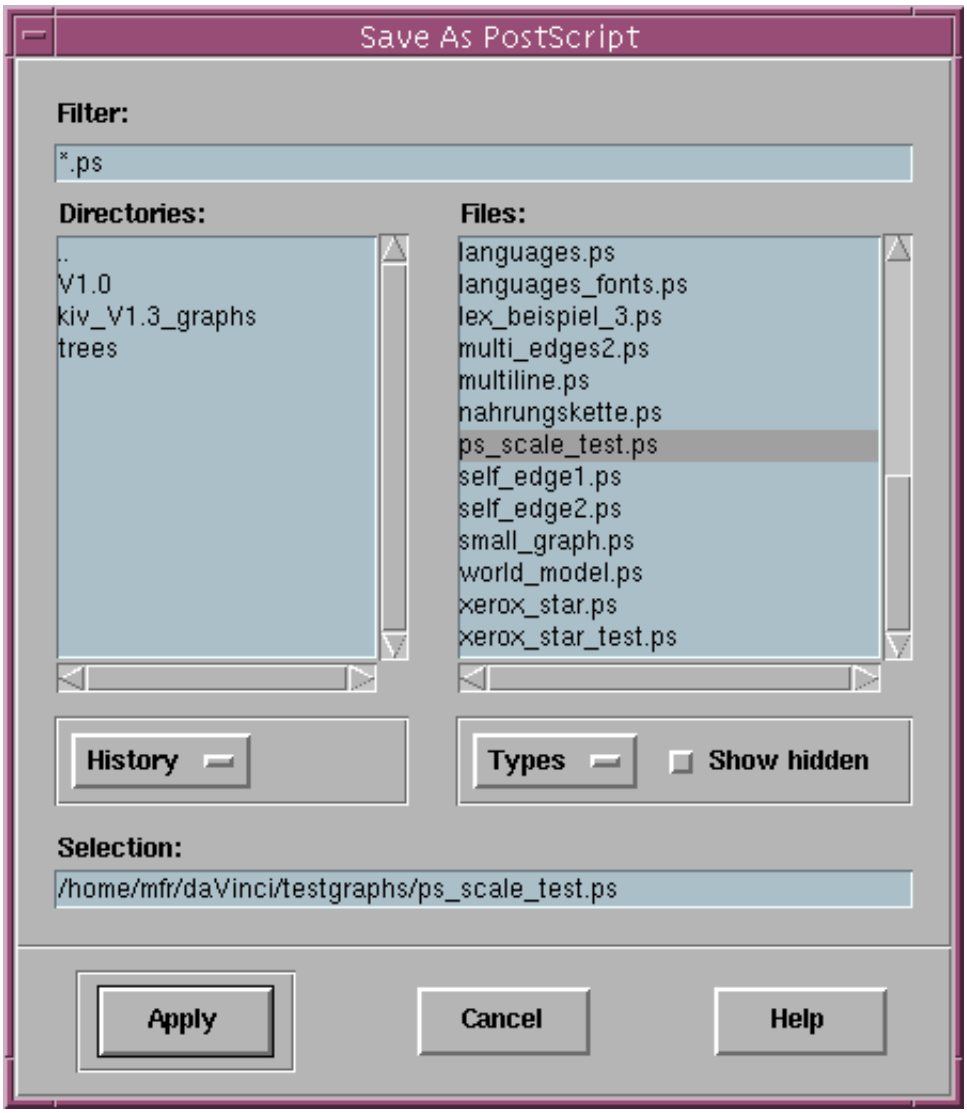
- **File Name**
  This specifies the filename where the PostScript representation of a graph is stored after the **Print** push button was pressed. Either type the filename directly into this field or use the file browser for more convenience by pressing the **Browse...** button.

- **Print** (or **Save**)
  Push button to print the graph in PostScript format either on a printer or save it into a file, according to the **Send to** radio buttons.

- **Cancel**
  Push button to close the print dialog without performing any action.

- **Help**
  Push button to start the online help system with this page.

File Browser to specify PostScript Files



This file browser can be used to locate a file for saving a graph in PostScript format. The browser does not save the file directly, it just places the selected filename into the **File Name** field of the print dialog window. There you need to press the **Save** button to actually save the file.

The browser has two views, one for the **Directories** and one for the **Files** in the current directory. The current directory is the one shown in the **Selection** field. To navigate through the directories, double-click on the elements in the **Directories** view. By clicking on '..', you move to the parent directory. To choose a file, either select it in the **Files** view and press the **Apply** push button or simply double-click it. To save a new file, you need to enter the filename in the **Selection** field.

The file browser window can also be used mouseless with keyboard operations.

Dialog Items of the File Browser

- **Filter**
  Shows the filter expression currently applied to the **Files** view. The filter can be typed manually using an arbitrary shell expression.

- **Directories**
  A list to show the directories inside the current directory (which is displayed in the **Selection** field). Double-click on the list elements to navigate through the directories. '..' moves to the parent directory.

- **Files**
  A list to show the files inside the current directory (which is displayed in the **Selection** field). This file browser only shows PostScript files. Select one of the list elements and press the **Apply** push button to use this filename into the **File Name** field of the print dialog window. You can accelerate this by double-clicking a file. To save a new file, you need to enter the filename in the **Selection** field.

- **History**
  Used to go back to a previously visited directory or to the home directory of the user.

- **Types**
  Not used in this file browser, because only PostScript files can be displayed here.

- **Show hidden**
  Select this check button to see hidden files and directories.

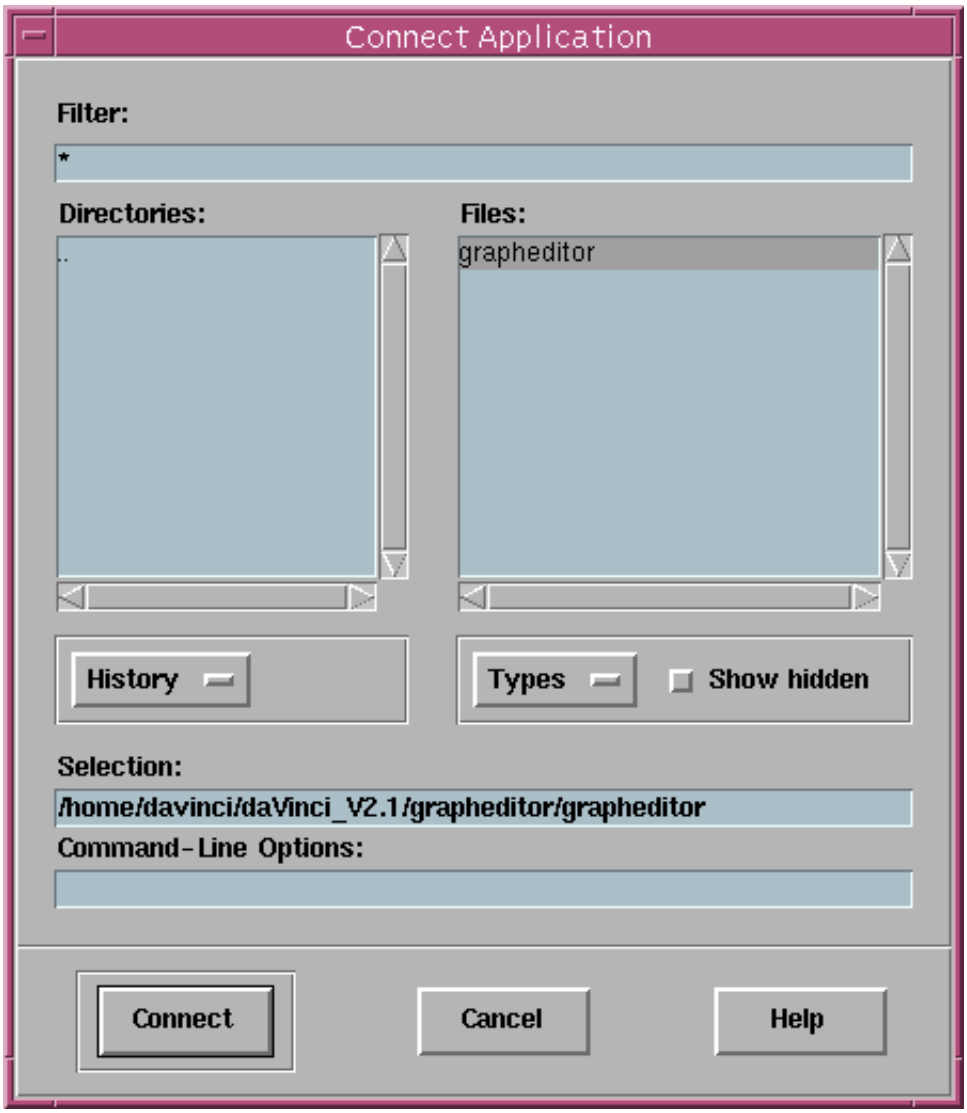
- **Selection**
  This text entry field shows the current selection which is modified when you browse through the directories. The browser usually suggests an appropriate initial selection. You can also edit the selection field by hand: If the entered selection is a directory, the browser will move to that directory, otherwise an apply action is performed with the specified file.

- **Apply**
  Push button to place the choosen file, specified in the **Selection** field, into the **File Name** field of the print dialog window. If the file already exists, the browser will ask the user to confirm overwriting.

- **Cancel**
  Push button to close the file browser without performing any action.

- **Help**
  Push button to start the online help system with this page.

File Browser to connect Applications



This file browser can be used to locate the binary of an application program which shall be connected to the API. Such an application can use *daVinci* as a user interface for graphs. One application is the graph editor which is part of the *daVinci* distribution.

The browser has two views, one for the **Directories** and one for the **Files** in the current directory. The current directory is the one shown in the **Selection** field. To navigate through the directories, double-click on the elements in the **Directories** view. By clicking on '..', you move to the parent directory. To choose a binary, either select it in the **Files** view and press the **Connect** push button or simply double-click it. The browser beeps when you try to access a file or directory for which you do

not have permissions, or when the specified file does not exist.

The file browser window can also be used mouseless with keyboard operations.

Dialog Items of the File Browser

- **Filter**
  Shows the filter expression currently applied to the **Files** view. The filter can be entered manually by typing an arbitrary shell expression.

- **Directories**
  A list to show the directories inside the current directory (which is displayed in the **Selection** field). Double-click on the list elements to navigate through the directories. '..' moves to the parent directory.

- **Files**
  A list to show the files inside the current directory (which is displayed in the **Selection** field). This file browser only shows executable files. Select one of the list elements and press the **Connect** push button to connect a particular binary to the *daVinci* API. You can accelerate this by double-clicking a file.

- **History**
  Used to go back to a previously visited directory or to the home directory of the user.

- **Types**
  Not used in this file browser, because only executable files can be displayed here.

- **Show hidden**
  Select this check button to see hidden files and directories.

- **Selection**
  This text entry field shows the current selection which is modified when you browse through the directories. You can also edit the selection field by hand: If the entered selection is a directory, the browser will move to that directory, otherwise a connect action is performed with the specified file.

- **Command-Line Options**
  This text entry field can be used to pass additional command-line options to the connected binary.

- **Connect**
  Push button to connect the choosen binary file specified in the **Selection** field. The file must exist to proceed with the connect operation, otherwise the browser beeps.

- **Cancel**
  Push button to close the file browser without performing any action.

- **Help**
  Push button to start the online help system with this page.

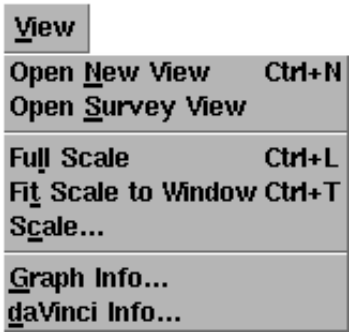
Edit Menu

The Edit menu contains operations for performing actions on the data. *daVinci* itself does not use this menu for its own operations. It is the primary philosophy of *daVinci* that the structure of a loaded graph cannot be modified by *daVinci* itself, so basic edit operations does not make sense.

Instead, this menu is the place where *daVinci* application programs, connected to the API, are able to attach their own menus. So entries in this menu are specific to a connected application and cannot be described here. Refer to the manual of the particular application for details.

One important *daVinci* application is the graph editor which is part of the *daVinci* distribution. The editor takes advantage of the API's menu feature and attaches its operation to the edit menu. Refer to the graph editor documentation for details.

View Menu



The View menu contains operations for changing the user's view to the data: Opening additional views to a graph, setting the scale of a graph visualization, or getting information.

Note: Menus can also be used mouseless with keyboard commands.

Multi-View Management

- **Open New View**
  This operation opens an additional view (base window) to the graph. Read more about multi-view and multi-graph in the concept document. All views of a graph are coupled with eachother, so interactions in one view affect the graph visualization in all the other views of the same graph, too. Loading a graph in one view disconnects this base window from the other views, so the new graph is only displayed in the view where the load operation was triggered. This way, the user can load many different graphs at a time (multi-graph). The maximal number of open base windows in *daVinci* is 64.

- **Open Survey View**
  This operation also opens an additional view to the graph, but with some special properties. Read more about multi-view in the concept document. The base window, where this operation is executed, is called *detail view*. A *survey view*, which will appear after using this operation, visualizes the graph in a reduced scale to see the whole structure at all times. The scale is reduced automatically when the window size is manipulated. A survey view shows the currently visible area of the corresponding detail view with a blue rectangle. This rectangle is only visible in a survey view when the scale is small enough to show the complete graph. So, if the blue rectangle has become invisible in a survey view, then use menu *View/Fit Scale to Window* (see below) to get the rectangle back. If the scale of the survey view is 100%, because the graph is already completely visible in the detail view, you will not see the blue rectangle as well. Another useful feature of the survey view is automatic scrolling. When the user selects a node in the survey view, the same node is focused in the corresponding detail view by using animation. This behaviour can be switched off in dialog Options/General Settings... by deselecting checkbox **On selection in Survey View**. The maximal number of open base windows in *daVinci* is 64.

Scaling

- **Full Scale**
  Used to set the graph visualization scale to 100%.

- **Fit Scale to Window**
  Used to set the graph visualization scale to the maximal rate where the graph is completely visible in the base window.

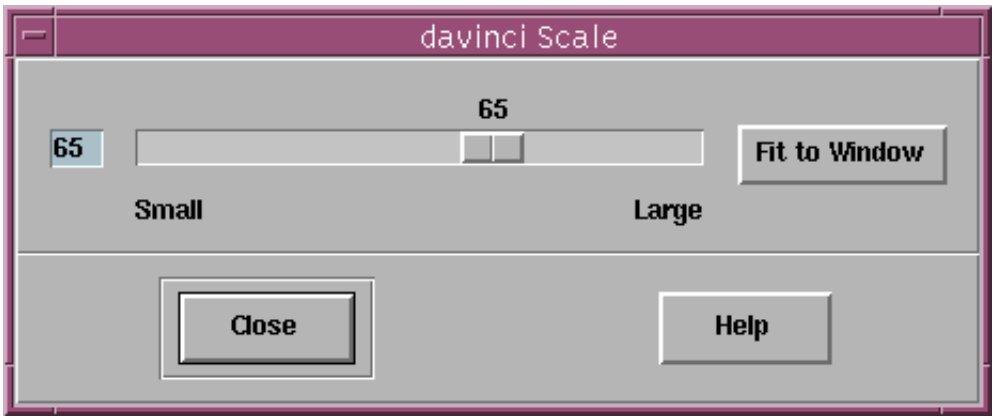
- **Scale...**
  Used to set the graph visualization scale to any rate between 1% and 100%. Read more about scaling in the concept document. After selecting this menu entry, a scale dialog window will appear to set the scale. In a reduced scale, all interactive operations are still available. The scale of a graph can only be reduced with this operation, but not magnified. To magnify a graph visualization, choose a bigger font in dialog Options/Layout Dimensions....

Getting Informations

- **Graph Info...**
  This operation displays the graph info dialog to get statistic informations about the currently visualized graph.

- **daVinci Info...**
  This operation shows a dialog window with daVinci product informations.

Scale Dialog Window



This dialog window is used to get an overview of a graph by setting the scale to any rate between 1% and 100%. In a reduced scale smaller than 100%, nodes are drawn more abstract (e.g. text or icons are not visible), but all interactive operations are still available. The scale of a graph can only be reduced with this operation, but not magnified. To magnify a graph visualization, choose a bigger font in dialog Options/Layout Dimensions.... In multi-view mode, the scale of each view can be set independently from the other views to the same graph. Background information about scaling is available in the concept document.

To set the scale, move the slider to the left or to the right. The scale of the graph visualization will change as soon as the mouse button is released. The current scale (in %) is shown above the slider. You can also enter the percent value directly by using the text entry field left to the slider. The **Fit to Window** button calculates the maximal scale for the current base window size to see the graph completely. You can also click on the **Small** and **Large** label to set the slider to the minimal (1%) and maximal value (100%).

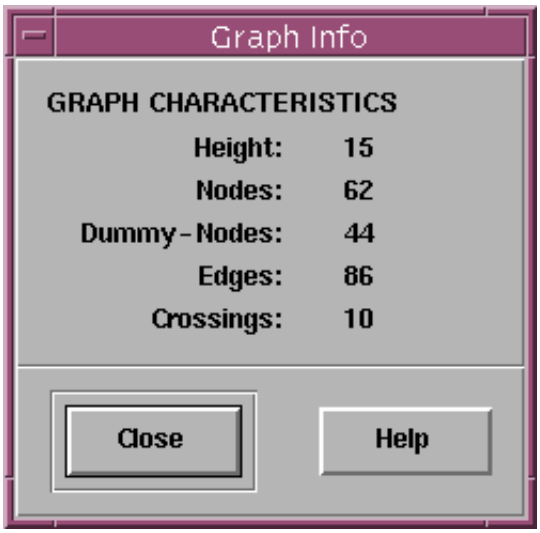
The scale dialog window can also be used mouseless with keyboard operations.

Dialog Items of the Scale Window

- **Fit to Window**
  Set the maximal scale for the current base window size to see the graph completely.
- **Close**
  Push button to close the scale dialog without performing any action.
- **Help**
  Push button to start the online help system with this page.

daVinci V2.1 Online Documentation - Page update: June 15, 1998

Graph Info Dialog Window



This dialog window is used to show some statistic informations about the currently visualized graph.

- **Height**
  Number of horizontal node levels in the graph (or vertical levels for other layout orientations).
- **Nodes**
  Number of nodes in the graph.
- **Dummy-Nodes**
  Number of invisible dummy nodes in the graph. Dummy nodes are generated automatically in a hierarchical layout to fill each passed level for long span edges. Read more about graph layout in the concept document.
- **Edges**
  Number of edges between nodes in the graph. Automatically generated edges from and to dummy nodes (i.e. the fragments of long span edges) are not counted.
- **Crossings**
  Number of crossings between edges in the graph. You can reduce the crossings by starting the graph layout algorithm with menu Layout/Improve All.

Dialog Items of the Graph Info Dialog

- **Close**
  Push button to close the graph info dialog without performing any action.

- **Help**
  Push button to start the online help system with this page.

---

daVinci V2.1 Online Documentation - Page update: June 15, 1998

The daVinci Info Dialog Window

This dialog window is used to show product informations about *daVinci*.

Dialog Items of the daVinci Info Dialog

- **Close**
  Push button to close the daVinci info dialog without performing any action.
- **Get Introduction**
  Push button to start the online help system with the introduction page.

Navigation Menu

The Navigation menu contains operations for performing actions on the node selection, such as selecting relatives and neighbours of a node, and to search for nodes with a particular text.

Note: Menus can also be used mouseless with keyboard commands.

Node Selection Operations

- **Select Parents**
  Changing the selection to highlight all parent nodes of the currently selected nodes.

- **Select Siblings**
  Changing the selection to highlight all sibling nodes of the currently selected nodes. The siblings of a node are all children of all parents of that node.

- **Select Children**
  Changing the selection to highlight all child nodes of the currently selected nodes.

- **Navigator...**
  Used to move the focus in a graph. Starting at a selected node, both neighbour- and relative nodes can be reached in all four directions. After selecting this menu entry, a navigator dialog window will appear. **Note:** Navigation is also available by using the cursor keys on the keyboard.

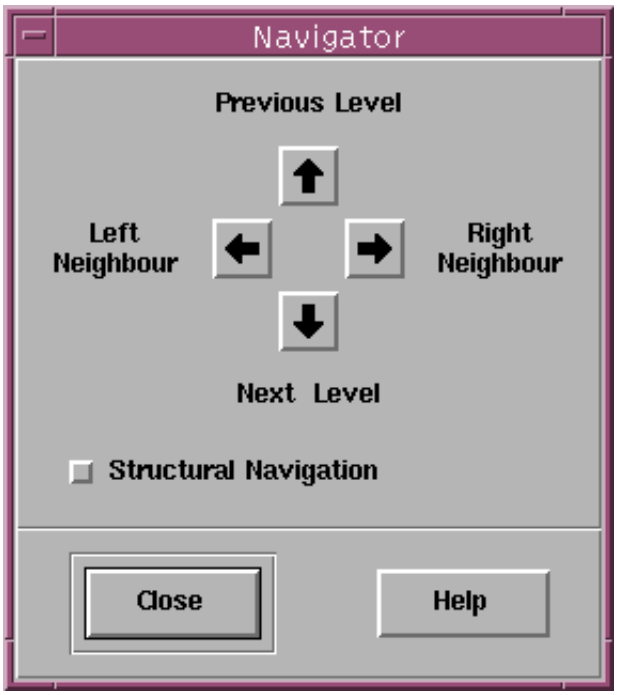
Search Operations

- **Find...**
  Searching for a node with a particular text. After selecting this menu entry, a find dialog window will appear. If exactly one node is selected, the search starts behind this node, otherwise it begins at the topmost and leftmost node in the graph (for top-down layouts).

- **Find Again**
  Continuing a search operation to find the next node with a particular text which has been specified before in the find dialog, without opening this window again. Therefore, the operation is only available when a text is entered in the find dialog. If exactly one node is selected, the

search continues behind this node, otherwise the search begins at the topmost and leftmost node in the graph (for top-down layouts). This operation beeps when there is no node matching the search criteria.

Navigator Dialog Window



This dialog window is used to move the focus in a graph. By starting at a selected node, both the neighbour- and relative nodes can be reached in all four directions. To work with the navigator, exactly one node needs to be selected in the graph. If no node or more than one node is selected, then the navigator does not work and beeps when the arrow buttons are pressed. By pressing one of the four arrow buttons in the dialog window, the focus (i.e. node selection) moves from the selected node to the left/right neighbour node or the nearest node at the next/previous level, according to the choosen direction. The navigator beeps when there is no such target node, e.g. nodes at the topmost level do not have any parents. Movement from the old to the new node is shown with animation by scrolling to the new node. This animation can be switched off or controlled in speed with menu Options/General Settings....

By switching the orientation with menu Layout/Orientation, the functions behind the arrow buttons of the navigator are also rotated to get proper results for navigation. This is emphasized by changing the text labels of the navigator window which are next to the four arrow buttons. So in *Left to Right* orientation, the *Previous Level* label for *Top Down* orientations becomes *Upper Neighbour*.

The navigator supports an alternative mode which can be set by selecting the **Structural Navigation** check button. This mode considers the structure of the graph rather than the geometrical relationships in the current layout. So in structural mode, pressing the left arrow button does not move to the left neighbour node, but to the nearest sibling at the left side of the currently selected node. In the same way, pressing the up and down arrow buttons moves to the nearest parent and child. "Nearest" means:

The relative node with the smallest distance on the x-axis, coordinates on the y-axis are not considered. These examples are for *Top-Down* layout orientations. In other orientations, the navigation buttons are rotated as well, so in a graph with *Left-Right* layout where the parents are always at the left side of their children, pressing the left arrow button in structural mode will move to the nearest parent node (instead of the nearest left sibling node for *Top-Down* orientations). The text labels next to the arrow button always characterize the functions behind the buttons for all orientations, so you do not have to be worried about this complicated issue.

The navigator dialog window can also be used mouseless with keyboard operations. Further, navigation is available without using this dialog window by simply pressing the cursor keys in a base window, refer to keyboard usage for details.

Dialog Items of the Navigator Window

Note: For layout orientations other than *Top Down*, the functions behind the arrow buttons are also rotated to get proper results for navigation! So the up arrow always moves upwards.

- **Up Arrow**
  Starting at the currently selected node, move to the nearest node in upward direction (or to the nearest related node in structural navigation mode).

- **Down Arrow**
  Starting at the currently selected node, move to the nearest node in downward direction (or to the nearest related node in structural navigation mode).

- **Left Arrow**
  Starting at the currently selected node, move to the nearest node in left direction (or to the nearest related node in structural navigation mode).

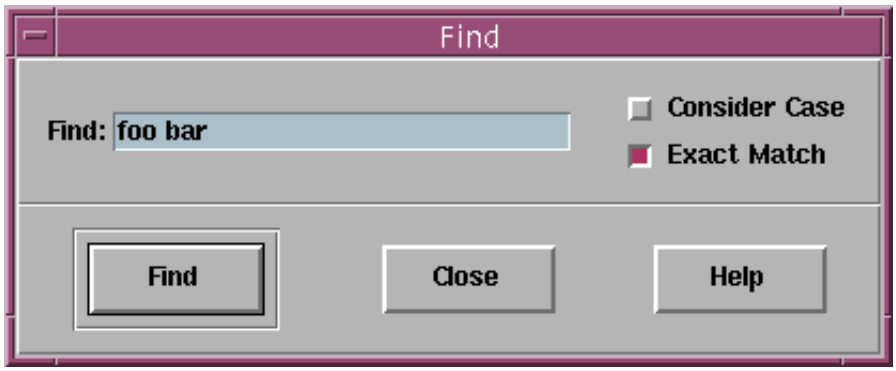
- **Right Arrow**
  Starting at the currently selected node, move to the nearest node in right direction (or to the nearest related node in structural navigation mode).

- **Structural Navigation**
  This check button switches between structural navigation mode, where the structure of a graph is considered, and the default mode, where only the geometrical relationships in a graph are considered for navigation (read above for details). Using this check button takes an influence on the operations behind the arrow buttons in this window.

- **Close**
  Push button to close the navigator dialog without performing any action.

- **Help**
  Push button to start the online help system with this page.

Find Dialog Window



This dialog window is used to find nodes in a graph that match a specified string. The string has to be entered in the **Find** field of the dialog. Two search options are available: **Consider Case** to distinguish between upper and lower case letters and **Exact Match** to ignore nodes where the entered string appears as a substring. Both options are not applied by default, so you have to select the check buttons to get the described behaviour.

After typing the string, the search can be started with the **Find** push button. This will not close the window. If exactly one graph node is selected, then the search starts behind this node, otherwise at the topmost and leftmost node of the graph. *daVinci* looks at each node of a level from the left to the right and continues with the next level by going from top to down. As soon as the text of a node (given by the node's attribute OBJECT) matches the specified string, the search stops and the found node is selected. Movement from the old to the new node is shown with animation which can be switched off or controlled in speed with menu Options/General Settings.... When no node of the graph matches the entered string, the search will be stopped after reaching the start node again and the user will hear a beep.

The navigator dialog window can also be used mouseless with keyboard operations.

Dialog Items of the Find Window

- **Find** (text entry field)
  In this field, you have to enter the text of the node you are looking for. Wildcards are not supported, so characters like * or ? do not have a special meaning.

- **Consider Case**
  Select this check button when upper and lower case letters should be distinguished from eachother for the search.

- **Exact Match**
  Select this check button when you do not want to find nodes with a text containing the searched string as a substring.

- **Find**
  Push button to start the search by looking for a node that matches the string currently entered in the **Find** text entry field.

- **Close**
  Push button to close the find dialog without performing any action.

- **Help**
  Push button to start the online help system with this page.

Abstraction Menu

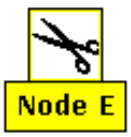
The Abstraction menu contains operations for hiding particular parts of a graph visualization. Two kinds of abstractions are currently available: Subgraph abstractions and edge abstractions.

Note: Menus can also be used mouseless with keyboard commands.

Subgraph Abstraction

- **Hide Subgraph**
  This operation is used to fade out the subgraphs(s) of the currently selected node(s). The advantage of using this abstraction is to get a more clear graph layout where uninteresting or confusing subgraphs do not appear. The operation is only available when nodes are selected where the subgraph is not already hidden. After using this function, all children (and their children, and so on) of the selected node(s) will be removed from the graph visualization. This recursive hiding process stops as soon as a child is reached that has at least one parent which is not hidden at the same time. So, this operation will hide only the children that can be exclusively reached by starting at the selected node(s).

  Nodes with hidden subgraphs get an alternative visualization. This alternative visualization can be specified with an attribute in the term-representation. In the default case an icon with a pair of scissors is used to emphasize the applied abstraction:



  To undo a subgraph abstraction, select menu **Show Subgraph** or **Restore all Subgraphs**, see below. This will display the hidden subgraph(s) again.

  After using this abstration, the corresponding subgraph will be simply removed from the visualization without taking advance of the space that becomes free. If you want to utilize the free space in the layout after using subgraph abstractions, choose menu Layout/Improve Spacing.

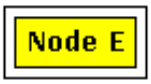
- **Show Subgraph**
  Undo previously applied subgraph abstractions for the currently selected node(s). The operation is only available when nodes are selected where the subgraph is actually hidden. The subgraph is inserted by using the incremental graph layout. If the layout has changed by using menus Layout/Improve All, Layout/Improve Spacing or after fine tuning, the graph nodes may not be at the same position as before, because other nodes may now occupy the space needed to draw the subgraph again.

- **Restore all Subgraphs**
  This operation undos previously applied subgraph abstractions for all nodes in the graph in one step. See menu operation **Show Subgraph** above for details about undoing subgraph abstractions.

Edge Abstraction

- **Hide Edges**
  This operation is used to fade out the incoming and outgoing edges (from the parents and to the children) of the currently selected node(s). The advantage of using this abstraction is to get a more clearer graph layout where uninteresting or confusing edges do not appear. The operation is only available when nodes are selected where the edges are not already hidden. After using this function, all edges of the selected node(s) will be removed from the graph visualization.

  Nodes with hidden edges get an alternative visualization by drawing a rectangular frame around the node:

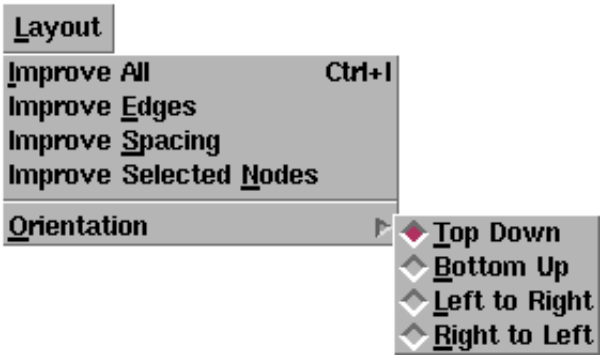
  To undo an edge abstraction, select menu **Show Edges** or **Restore all Edges**, see below. This will display the hidden edges again.

  After using this abstration, the corresponding edges will be simply removed from the visualization. They can be redrawn at any time by using the two menu operations below without calculating a new graph layout.

- **Show Edges**
  Undo previously applied edge abstractions for the currently selected node(s). The operation is only available when nodes are selected where the edges are actually hidden. Unlike subgraph abstractions, showing the edges will never start the layout algorithm. So after using this operation, the graph nodes will always be at the same position as before.

- **Restore all Edges**
  This operation undos previously applied edge abstractions for all nodes in the graph in one step. See menu operation **Show Edges** above for details about undoing edge abstractions.

Layout Menu



The Layout menu contains operations for controlling the layout of a graph visualization which is shown in a *daVinci* base window. There are operations to optimize a layout and to select the layout orientation.

Note: Menus can also be used mouseless with keyboard commands.

Layout Optimization

- **Improve All**
  This operation starts an algorithm to optimize the layout of the whole graph. The most important task of the layout algorithm is to reduce the number of crossings between edges. Reducing edge crossings is a technique to clarify a graph layout. The current number of crossings is displayed in the View/Graph Info... dialog. Crossing minimization is done by reordering the nodes at each level using a heuristic. Details about the layout algorithm can be found in the concept document. The layout algorithm will never be used after loading a graph, so the user has to do this manually with this menu operation.

- **Improve Edges**
  Same as **Improve All** (see above), but the order of the nodes is fixed, i.e. only the layout of the dummy nodes is calculated. For the user this function has the visible effect of a better edge routing, without changing the order of the regular nodes.

- **Improve Spacing**
  This operation moves all nodes closer together in the graph visualization to take advantage of unused space. The order of nodes is not affected. In *daVinci* V2.0.2 or higher, empty levels at the top or bottom of the graph (which may be a result of vertical fine-tuning) are also eliminated. Usually, one will use this operation after hiding subgraphs with menu operation Abstraction/Hide Subgraph to get a more compact layout.

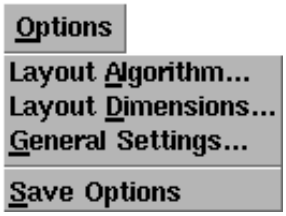
- **Improve Selected Nodes**
  Same as **Improve All** (see above), but considers only the nodes that are currently selected in the base window and all dummy nodes like in **Improve Edges** (see above). The advantage of a local layout optimization is that the resulting visualization may be better for the selected nodes, because the surrounding nodes which are not visible at this time must not be taken into account. The disadvantage is that the layout of the other nodes may be getting worse after using this operation. So use this operation when you are not interested in a good overall layout, but want to get the best quality for the selected nodes.

Controlling the Layout Orientation

- **Orientation**
  The operations in this cascade menu are used to set the orientation of a graph layout. By default, a hierarchical visualization of a graph is draw in top-down direction such that the first level (with the root nodes) is at the upper side of the graph layout and all edges point downwards. Use any of these menus to draw the graph in one of the four orientations. Details about the graph layout can be found in the concept document.

  - **Top Down**
    Drawing the graph in top-down layout whereby the first level is at the upper side and all edges point downwards. This is the default direction.
  - **Bottom Up**
    Drawing the graph in bottom-up layout whereby the first level is at the lower side and all edges point upwards.
  - **Left to Right**
    Drawing the graph in left-to-right layout whereby the first level is at the left side and all edges point to the right.
  - **Right to Left**
    Drawing the graph in right-to-left layout whereby the first level is at the right side and all edges point to the left.

Options Menu



The Options menu contains operations for controlling the options of the *daVinci* system. There are operations to modify the options and to save them for permanent use.

Note: Menus can also be used mouseless with keyboard commands.

Changing the Options

- **Layout Algorithm...**
  Used to set the options for controlling the graph layout algorithm and the manual fine tuning. After selecting this menu entry, a layout algorithm dialog window will appear to specify the options.

- **Layout Dimensions...**
  Used to set the options for controlling the dimensions of a graph layout. After selecting this menu entry, a layout dimensions dialog window will appear to specify the options.

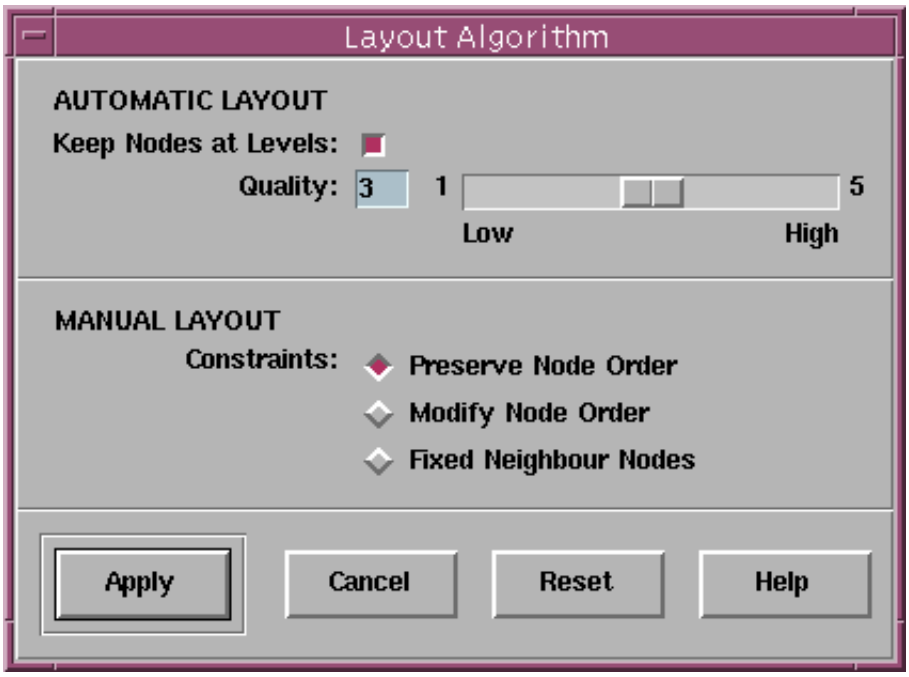
- **General Settings...**
  Used to set some general options of the *daVinci* system. After selecting this menu entry, a general settings dialog window will appear to specify the options.

Saving the Options

- **Save Options**
  This operation is used to save the current options, set by using the menu operations listed above, to a file for permanent use. *daVinci* will always load this file at start-up time to restore the preferences of the user. The options are saved in file *.daVinci* in the home directory of the user. **Note:** Do not edit this file by hand!

  In *daVinci* V2.0.3, the format of the option file has been modified. So options saved by V2.0.3 or higher are ignored by earlier V2.0.x releases, but V2.0.3 or higher can read the options saved by earlier V2.0.x releases using a compatibility mode.

Layout Algorithm Dialog Window



This dialog window can be used for controlling the graph layout algorithm (automatic layout) and interactive fine tuning operations (manual layout).

In *daVinci* V2.x, fine tuning is available to move a node vertically to another level by keeping the graph visualization hierarchically. Read more about graph layout and hierarchical visualizations in the concept document. By default, the layout algorithm keeps the nodes at their current levels when a new layout is calculated. This behaviour can be switched off by deselecting check button **Keep Nodes at Levels**. In this case, nodes are always assigned back to their initial level when the layout algorithm is started again with menu Layout/Improve All.

The **Quality** slider is used to adjust the accuracy of the layout algorithm, i.e. the number of iterations of the Sugiyama graph layout algorithm. The higher the slider value, the better the layout results which will give a reduced number of edge crossings. But higher settings need a lot of computing time, especially for very large graphs..

For the fine tuning (manual layout), some **Constraints** can be set to control horizontal movement of a node. The three constraints specify the behaviour when a node, moved with the mouse, will touch its left or right neighbour node at the same level. A neighbour node is touched when the minimal node distance is reached by the mouse pointer.

- In default mode **Preserve Node Order**, the touched node is moved as well to keep the order of nodes.

- In mode **Modify Node Order**, the moved node and the touched node exchange their position as soon as there is enough space to do so. This mode is useful to modify the order of nodes on a level, e.g. to remove edge crossings by hand.

- In the third mode **Fixed Neighbour Nodes**, fine tuning stops as soon as a neighbour node is touched which is useful to protect the position of neighbour nodes.

The layout algorithm dialog window can also be used mouseless with keyboard operations.

Dialog Items of the Layout Algorithm Window

- **Keep Nodes at Levels**
  Keeping the graph nodes at their current level when a new graph layout is calculated by starting the layout algorithm with menu Layout/Improve All. Nodes may be assigned to a new level by using vertical fine tuning. If you want to set the nodes back at their initial levels with the next run of the layout algorithm, then deselect this check button.

- **Quality**
  Setting the accuracy of the layout algorithm to some abstract value between 1 and 5. The higher the slider value, the better the layout results (reduced number of edge crossings). But higher settings need a lot of computing time, especially for very large graphs.

- **Preserve Node Order**
  Fine tuning constraint for horizontal node movement to keep the order of nodes at a level.

- **Modify Node Order**
  Fine tuning constraint for horizontal node movement to exchange the order of nodes at a level.

- **Fixed Neighbour Nodes**
  Fine tuning constraint for horizontal node movement to freeze the position of all other nodes at a level.

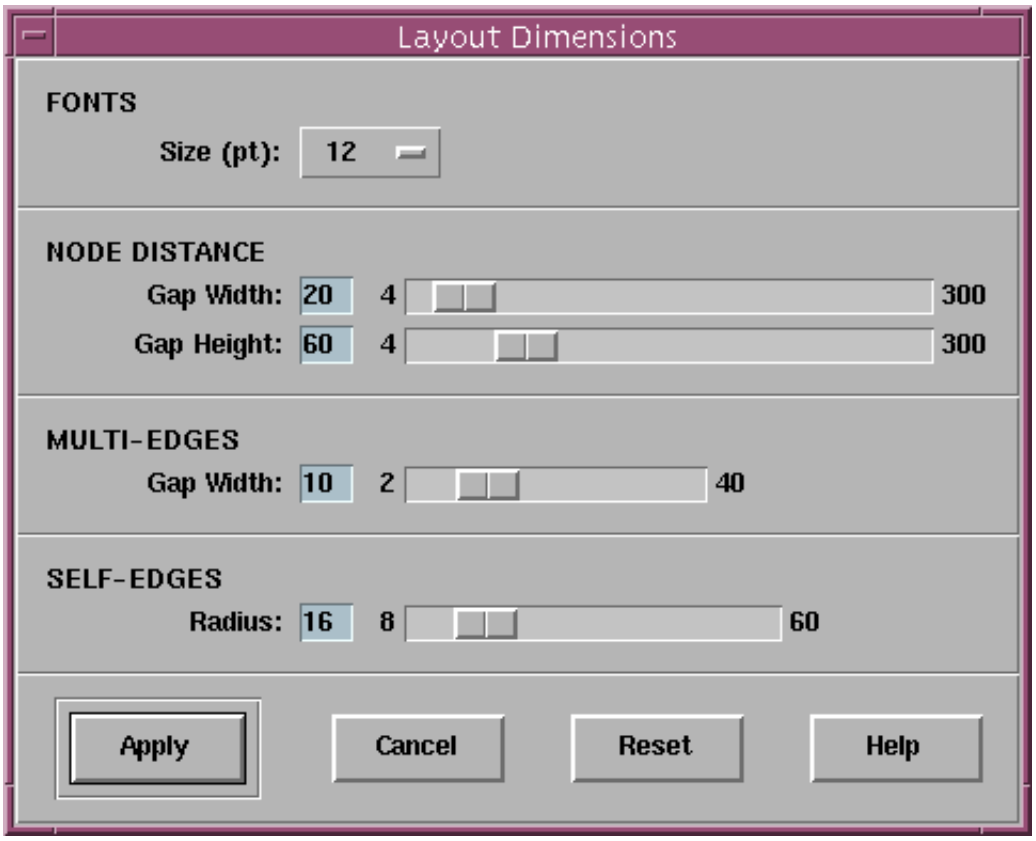
- **Apply**
  Push button to accept the current settings of this dialog window. Without pressing **Apply**, modifications to these options are not considered by *daVinci*.

- **Cancel**
  Push button to close the layout algorithm dialog without performing any action.

- **Reset**
  Push button to undo any modification to these options unless **Apply** is not pressed. So **Reset** goes back to the current settings.

- **Help**
  Push button to start the online help system with this page.

Layout Dimensions Dialog Window



This dialog window can be used for controlling the dimensions of a graph visualization. Modifying these options has an influence on the extension of a graph visualization, so if you do not want to reduce the scale of a graph with dialog View/Scale..., you can reduce the font size or the node distances here to get a smaller visualization.

After changing the dimensions, the graph layout has to be recalculated to deal with new node extensions, but the order of nodes is preserved. This way, interactive fine tuning of a graph layout is usually lost. So, be sure to set the final dimensions before starting with serious fine tuning. By selecting a smaller or larger font size, the other dimensions in this window are also adjusted to keep the layout proportional.

The layout dimension dialog window can also be used mouseless with keyboard operations.

Dialog Items of the Layout Dimensions Window

- **Size (pt)**
  Setting the font size in the visualization of a graph by choosing a value between 6 point and 34 point from the option button. Changing the font size will also adjust the other dimensions in this window to keep the layout proportional. The font family and style cannot be selected interactively in *daVinci*. This is determined by the node attributes in the term representation of a graph.

- **Gap Width**
  Setting the minimal horizontal distance between nodes at the levels (horizontal for top-down and bottom-up layouts, vertical otherwise).

- **Gap Height**
  Setting the minimal vertical distance between the levels (vertical for top-down and bottom-up layouts, horizontal otherwise).

- **Gap Width** (Multi-Edges)
  Setting the gap which is used for drawing multi-edges.

- **Radius**
  Setting the radius which is used for drawing self-edges.

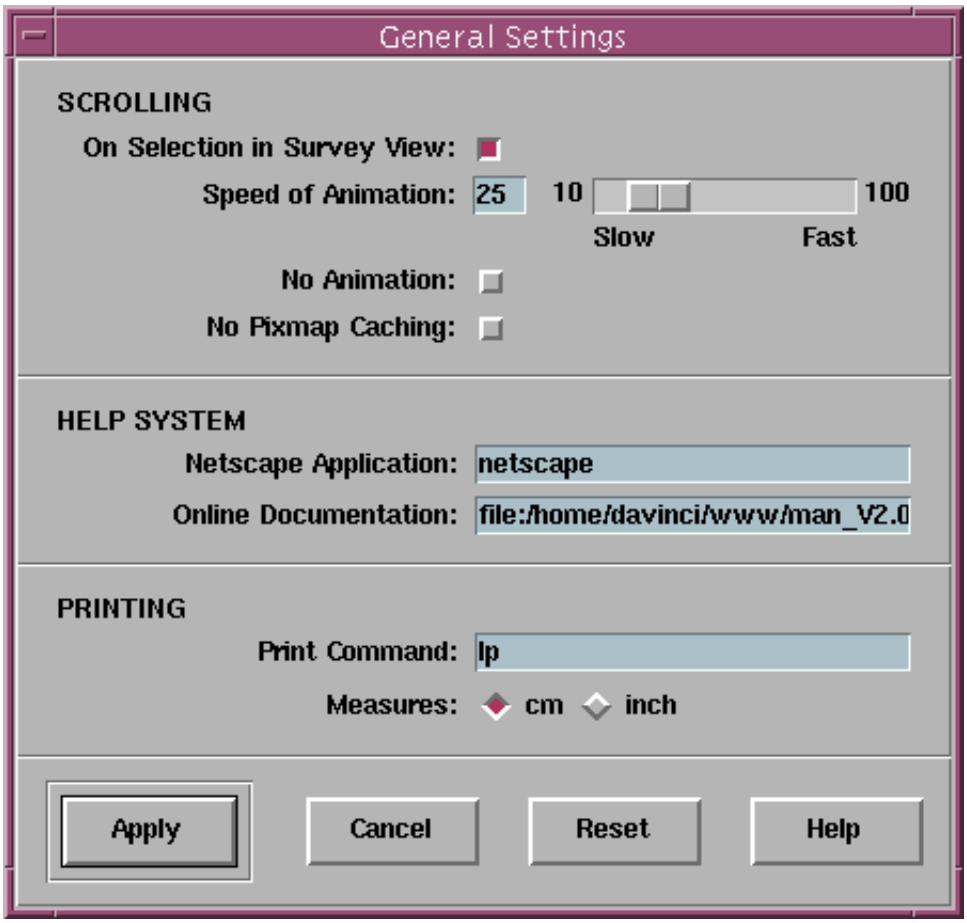
- **Apply**
  Push button to accept the current settings of this dialog window. Without pressing **Apply**, modifications to these options are not considered by *daVinci*.

- **Cancel**
  Push button to close the layout dimensions dialog without performing any action.

- **Reset**
  Push button to undo any modification to these options unless **Apply** is not pressed. So **Reset** goes back to the current settings.

- **Help**
  Push button to start the online help system with this page.

General Settings Dialog Window



This dialog window can be used for controlling some general settings of *daVinci*. The first set of dialog items deals with the scrolling management. For example, after opening a survey view from a detail view with menu View/Open Survey View, *daVinci* will automatically scroll to a particular node in the detail view if the same node is selected in the survey view. This scrolling is animated by moving the visible area of the base window towards the selected node (the navigator and find operation use animation, too). The animation's moving speed can be set with slider **Speed of Animation** or turned off with check button **No Animation**. Further, the behaviour that selection in a survey view will automatically scroll in the detail view can be switched off with check button **On Selection in Survey View**. Another check button **No Pixmap Caching** is used to turn off the cache for already drawn parts of a graph visualization (read about drawing technology in the concept document). This should be done when you get memory problems on computers with an insufficent amount of RAM for the size of graph you intend to visualize (*daVinci* will crash with a *X Error: BadAlloc* message as soon as no more memory is available). Switching off the pixmap cache slows down the scrolling performance, but has no effect on the results a user will get.

The online help system has two options to specify the filename of the Netscape Navigator application on your computer and the location of the HTML online documentation. *daVinci* constructs the Netscape call from these two options when the user activates the help system, e.g. by pressing a Help button in some dialog window. The **Online Documentation** field is an URL, specifying a directory containing the documentation. The requested HTML page, depending on the kind of help the user wants to get, is added behind this URL. Usually, the documentation can be found in the *docs* directory of the *daVinci* package. This is the default place where *daVinci* looks if the **Online Documentation** field is empty. The **Netscape Application** field must specify the valid command to start Netscape Navigator 1.1 or higher on your computer. If you have problems to start the online documentation from *daVinci*, then check this condition by typing the content of this field in a UNIX shell to see if Netscape will be launched.

The last set of dialog items deal with the printing management which can be accessed by using the File/Print... dialog window. The **Print Command** is the UNIX command (probably with printer options) to print a PostScript file on your computer. Of course, the name of the PostScript file is not entered here, it is a temporary file created when the graph is actually printed. The **Measures** radio button specifies the measure unit (cm or inch) used to display page and graph size in the print dialog window.

The general settings dialog window can also be used mouseless with keyboard operations.

Dialog Items of the General Settings Window

- **On Selection in Survey View**
  Used to switch the auto-selection behaviour. If auto-selection is on and the user selects a node in a survey view, then *daVinci* automatically scrolls to the same node in the corresponding detail view (details about multi-view can be found in the concept document).

- **Speed of Animation**
  With this slider, the scrolling animation speed can be set. Animation is used for node selections in a survey view (see above), for the navigator and the find operation to move the visible part of a base window towards a particular node.

- **No Animation**
  Use this check button to turn off the scrolling animation. Animation is used for node selections in a survey view (see above), for the navigator and the find operation to move the visible part of a base window towards a particular node.

- **No Pixmap Caching**
  Select this check button to turn off the cache used for storing already drawn parts of a graph visualization. This should be done when *daVinci* crashes in situations where no more memory is available to draw a hugh graph (i.e. when you get error messages like *X Error: BadAlloc*).

- **Netscape Application**
  UNIX Command to start Netscape Navigator V1.1 or higher on your computer.

- **Online Documentation**
  Directory where the *daVinci* online documentation is located. This is usually the *docs* directory of the *daVinci* distribution (i.e. *$DAVINCIHOME/docs*, but do not type shell variables such as *$DAVINCIHOME* in the dialog, because Netscape does not understand this). The directory entered here is an URL, so for a *daVinci* documentation on the local file system, the URL has to

start with '*file:/*'.

- **Print Command**
  UNIX Command to print a PostScript file on your computer, probably with printer options, but without using a filename.

- **Measures**
  Use these radio buttons to switch between cm and inch measure units in the File/Print... dialog window.

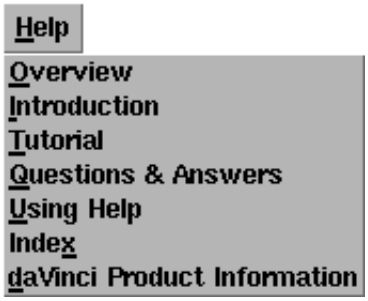
- **Apply**
  Push button to accept the current settings of this dialog window. Without pressing **Apply**, modifications to these options are not considered by *daVinci*.

- **Cancel**
  Push button to close the general settings dialog without performing any action.

- **Reset**
  Push button to undo any modification to these options unless **Apply** is not pressed. So **Reset** goes back to the current settings.

- **Help**
  Push button to start the online help system with this page.

Help Menu



The Help menu contains operations that provide user help facilities. To show help information about a particular topic, Netscape Navigator (version 1.1 or higher) is instructed to display pages of the *daVinci* HTML online documentation. This requires a proper installation of *daVinci* or a manual configuration of the help system in the Options/General Settings... dialog window. Read about using the help system for more details.

Note: Menus can also be used mouseless with keyboard commands.

Help Facilities of the Online Documentation

- **Overview**
  Instruct Netscape to display the Overview page of the online documentation.
- **Introduction**
  Instruct Netscape to display the Introduction page of the online documentation.
- **Tutorial**
  Instruct Netscape to display the Tutorial page of the online documentation.
- **Questions & Answers**
  Instruct Netscape to display the Questions & Answers page of the online documentation.
- **Using Help**
  Instruct Netscape to display the Using Help page of the online documentation.
- **Index**
  Instruct Netscape to display the Index page of the online documentation.
- **daVinci Product Information**
  This operation shows a window with product informations. It is the same window which will appear when menu View/daVinci Info... is selected.

daVinci V2.1 Online Documentation - Page update: June 15, 1998

Overview -> Tutorial

daVinci - Online Documentation V2.1

Tutorial

This chapter covers a tutorial for a guided tour through *daVinci* V2.1. The tutorial is written for beginners, so experienced users may skip this chapter. Please read the sections in the given order.

- **Before you begin**

- **Section 1: Managing Files and Basic Tasks**
- **Section 2: Optimizing the Graph Layout**
- **Section 3: About Scaling, Abstractions, and Navigation**
- **Section 4: Multi-Graph and Multi-View**
- **Section 5: Using the Graph Editor**

daVinci V2.1 Online Documentation - Page update: June 15, 1998

Before you begin

Before you begin with the *daVinci* tutorial, be sure to read at least the following documents:

- Introduction to *daVinci*
- Using the Online-Documentation

A closer look to *daVinci's* concepts might be a good idea, but is not essential for the tutorial.

End of introduction
Go back to the Tutorial Overview

Section 1: Managing Files and Basic Tasks

This tutorial section covers the following exercises:

- Loading a Graph from File
- Starting the Graph Layout Algorithm
- Saving the Status of a Graph
- Printing a Graph Visualization

Loading a Graph from File

After starting *daVinci* without any options or arguments, an empty *base window* will appear (click here to get an image and more informations about the base window). In the upper part of the window, you have *pull down menus* where you have access to all functions. The *icons* at the left side are buttons for direct access to frequently used menu operations. We will come to this later. One important component of the base window is the lower *footer area* where status messages are displayed. Keep your eye on this area to be informed about important system events, for example when you have to wait for an operation to be finished.

Now we will load a graph for visualization. Graphs are stored in files with a *.daVinci* suffix, containing a so called *term representation*. This format only defines the structure of a graph, but has no layout informations such as the coordinate of a particular node. The term representation format is discussed in detail in the reference chapter. To load a graph, you need the file browser for selecting a file name. You can get a browser by selecting menu **File/Open...** (i.e. move the mouse pointer to the **File** pull down menu and press the right mouse button which will show the *File menu*. Then move the mouse pointer to the **Open...** menu entry by still pressing the right mouse button and finally release the mouse button when you are over the **Open...** menu entry). This will pop up the *file browser* for opening a graph.

The file browser has two different lists: One for the directories inside the current directory and one for the files in the current directory. The current directory is always displayed in the **Selection** field of the file browser. Now you can browse through the directories of the file system of your computer by double-clicking the list elements of the **Directories** view with the left mouse button (the left mouse button is used to select or manipulate objects visible on the screen, whereby the right button is used to choose from a menu). A double-click on '..', the first element of the directory list, will move to the parent directory. When there is not enough place to show all directories or files, then you can use the vertical scrollbars at the right side of the lists to move the focus. Finally, go into the *example_graphs/* directory of the *daVinci* distribution. Ask your administrator if you are not sure about the location of the *daVinci* distribution on your file system.

When you are inside the *example_graphs/* directory, click on the **Types** option button to select the **Graph Files** filter. This will remove all files without suffix *.daVinci* from the files view. Now double-click on file *graph_example.daVinci* to load this graph in *daVinci*. This finishing action will automatically close the file browser.

Starting the Graph Layout Algorithm

After successfully loading the graph *graph_example.daVinci* from file, the visualization should immediately appear in the *graph area* of the base window. This graph is larger than the base window, so you can only see a portion of the whole graph visualization. Use the two scrollbars of the base window to move the focus to other areas of the graph. A more comfortable method to set the focus in a large graph visualization is 2D-scrolling. This operation is available inside the graph area of a base window if you press and hold down the middle mouse button while moving the mouse. Try this with the current graph and you will see how the system scrolls in both directions simultaneously by following the movements of the mouse.

As you can see on the screen, all graph nodes are automatically placed on horizontal levels such that all (directed) edges between two nodes are pointing downwards. This is called a hierarchical visualization of a directed graph. Other styles of graph layout are currently not supported by *daVinci*. Background information about graphs can be found in the concept document.

Except of assigning nodes to levels, the graph layout algorithm is not used directly after loading a graph, to display it as soon as possible. By looking at the graph visualization, you can see that the layout is not as good as it can be. For example, several edges cross eachother, which makes the layout confusing and somehow unattractive. You can see the total number of edge crossings of the current graph layout by opening a statistic window with menu View/Graph Info.... Improving the graph layout can be done by reordering the nodes at each level such that the number of edges is reduced. *daVinci* offers an automatic graph layout algorithm for this task. Start this algorithm by using menu Layout/Improve All. After doing so, you will notice a significantly improved layout. You can read more about graph layout in the concept document.

In *daVinci*, a graph is drawn top-down by default, so edges are usually pointing downwards. You can choose one of the other three layout orientations (upwards, to the left or to the right) with menu Layout/Orientation. For example, try to rotate the layout by selecting menu **Layout/Orientation/Left to Right**.

Saving the Status of a Graph

Assuming that you are a careful person, you might wish to store the current layout of the graph in a file. As stated above, a term representation file does not contain layout informations, so this way your modifications will be lost in a later session. To store a graph together with informations about the current layout (node coordinates etc), you can choose *daVinci's* alternative graph storage format, called *status*. A status also contains most of the user interface settings, so by loading a status file, a previous session can be restored very easily.

Again, you need a file browser to save the current layout to a status file. One way to do so is by selecting menu File/Save As.... But at this time, we will use the icon bar of the base window for more comfort. Move the mouse pointer to the save button which looks like this:

Button of the icon bar to save a graph

When the mouse pointer is over this button, a message is displayed in the footer area with a short explaination of the button's function. Now press the button to get the file browser for saving files. Pressing this button on the icon bar is the same as selecting menu File/Save As....

In the file browser, you have to choose the file type you want to save by using the **Types** option button: Either **Status Files** (default) or **Graph Files**. Saving the current graph in term representation format by using type **Graph Files** does not make sense at the moment, because the structure of the graph is still the same, which means that you would get the same file as loaded before. So use the default type **Status Files** such that the title of the browser window shows "Save As Status".

The browser usually suggests a name for the file you want to save. The filename of the graph (term representation) you have loaded before is *graph_example.daVinci*, so the suggested filename is *graph_example.status*. Status files should usually have suffix *.status*. Unfortunately, the suggested filename already exists (as you can see in the **Files** view of the browser) and you do not want to overwrite it. By the way, the browser will ask the user before overwriting files. So choose a new name for your file by editing the **Selection** field of the browser. Click in the field to get a text cursor and erase the filename (but not the directory) with the Backspace key. Then enter a new filename, e.g. *my_graph.status*, and press the **Save** push button to start the saving operation.

If you do not have write permissions for the *example_graphs* directory of the *daVinci* distribution, then saving will fail and a notice window will appear. In this case, open the browser again to repeat this operation, but at this time choose another directory where you have permissions to write, for example your home directory.

Printing a Graph Visualization

You can print a graph on a PostScript printer directly from *daVinci* to get the current layout on paper. For this operation the print dialog window is needed which will appear after selecting menu File/Print... (or by pressing the printer button of the icon bar).

But at this time, we will use the keyboard instead of the mouse for choosing this menu operation. Press the keys **Alt** and **f** simultaneously which is usually written as **Alt+F**. This will open the File menu. **f** is the underlined character (mnemonic) of the File menu, the keyboard equivalent for mouseless operation. After opening the menu with these two keys, you can navigate through all menus with the four cursor keys of the keyboard. By pressing **Return**, you select the currently highlighted menu entry. In an opened menu, you have another option to choose from a menu by pressing the mnemonic character (at this time without the **Alt** key!) of one of the menu entries. For example, when the **File** menu is open, you select **Print...** by pressing the **p** key. So the overall keyboard command to select this particular menu is **Alt+F P** (upper case letters are not significant here). You can read more about keyboard usage by following this reference.

After opening the print dialog window with one of the methods described above, you have the opportunity to set some options before the file will actually be printed. Lets asume you want to print the currently loaded graph on one of the PostScript printers in your network. First of all, you need to know the UNIX command used for printing PostScript files at your site. For example, if you usually use command

lp -d our_ps_printer my_file.ps

to print file *my_file.ps* on printer *our_ps_printer*, you have to enter *lp -d our_ps_printer* in the **Print Command** field of the print dialog. Be sure that you have set the correct page size supported by your printer. If this is not the default format **A4**, then choose another setting with the **Page Size** option button and press the **Fit to Page** button afterwards to ensure that the graph will be completely visible and centered on the paper. Finally, you can send the graph in PostScript format to the printer by pressing the **Print** push button.

End of section 1
Go back to the Tutorial Overview

---

daVinci V2.1 Online Documentation - Page update: June 15, 1998

Section 2: Optimizing the Graph Layout

This tutorial section covers the following exercises:

- Setting the Layout Dimensions
- Horizontal Fine Tuning
- Vertical Fine Tuning

Setting the Layout Dimensions

First you need to load a graph in *daVinci* by using menu File/Open.... We suggest to use file *graph_example.daVinci* in the examples directory of the *daVinci* distribution again (refer to section 1). Afterwards, start the graph layout algorithm with menu Layout/Improve All.

Now we will modify the dimensions of the current graph visualization. This can be done with the layout dimensions dialog, available after selecting menu Options/Layout Dimensions. In the dialog you can find an option button **Size (pt)** to set the font size (measured in *point*). Change the font size to 24. You can see that the other dimensions in this window are automatically adjusted to keep the layout proportional when the nodes become smaller or larger. At this time the graph visualization is still the same, because the modification will take effect only after pressing the **Apply** push button of the dialog. So press this button now. In the footer area of the base window you can see a message when the new font is loaded. This may take some time. Afterwards the dialog window disappears and the graph is redrawn by using the new font size.

Select menu Options/Layout Dimensions again to set some other dimensions. First, choose 10 as the new font size. Then move the **Gap Height** slider to 10 by using the mouse. This dimension specifies the vertical node distance between the horizontal levels (in pixel). Maybe it is not possible for you to choose a distance of 10 by moving the slider, because the next value after 9 is 11. In this case set it to 9 and click at the right side of the slider. This will move the slider one step to the right to get a 10. Finally set the **Gap Width** to 10 pixel, too. To do that in another way this time, select the text entry field between the **Gap Width** label and the slider, delete the current setting with the backspace key and type 10. The gap width is the minimal distance between the nodes at each level. The actual distance between two nodes may be greater than this value (to get a better layout with straighter edges), but never smaller. Press the **Apply** button to use these settings and the graph will be redrawn much more compact.

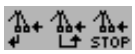
Horizontal Fine Tuning

Now we will modify the automatically calculated graph layout with the mouse. This kind of direct manipulation is called *fine tuning*. To do so, select some graph node with the mouse. Then move the mouse slowly to the left and to the right by still holding down the left mouse button (this is called *dragging*). Be careful not to drag the node up- or downwards to the next or previous level. This will cause a vertical fine tuning which is discussed in the next exercise. By moving the mouse to the left and to the right, you can see how the node position is changed. Move the node towards its left or right

neighbour node at the same level. When you have reached the minimal distance, the neighbour node is moved as well in the same direction, and so on.

We will now manually change the order of nodes at a level. Select menu Options/Layout Algorithm to open the layout algorithm dialog. Under **MANUAL LAYOUT** you can find three radio buttons to set the constraints of the horizontal fine tuning. The current setting is **Preserve Node Order** to move a touched neighbour node as well. This is what we have just seen before. Now select button **Modify Node Order** and press the **Apply** push button afterwards. Move some node again towards its left or right neighbour node. This time, when you have reached the minimal distance, both nodes will exchange their position. This operation allows you to change the order of nodes at a level, for example to eliminate obvious edge crossings by hand which have been ignored by the graph layout algorithm.

You might have noticed that by selecting the **Modify Node Order** constraint, the last group of buttons on the icon bar was modified, too. For more comfort, the last three radio buttons on the icon bar can also be used to set the fine tuning constraints.



Buttons of the icon bar to control fine tuning

So select the last button (the one with STOP) of the base window's icon bar to set the **Fixed Neighbour Nodes** constraint. By moving some node, you can see that this setting protects the position of the left and right neighbour node when they are touched by a dragged node.

Vertical Fine Tuning

The last exercise of this chapter deals with vertical fine tuning, used to move a selected node to a different level. From the previous section of the tutorial you already know that *daVinci* generates a hierarchical layout were all edges are pointing downwards. By moving one node to a new level, the hierarchy property of the drawing might be violated. So for example, when you move a node one level downwards, then probably the children of this node need to be moved as well (and their children, and so on) to guarantee that each child node is at least one level under its parent node.

Ok, enough explanation. Just do it: Select a node, for example one at the topmost level, hold down the left mouse button and move the mouse slowly downwards. As soon as the mouse pointer reaches the center coordinate of the next level, a dynamic layout process starts to move the node to the new level and to rearrange the subgraph with the children (as it is explained above). Dynamic layout may take some time, especially when large parts of the graph need to be rearranged. So either you need to be patient or you have to order a more powerful computer. ;-)

It is possible that the total graph layout is getting worse after several vertical fine tuning steps. So start the layout algorithm to clean up that mess. You can see that *daVinci* keeps the nodes at their current level to preserve your vertical fine tuning operations. If you want the nodes back at their initial level after starting the layout algorithm, then you must deactivate the **Keep Nodes at Levels** check button (such that it is no longer highlighted in red) in the layout algorithm dialog, available after selecting menu Options/Layout Algorithm. Do not forget to press the **Apply** button in the dialog to activate this new setting!

End of section 2
Go back to the Tutorial Overview

daVinci V2.1 Online Documentation - Page update: June 15, 1998

Section 3: About Scaling, Abstractions, and Navigation

This tutorial section covers the following exercises:

- Get an Overview with Scaling
- Using Abstractions to simplify a Layout
- Navigation Features to browse in a Graph

Get an Overview with Scaling

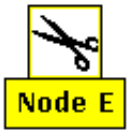
First you need to load a graph in *daVinci* by using menu File/Open.... We suggest to use file *graph_example.daVinci* in the examples directory of the *daVinci* distribution again (refer to section 1). Afterwards, start the graph layout algorithm with menu Layout/Improve All.

Scaling is a feature to reduce the size of a probably large graph visualization for getting an overview. In a scaled graph, some details are dropped (e.g. the text of nodes is not drawn), but all the other interactions are still available. So for example, you can even use fine tuning in a scaled drawing of a graph. One way to reduce the scale is by selecting menu View/Fit Scale to Window. Do this right now! A new (maximal) scale is calculated and set such that the graph is completely visible in the base window. Of course, this modifies the scale only if the graph is larger than the base window.

Now open the scale dialog by using menu View/Scale.... In this window you can set the scale to any rate between 1% and 100%. Simply move the slider to some arbitrary values and observe the effect on the visualization. Pressing button **Fit to Window** in the dialog is the same as selecting the menu you have just used before. You cannot magnify the graph this way (i.e. there is no scale >100%), but you can use a larger font for magnification as it is discussed in the previous tutorial section. Before going to the next exercise, set the scale back to 100% (there are different ways to do so, for example by clicking on the **Large** label in the scale dialog) and afterwards finish the dialog by pressing the **Close** push button.

Using Abstractions to simplify a Layout

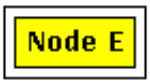
Interactive abstractions are used to hide particular parts of a graph that may be uninteresting or confusing. Abstractions are temporary operations that can be undone at any time. Select some node in the graph that has at least one child node. Now you can fade out the subgraph of this node by selecting menu Abstraction/Hide Subgraph. After doing so, the children of the node are removed from the visualization and the selected node is drawn with an icon showing a pair of scissors to emphasize the applied abstraction:



Node with a hidden subgraph

You can restore the subgraph of the selected node by using menu Abstraction/Show Subgraph. Make sure to select a node with a hidden subgraph, otherwise the show operation is not available. Try to hide three or four large subgraphs. You can see that the position of the other nodes is not affected by this operation, to reserve the place until the subgraph is shown again. So maybe a lot of unused space is present in the visualization after hiding some subgraphs. Take advance of this space by selecting menu Layout/Improve Spacing which will result in a more compact layout. Now you can restore the original layout by using menu Abstraction/Restore all Subgraphs to restore all subgraphs. Because the place of the subgraphs is occupied after compactification, the layout may not look as it looked before the abstractions, but with the new incremental layout introduced in *daVinci* V2.1 you'll get a quite good layout compared to the messy one you got in former versions.

Another abstraction is hiding all the edges of a selected node: Both the incoming (from the parent nodes) and the outgoing edges (to the child nodes). The three operations for edge abstractions are also available in the Abstraction menu. After hiding the edges of a selected node, they are removed from the graph visualization and the node is drawn with a rectangular border to emphasize the applied abstraction:



Node with hidden edges

Navigation Features to browse in a Graph

For the last exercise we will come to the operations of the Navigation menu. First, select an arbitrary node in the graph. To see the parent-, sibling- or child nodes of the current selection (the siblings are the children of all parents of a particular node), choose one of the first three menu entries of the Navigation menu, e.g. Navigation/Select Children. This will clear the current selection and the corresponding set of parent, sibling or child nodes is selected, depending on the choosen operation. You can even select more than one node before using these operations to find all of their parents, siblings or children (the Using the Mouse document explains how to select more than one node at a time).

A very useful feature is the navigator dialog which appears after using menu Navigation/Navigator.... Select exactly one node in the graph and press one of the four arrow buttons of the navigator dialog to go to the next node in the corresponding direction. This way you can easily browse a graph by hopping from node to node. The speed of animation, used for scrolling to the new focus point, can be adjusted with menu Options/General Settings.... You can also use the cursor keys of the keyboard to trigger navigation, so usually you do not need the navigator dialog. Check button **Structural Navigation** is explained on the navigator dialog help page of the user interface documentation.

Finally we will search for nodes with a specified text by using the find dialog. Select menu Navigation/Find... to pop up this window. Now enter the substring you are looking for in the **Find** field. If you have loaded file *graph_example.daVinci*, then for example type a 6 and press the **Find** push button afterwards. This will search for a node with a text containing the 6. Press **Find** again to go

to the next match.

After playing a little bit with these features, close both the find- and navigator dialog by selecting the **Close** button, if not already done.

End of section 3
Go back to the Tutorial Overview

---

daVinci V2.1 Online Documentation - Page update: June 15, 1998

Section 4: Multi-Graph and Multi-View

This tutorial section covers the following exercises:

- Multi-View: Using many Views for one Graph
- Working with a Survey View
- Multi-Graph: Visualizing different Graphs simultaneously

Multi-View: Using many Views for one Graph

First of all you need to load a graph. This time we do it by using a command-line argument when *daVinci* is started. If you have already started *daVinci*, quit it now with menu File/Exit daVinci. Start the system again by appending the filename of a graph or status file to the command-line. As you have already guessed, we suggest to use file *graph_example.daVinci* in the examples directory of the *daVinci* distribution again. So for example, when you are in the topmost directory of the *daVinci* distribution, you can type the following command in a UNIX shell:

daVinci example_graphs/graph_example.daVinci

After you have got a base window with a graph visualization, open a second view of this graph by selecting menu View/Open New View. This will pop up an additional base window showing the same graph. In fact the two views (base windows) are connected with each other, so interactions in one view are also applied to the other view (except of scrolling and setting the scale). Make some fine tuning operations by moving nodes to new positions. You can see that both views will be updated when interactions take place in any of the two windows.

Open even more views of this graph from any existing window to get a total number of, say, five windows. The maximal number of open base windows in *daVinci* is 64. You can now close each individual view with menu File/Close without quitting *daVinci*. The last base window cannot be closed.

Working with a Survey View

A special view with some additional features is the *survey view* which can be opened from any view (except of another survey view) by using menu View/Open Survey View. So make sure that you have one *daVinci* base window showing a large graph (e.g. *graph_example.daVinci*). We will call this window *detail view*. Reduce the window size such that you can only see a small portion of the whole graph. Now, select menu View/Open Survey View in the detail view to get a new survey window. As with all views, detail- and survey view are coupled with eachother to show the same graph layout.

A survey view displays the currently visible graph area of the corrsponding detail view with a blue rectangle which is continuously updated. Use navigation or move the scrollbars in the detail view to see how the blue rectangle will follow these movements in the survey view to visualize what is visible in the detail view. When you do not see a blue rectagle in the survey view, then either the graph is not

completely visible in the survey view (correct this by using menu View/Fit Scale to Window) or it is completely visible in the detail view (use a larger graph in this case).

Another nice feature of the survey view is that after selecting a node in the survey view, *daVinci* will automatically scroll to this node in the corresponding detail view with animation. Try this by clicking on some nodes in the survey view.

Multi-Graph: Visualizing different Graphs simultaneously

daVinci V2.x does not only allow a user to have many views (base windows) to one graph, but also to load different graphs at the same time which are visualized in different base windows. Before working in multi-graph mode, be sure to have at least two open view windows, because otherwise there is no second window and loading a new graph in one window will eliminate the previous graph there. To load a second graph, simply use the File/Open... menu in one of the views. This will load the new graph only in the window where the load operation was executed. The other views are not affected. So from a technical point of view, the window where the graph was loaded is disconnected from the other views to act independently.

Of course, the window with the new graph may have its own views, too, and so on. Keep in mind that the maximal number of open base windows in *daVinci* is 64 and that multi-view and multi-graph is an expensive feature (think about the required memory!), so do not overstrain your computer with too many graphs and windows.

End of section 4
Go back to the Tutorial Overview

---

daVinci V2.1 Online Documentation - Page update: June 15, 1998

Section 5: Using the Graph Editor

This tutorial section covers the following exercises:

- Starting the Graph Editor
- Building a new Graph from Scratch
- Modifying existing Graphs
- Editing several Graphs
- Using Drag and Drop

Starting the Graph Editor

The graph editor is an external application that communicates with *daVinci* by using the API. The current version is GraphEditor V2.1. With the editor you can interactively edit the structure of a graph which is currently loaded in *daVinci*, e.g. to insert or delete nodes and edges. The responsibility of the graph editor application (and any other *daVinci* application) is to maintain the structure of the graph, whereas *daVinci* can only visualize the graph without permission to manipulate the graph's structure on its own. This topic is discussed in detail in the API introduction.

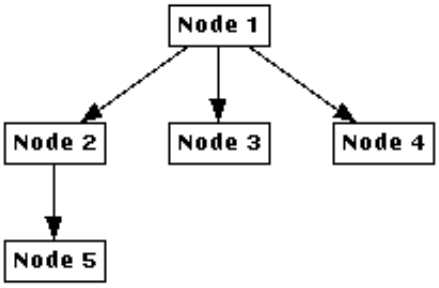
The user does not see anything of this system architecture with two different processes communicating with eachother. The user controls the application (i.e. the graph editor in this case) from inside the *daVinci* user interface, so the easiest way is to think about the graph editor application as a "plug-in" to extend the basic functionality of *daVinci*.

Before using the graph editor, you have to connect it to the *daVinci* API. This can be done in *daVinci* at runtime by using menu File/Connect Application... (another method is to start *daVinci* with option -graphedit). By using this menu, a file browser appears to let you select the binary file of the application that should be connected. The filename of the graph editor binary is *grapheditor*. You can find it in directory *daVinci_V2.1/grapheditor/* of the *daVinci* distribution. If you do not know where *daVinci* is installed on your computer, type *echo $DAVINCIHOME* in a UNIX shell to display this location.

Now move to the graph editor directory with the file browser and double-click on file *grapheditor* there. This will start and connect the editor. Afterwards you should see a status message "GraphEditor V2.1 (Drag & Drop)" in the lower right footer area of the *daVinci* base window and seven new graph editor icons on the icon bar of the *daVinci* base window. Do not worry, the usual *daVinci* icons are still available! You can switch between the *daVinci* icons and the graph editor icons by using the small arrows (,) which are at the lower end of the icon bar.

Building a new Graph from Scratch

With the connected graph editor application we can now interactively construct our own graph in *daVinci*. Start with a root node by selecting menu Edit/Node/Insert Root. This creates a single node, more precisely a white box with text "Node 1". Like all inserted nodes, the new node is automatically selected after creation (i.e. it has a black shadow). Lets turn the selected node into a parent by using menu Edit/Node/Insert Child. This is a context sensitive menu which means that the currently selected node will become the parent of the new inserted node by creating an edge as well. You can also insert child nodes by using the nodes popup-menu. Just press the button BMenu over the parent node and choose option Insert Child. Insert some more child nodes such that you will get this graph:



If you have made a mistake with your graph, you can always delete a single node with menu Edit/Node/Delete Node or a node together with its subgraph using menu Edit/Node/Delete Subgraph (refer to this link if you want to know what exactly a "subgraph" is). These menus are also available in the node popup-menu.

Now we will create some edges between existing nodes. First select "Node 4" and then select "Node 5" such that only these two nodes are selected. The order is important, so click "Node 4" first! Hint: you have to hold down the **Shift** key for multiple selection. With two selected nodes, menu Edit/Edge/Insert Edge becomes available. Select this menu to create a directed edge pointing from parent "Node 4" to child "Node 5".

What would have been happened in this situation if you had selected "Node 5" first and then "Node 4"? Try it by removing the previously inserted edge to go back to the old graph shown above. To do so, select this edge and use menu Edit/Edge/Delete Edge (or the same menu in the edge popup-menu). Insert the edge again, but do not forget to select "Node 5" first and "Node 4" afterwards at this time. As you can see, the parent and child node of the inserted edge are now exchanged. This will result in a completely different visualization for a hierarchical graph drawing where edges are always pointing downwards from the parents to the children.

By the way, you already might have been noticed that you can also use the graph editor icons on the icon bar of the *daVinci* base window for quick access to the seven most frequently used operations of the graph editor.

Modifying existing Graphs

After getting familiar with the basic editing operations, we will now come to some more advanced features of the graph editor. They will be explained based on one of the example graphs of the *daVinci* distribution, to see how modification of an existing graph works. You should now save your current graph before with menu File/Save As... if you want to keep it for a later session. After doing this, load file *graph_example.daVinci* or *graph_example.status* in the *example_graphs/* directory of the *daVinci*

distribution (remember: environment variable $DAVINCIHOME, entered in a shell, points you to the location).

Now you should see a large graph in the *daVinci* base window. You can modify the structure of this graph with the basic editing operations described above. But here is something new: select an arbitrary node in this graph and choose menu Edit/Node/Attributes... afterwards. A dialog window will appear where you can modify all attributes of this specific node. You can also select the third graph editor icon, simply double-click on a node or directly select the menu from the node popup-menu by pressing button BMenu over the node to get this dialog. Play a little bit with the dialog items of this window and press the **Apply** button to consider your modifications for the graph visualization. For more convenience, the window will remain open until you press the **Cancel** button. When you select a new node in the graph, the window is updated to show the attributes of the current selection. You can even modify the attributes of many selected nodes in one step.

For example, change the shape to **Icon** for one specific node and enter the string *mfr.xbm* in the **Icon File:** field which will become enabled. Do the same with another node, but use the string *mawe.xbm* this time. Wow, you can see photos of the two *daVinci* developers! By the way, this is not a built-in easteregg. Both .xbm files are in the *icons/* directory of the *daVinci* distribution, which is part of the icon search path by default (refer to environment variable $DAVINCI_ICONDIR).

Different colors are also available to be used as node attribute. If you do not like the given palette, then you could set your own preferences with the color editor which appears after double-clicking on one of the colors. Next, change the text of a particular node with the attribute dialog. You can even get nodes with multi-line text by typing return characters in the text entry field. As you have already seen, newly inserted nodes get a default text. You can specify the pattern for this text with menu Edit/Node/Naming Scheme..., whereby the special character # is a placeholder for the current node number.

Edges also have attributes. Double-click on one of the edges in the graph to get the edge attribute dialog. Modify the attributes of some edges to see what happens.

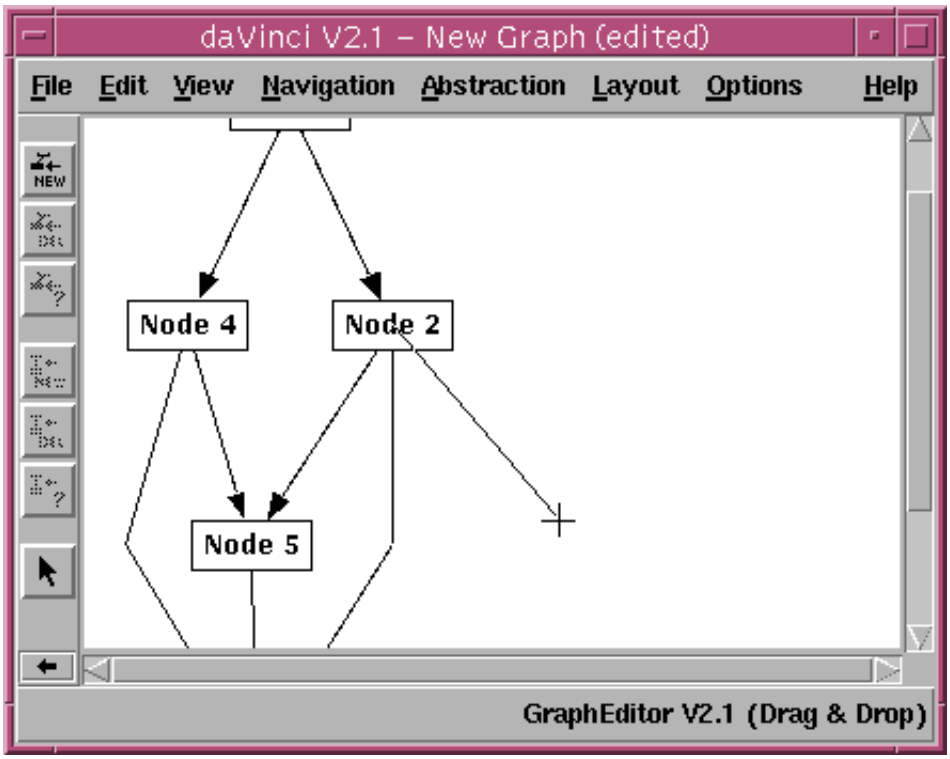
Editing several Graphs

In the graph editor it is even possible, to edit several graphs at a time. To open a window for a new graph, choose menu File/New. This will open a new base window. Try to insert some nodes in the new window and you will see, that this doesn't change the graph in the old window.

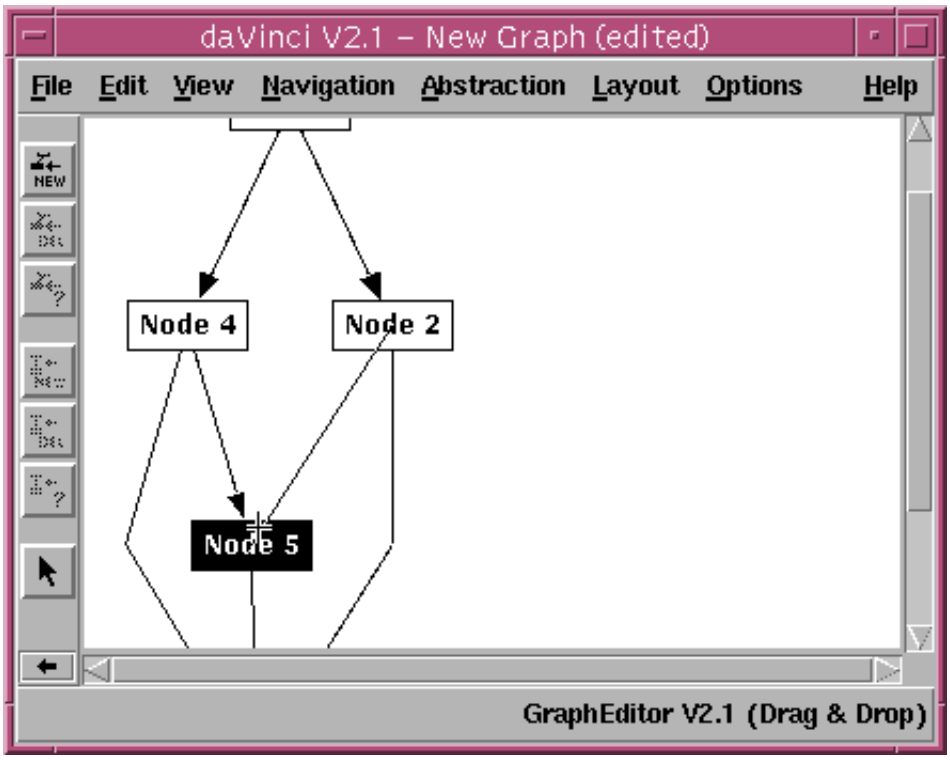
Using Drag and Drop

In the above examples you have used menus and icons to manipulate the graph. With the Drag and Drop feature of *daVinci* you are able to directly manipulate the graph with the mouse. Just click the button BTransfer somewhere in the graph-area of a *daVinci* base window. A new node will be inserted directly below the mouse pointer, if this is possible according to the *daVinci* layout rules. Now press the button over an existing node and drag the mouse while holding the button. An abstract edge will appear. Move the mouse pointer somewhere over the graph area but not over another node and release the button. A new child node of the node you've first selected, together with the connecting edge will be inserted. Last but not least insert a new edge between two nodes. Again press the button over an existing node but now move the pointer to the designated child node. You'll see that all nodes you pass with the pointer aredrawn inverted to indicate which node would be the child node. After releasing the button over a node, a new edge is inserted. You can even insert a self-edge, if you release the button over the node where you've started after you've moved the mouse a little bit.

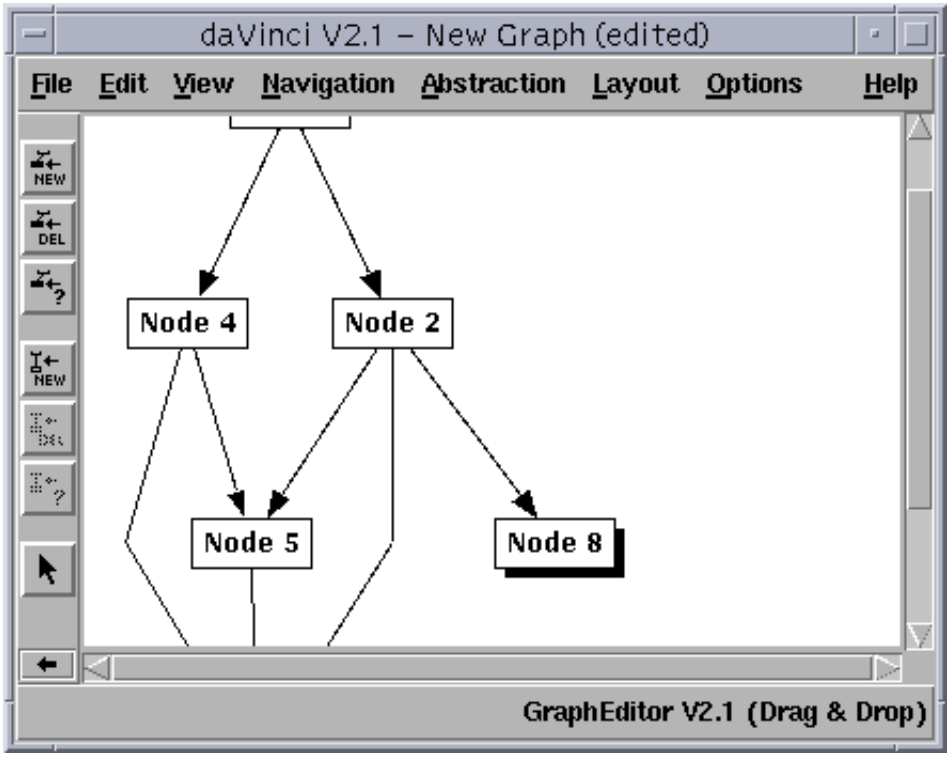
The following screenshots show you some of the described drag and drop actions:



Insert a new child node with drag and drop.



Insert a new edge with drag and drop.



Graph after applying both actions.

Another way to use drag and drop is to copy a node. Just press the Shift-Key and the button BTransfer over a node. The node is drawn inverted to indicate the dragged node. Than move the mouse to another place in the graph-area or if you are editing several graphs, move it even to another window and release the button. A copy of the node (with the same attributes and text) is inserted. If you've released the button over another node, the copy of the node is inserted as a child node of the node under the mouse pointer, together with its connecting edge.

End of section 5
Go back to the Tutorial Overview

---

daVinci V2.1 Online Documentation - Page update: June 15, 1998

Overview | Intro | Reference | Tutorial | Questions & Answers | New Features | Index

Overview -> Questions & Answers

daVinci - Online Documentation V2.1

Questions & Answers

This document describes solutions for frequently asked questions about *daVinci* V2.1.x.

Overview:

- **Availability:**
  - What is the current release of *daVinci* and where can I get it?
  - Is *daVinci* available for MS-Windows?
  - Can I get the sources of *daVinci*?
  - Where can I get *daVinci* for my UNIX brand?
  - Your ftp-server is too slow. Are there any mirrors?
- **License:**
  - We are a company. Do we have to pay for a commercial license?
- **System Features:**
  - Does *daVinci* allow manipulation of a graph?
  - Is it possible to send *daVinci* graphs to a plotter?
- **Graph Layout:**
  - Is it possible to draw cyclic graphs?
  - How can I get edge labels?
  - Are nested graphs (i.e. a graph node is a box including another graph) supported?
  - Is it possible to specify the root nodes in a graph visualization?
  - Nodes drawn with circles don't have a common radius. Can I change this?
- **Term Representation (*daVinci's* external Graph Format):**
  - Can I use two sets of nodes and edges instead of the term representation?
  - How do we add comments to a *daVinci* term representation file?
  - Is it possible to load a graph in adjacency matrix format?
  - How large can the string of a node or edge ID be, so *daVinci* can differ them?
  - Where can I find the documentation of the status file format?
- **Application Programmer Interface (API):**
  - Can I connect *daVinci* and my program written in *XXX* by using the API?
  - Is it possible to construct graphs directly from a program written in *XXX?*
  - I get selection answers after sending commands, but before receiving the 'ok'!
- **Graph Editor Application:**
  - Can I insert self edges with the graph editor?
- **Problems:**
  - I always get a core dump after starting *daVinci*. What is going wrong?

- *daVinci* crashes with *"X Error: Bad alloc"*. What can I do?
- I get an error "Tk: can't read "tk_version": no such variable" after starting *daVinci*.
- I cannot start *daVinci* V2.1 on SunOS 4. There are errors from ld.so!
- I cannot start *daVinci* V2.1 on DEC UNIX. There are errors from /sbin/loader!
- Does *daVinci* have any problems with the year 2000?

- **Help System / Online Documentation:**
  - *daVinci* doesn't talk with Netscape. What can I do??
  - What can I do if Netscape cannot find a file after starting the help system?
  - I need to print a paper version of the documentation. Is it available?
- **More Information:**
  - Are any papers available about *daVinci*?
  - Is there a mailing list?
  - Where can I find a paper about the layout algorithm(s) used in *daVinci*?

Availability:

Q: What is the current release of *daVinci* and where can I get it?

A: The current release is *daVinci* V2.1. Informations about downloading *daVinci* by ftp can be found in the online documentation.

Q: Is *daVinci* available for MS-Windows?

A: No, not at the moment. Although the *daVinci* V2.1 user interface is implemented in Tcl/Tk, which is also available for Microsoft Windows 95 and NT, some parts of the source code are not portable per se (e.g. the highly optimized drawing engine, which is based on Xlib, and the API which currently implemented with UNIX pipes). Because of limited resources, it is not possible for the *daVinci* project to go ahead with a MS-Windows port at the moment. Further, the port cannot be done by a third party due to the need of fundamental modifications in the *daVinci* kernel.

Q: Can I get the sources of *daVinci*?

A: Sorry, but only the binary distribution of *daVinci* is public available. First of all, the system is written in the pure functional language *ASpecT* which is automatically translated to C. The generated C-code is more or less unreadable for a human being and functional languages are not yet mainstream. Moreover, the ASpecT language is almost unknown for people outside University of Bremen. So the sources are of minor use for most programmers. Second, the *daVinci* project would like to keep the sources for a potential commercial exploitation of the *daVinci* software in the future.

Q: Where can I get *daVinci* for my UNIX brand?

A: There are binary distributions available for all major UNIX platforms. Check the online documentation for details about downloading *daVinci*. If you miss support for a particular UNIX operating system and have some time, then maybe you can help yourself and other people by doing the port on your own. All you have to do is asking the *daVinci* developers for the compilation scripts. The only requirement for the port is that a C-compiler (gcc prefered) and Tcl7.4/Tk4.0 is installed on your computer.

Q: Your ftp-server is too slow. Are there any mirrors?

A: You can try the US mirror (*ftp://ftp.wustl.edu/graphics/graphics/packages/daVinci*) if transmission from University of Bremen (*ftp://ftp.tzi.de/tzi/biss/daVinci*), the primary server for

daVinci, is too slow. There are even more mirrors, but it may take some time to see a new release there. So check if the release is up-to-date. You can find more mirror server by using the *xarchie* tool which may be installed on your computer (ask your system administrator if not).

License:

Q: We are a company. Do we have to pay for a commercial license?

A: This depends on what you want to do. *daVinci* is licensed free of charge for non-profit use (refer to the license). So companies and other institutions can use the software for free for these kinds of application, e.g. to develop internally used software which is never given to customers (customers are people who pay a company or institution for their products or service). Bundling *daVinci* with other freeware software packages (which are available for no cost) is also considered as non-profit use.

But if *daVinci* is used for developing commercial software (which is not given for free to a third party), or if print-outs of *daVinci* graphs are presented to customers (e.g. for prototypes or advertising) or if *daVinci* is bundled with non-free software, then a commercial license is required. Please write an e-mail to the *daVinci* project to get the conditions of a commercial license.

System Features:

Q: Does *daVinci* allow manipulation of a graph?

A: Yes and no. There are a lot of interactive functions in *daVinci* to manipulate the *visualization* of a graph, e.g. scaling, abstractions, fine-tuning, etc. But there is no way to directly manipulate the *structure* of a graph, so *daVinci* is not a graph editor. This is one of the fundamental concepts of *daVinci* to be as generic as possible. The structure of a graph has to be controlled by an external application program which can be connected to the *daVinci* API.

One application program, bundled with each distribution, is the graph editor which allows you to interactively edit a graph currently visualized with *daVinci* (e.g. by adding and removing nodes and edges). Informations about the graph editor can be found in the online documentation.

Q: Is it possible to send *daVinci* graphs to a plotter?

A: In *daVinci* you can save a graph in PostScript format. If your plotter understands PostScript, it will work. If not, you need to find a converter from PostScript to your printer/plotter format.

Graph Layout:

Q: Is it possible to draw cyclic graphs?

A: Yes, although *daVinci* gives you a hierarchical graph layout with levels of nodes and directed edges between them, the system is also able to handle cyclic graphs (by transforming them internally into acyclic graphs).

Q: How can I get edge labels?

A: Sorry, but edge labels (i.e. a text next to an edge in the visualization) are currently not supported. The *daVinci* project knows about the strong user demand for edge labels, but this feature requires fundamental modifications to the layout engine which is not possible with the current development resources. Edge labels are still on the list for a future version of the *daVinci*

system.
At the moment there is a sufficient work-around for edge labels (by using graphical object *"text"*) which is described in the online documentation.

Q: Are nested graphs (i.e. a graph node is a box including another graph) supported?
A: No, not at the moment. This is also a feature scheduled for one of the next versions. We will tell you a secret. Parts (although still very buggy) of this new feature are already integrated in *daVinci* V2.1. Please contact the *daVinci* developers for informations how to test this feature.

Q: Is it possible to specify the root nodes in a graph visualization?
A: In most cases, the root nodes of the term representation are also the root nodes in the graph visualization. But if there are some edges pointing to a root of the term representation or if the graph is cyclic, then *daVinci* may choose another node as root and the term representation root becomes an ordinary node with parents. The user currently has no influence on this behaviour.

Q: Nodes drawn as circles don't have a common radius. Can I change this?
A: The width and height of graph nodes in the visualization depends on the text extension of each individual node. It is currently not possible in *daVinci* to specify the width and height of a node. All you can do is using one common length for all node strings (e.g. by filling up strings with spaces) and using font family Courier (refer to term representation, node attributes).

---

*Term Representation (*daVinci's *external Graph Format):*

Q: Can I use two sets of nodes and edges instead of the term representation?
A: The term representation is the only graph format currently supported by *daVinci*, so it is not possible to write down the nodes first and then the edges (or vice versa). For a work-around, you can use the API of *daVinci* V2.1 which has a command graph(update(...)) for updating an already loaded graph. Graph updates are nodes and edges that should be added or removed. So first you have to remove a probably loaded graph by sending API-command menu(file(new)) and then you can send the new graph by specifying its node and edges in the format of the *graph(update(...))* command.

Q: How do we add comments to a *daVinci* term representation file?
A: Inline comments are not allowed in the *daVinci* term representation. But you can write whatever you want at the end of a term representation, i.e. behind the last ']'. *daVinci* will simply stop reading the file as soon as the end of the graph term is found.

Q: Is it possible to load a graph in adjacency matrix format?
A: The term representation is the only graph format currently supported by *daVinci*, so it is not possible to load a graph in adjacency matrix format. A converter from matrix to term representation format is not extremely difficult (a great exercise for students :-), so maybe some *daVinci* user is interested to write such a tool. Please inform the *daVinci* project by e-mail in this case.

Q: How large can the string of a node or edge ID be, so *daVinci* can differ them?
A: There is no restriction in length for node or edge identifiers in the term representation graph format. But the longer the labels are, the slower the performance will be, because *daVinci* calculates a hash value for each ID and if two IDs have the same hash value, then each time IDs have to be compare character by character. Corollary: better choose short node or edge IDs in order to get better performance.

Q: Where can I find the documentation of the status file format?

A: Status files are only generated by *daVinci* itself. They contain the complete system state and can be loaded in a later session to restore a previous one. There is no documentation of the status file format, because no other program except *daVinci* is supposed to read these files. In fact, even the *daVinci* developers do not have a documentation at hand, because a status file is simply a dump of the complete internal data structure of *daVinci* that can be reloaded easily. Further, the status file format is changed with each *daVinci* release because of new features, so a backward compatibility mode is needed to read old files.

Application Programmer Interface (API):

Q: Can I connect *daVinci* and my program written in *XXX* by using the API??

A: *daVinci* has an application programmer interface (API) which is independent from a particular programming language. The communication between a connected application and *daVinci* is done via UNIX pipes. Refer to the API reference in the online documentation for details.

Q: Is it possible to construct graphs directly from a program written in *XXX*?

A: First you have to connect your application program to the *daVinci* API. This is described in the API reference. Then your application has to construct the graph in term representation format and send it to *daVinci* for visualization by using API-command graph(new(...)).

Q: I get selection answers after sending commands, but before receiving the 'ok'!

A: Yes, this is the way it should be. Some users are confused about the communication behaviour of the API. The *daVinci* developers would like to clarify that API communication between *daVinci* and an application is asynchronous, so both processes can send messages at any time. The absence of hand-shaking in the API protocol has the following consequence: generally, each API command is directly confirmed with an answer (mostly *ok*). After receiving an API command, *daVinci* does not send any other answers (e.g. selection events) until the command is completely executed and confirmed with the corresponding answer.

But this does not mean that for the application, the confirmation answer will directly follow after sending a command to the API. Answers for event notification may occur at any time, even if a command is not yet confirmed! The reason is that the event may have happened in the time interval after a command is send by the application, but before the command is received by *daVinci*. For *daVinci*, the event answer is then sent to the application before receiving the next command (which blocks sending of event answers). But for the application, the event answer is received from *daVinci* before getting the confirmation answer for the command.

Corollary: your *daVinci* application should expect to receive event answers at any time, even if a command, sent to the API, is not yet confirmed with the corresponding answer.

Graph Editor Application:

Q: Can I insert self edges with the graph editor?

A: Yes, starting with grapheditor V2.1 and *daVinci* V2.1 you can insert self edges. Please refer to the graph editor documentation for further informations.

Problems:

Q: I always get a core dump after starting *daVinci*. What is going wrong?

A: First of all, make sure that you are using the right binary for the operating system you currently use. For example, there are different binaries for Linux SuSe and Redhat, whereby the Redhat binary may crash on Suse and vice versa. You can test a binary with the standard *file* command to see if it is recognized by your platform. Test the *daVinci* binary and some other binary (where you are sure that it works) with *file* to see if both outputs match.
Otherwise you might have found a bug in *daVinci*. Please contact the developers by e-mail and describe your problem and your configuration in detail to get help.

Q: *daVinci* crashes with *"X Error: Bad alloc"*. What can I do?

A: Either you are working with extremely large graphs or your computer has not enough memory. The error comes from the X-client which was not able to allocate memory for a new pixmap. Try to reduce memory consumption by starting *daVinci* with option -nocache or set this option in the general settings dialog.

Q: I get an error *"Tk: can't read "tk_version": no such variable"* after starting *daVinci*.

A: You have probably started *daVinci* remote on another computer without setting the DISPLAY environment variable there. DISPLAY has to be set on the remote computer to point to the X-Window display where you want to see the output. For example if you are sitting in front of computer *bloodyslow* and want to start *daVinci* remote on server *fastaslightning* then you have to do the following procedure:

```
% xhost fastaslightning
fastaslightning being added to access control list
% rlogin fastaslightning
... (login messages from fastaslightning)
```

For csh users:

```
% setenv DISPLAY bloodyslow:0
% daVinci
```

For sh users:

```
% DISPLAY=bloodyslow:0; export DISPLAY
% daVinci
```

You can also use an option to let *daVinci* redirect the display:

```
% daVinci -display bloodyslow:0
```

Q: I cannot start *daVinci* V2.1 on SunOS 4. There are errors from ld.so!

A: Tcl/Tk has some specific problems with obsolete X Window releases prior than X11R5, so the loader complains about missing functions in the shared X11 library. The development of the BSD based SunOS was stopped by Sun in 1991, so the default in all SunOS 4.1.x releases is only X11R4. It is recommended to install X11R5 or X11R6 in order to use *daVinci*.

Q: I cannot start *daVinci* V2.0.x on DEC UNIX. There are errors from /sbin/loader!

A: Users of DEC UNIX (OSF/1) have reported problems to start *daVinci*. They've always got an error message *".../sbin/loader: Fatal Error: lazy_text_resolve: symbol cos should not have any relocation entry"*. In this case set environment variable LD_BIND_NOW to a non-null value:

For csh users:

```
setenv LD_BIND_NOW yes
```

For sh users:

```
LD_BIND_NOW=yes; export LD_BIND_NOW
```

With this setting *daVinci* should work as expected.

Q: Does *daVinci* have any problems with the year 2000?

A: No. As long as your operating system handles the year 2000 correctly, no date related malfunction will happen in all releases of *daVinci* after January 1, 2000. Look at the Year 2000 Information Center for more information about the millennium bug.

Help System / Online Documentation:

Q: *daVinci* doesn't talk with Netscape. What can I do?

A: If you press the Help button in a *daVinci* dialog window or use the Help/ menu and nothing happens in Netscape then check the Netscape command specified in menu General Settings dialog (which is usually 'netscape'). Locate this Netscape command (e.g. with 'which netscape') and look at that file. If it is a shell script installed by your system administrator, then make sure that the script passes the arguments to Netscape. This can be done in the following way:

```
#!/bin/sh
exec <filename_of_netscape_binary> $*
```

Otherwise *daVinci* cannot tell Netscape about the document requested by the user.

Q: What can I do if Netscape cannot find a file after starting the help system?

A: Open the General Settings dialog with menu Options/General Settings... and look at the content of field **Online Documentation:**. This specifies the URL-prefix which is passed to Netscape. It should point to the topmost directory of the *daVinci* online documentation, usually located in directory *$DAVINCIHOME/docs* (do not use shell variables such as *$DAVINCIHOME* in the dialog, Netscape does not understand this). Change the URL-prefix in the dialog to point to the online documentation, press Apply and save the options with menu Options/Save Options.
If this does not work, remove your options file with *rm ~/.daVinci* and start *daVinci* again.

Q: I need to print a paper version of the documentation. Is it available?

A: The HTML online documentation of *daVinci* V2.1 is written as a hypertext. For example terms are not defined at first occurance, they are hyperlinks to the place where the term is described in detail. So it is very difficult and inconvenient to read this text in a linear manner on paper. It is strongly recommended to read it online with a WWW browser such as Netscape Navigator.

More Information:

Q: Are any papers available about *daVinci*?

A: See [FM94b] and [FW95] in the *daVinci* bibliography. Both papers are available by ftp, the URL's can be found in the bibliography. You can also cite the *daVinci* online documentation which has URL *http://www.tzi.de/~davinci/docs/*

Q: Is there a mailing list?

A: Yes, you can subscribe or unsubscribe to the mailing list by sending an e-mail to the *daVinci* developers. At the first use *daVinci* will automatically ask the user to subscribe to the mailing list, so usually there is no need to do this manually.

Q: Where can I find a paper about the layout algorithm(s) used in *daVinci*?

A: The graph layout algorithms implemented in *daVinci* are described in the concepts document of the online documentation. The original papers where the algorithms are described are [STT81] (for graph layout) and [Juu94] (for tree layout), refer to the *daVinci* bibliography which is part of the online documentation. Note that both algorithms have been slightly improved in *daVinci*.

---

daVinci V2.1 Online Documentation - Page update: July 13, 1998

Overview | Intro | Reference | Tutorial | Questions & Answers | New Features | Index

Overview -> New Features in *daVinci* V2.1.x

daVinci - Online Documentation V2.1

New Features in *daVinci* V2.1.x

daVinci V2.1 is a major release with a lot of significant new features over the previous version of *daVinci*.

All new features, compared to previous release V2.0.x, are described below in the following categories (you can click on the links for quick access):

- **Fully incremental graph layout**
- **Subgraph abstractions using incremental layout**
- **Completely rewritten, much faster graph layout algorithm**
- **New automatic layout functions**
- **Fine-Tuning for all layout orientations**
- **Visualization rules (API)**
- **Pop-Up menus for nodes and edges (API)**
- **Drag and Drop for nodes and edges (API)**
- **All open windows accessible via API commands**
- **Some more enhancements to the API**
- **New command-line options**
- **New Attributes for Nodes and Edges**
- **grapheditor supports all new API features, including drag and drop**
- **Several bug fixes and reduced memory footprint**
- **Shortcomings**

Fully incremental graph layout

Graph updates send to the *daVinci* API using command graph(update(...)) are now inserted into the existing graph layout incrementally instead of forcing a completely new layout. So the layout before the update is preserved as much as possible in order to support layout stability. Of course, after doing some incremental updates, the graph layout may be getting worse, so a new global graph layout might be required. The user can do this at any time with menu Layout/Improve All. Existing *daVinci* applications don't need to be modified to support incremental graph layout. Applications already using API command graph(update(...)) immediately benefit from this new feature. For example, you can connect the graph editor of *daVinci* V2.0.x to the *daVinci* V2.1.x API and it will use incremental graph layout. But there even is a new grapheditor, which supports much more of the new features.

Subgraph abstractions using incremental layout

Abstractions are used in *daVinci* with menu Layout/{Hide|Show} Subgraph. In former versions, showing a hidden subgraph after modifying the layout (e.g. by using menu Layout/Improve All with hidden subgraphs) messed up the graph layout completely. Now, menu Layout/Show Subgraph always uses incremental graph layout for fading in hidden subgraphs in order to preserve layout stability.

Completely rewritten, much faster graph layout algorithm

This algorithm is used to calculate the order of nodes and dummy nodes to reduce the number of edge crossings. This algorithm has been completely rewritten in V2.1 in order to:

- get **much** more performance. For example: When applied to a test graph with about 1000 nodes and 1500 edges, menu function Layout/Improve All is about 40 times faster in *daVinci* V2.1 than in V2.0.3.
- calculate a layout with fewer edge crossings. A new optimization phase gives slightly better results than in earlier releases.

New automatic layout functions

By taking advantage of layout constraints available with the rewritten graph layout algorithm, a new layout function has been implemented in menu Layout/Improve Edges. In opposite to menu Layout/Improve All which rearranges the nodes and edges to get fewer edge crossings, the new layout function re-routes only the edges, but preserves the given order of nodes at each level. This reduces not all of the unneccessary edge crossings, but the order of nodes is preserved to keep the layout more stable.

With the new menu Layout/Improve Selected Nodes you can rearrange only the selected nodes and the edges (as in "Improve Edges") without changing the order of all other nodes. The menu "Improve Visible" has been removed.

Fine-Tuning for all layout orientations

In *daVinci* V2.0.x you can only fine tune your graph in the layout orientation "Top Down". Now it is possible to do this in all layout orientations. On 24-bit graphic machines the moved node is now displayed correctly, instead of a black rectangle in V2.0.x. Additionally, the graph size is no longer fixed in fine-tuning. You might have noticed, that nodes can not be moved out of the surrounding rectangle of the graph in V2.0.x. Now it is possible to enlarge the graph by moving nodes.

Visualization rules (API)

daVinci V2.1 has a new API mechanism to control and maintain the visual appearance of graph nodes and edges more easily. In former versions it was neccessary to specify the visual appearance of each individual node or edge by using attributes in the term representation of a graph. By using visualization rules in V2.1, one can specify the attributes for a type (or class) of nodes/edges at one single point. Suppose you have a graph with four nodes: one root node 1 and three childs 2,3,4. The root has type T1, the childs have type T2 (types are the strings between the node/edge ID and the attributes in a term representation or graph update). You can send this example graph to the *daVinci* API using the following command (remove line breaks before sending commands to API):

```
graph(update([new_node ("1","T1",[a("OBJECT","1")]),
              new_node ("2","T2",[a("OBJECT","2")]),
              new_node ("3","T2",[a("OBJECT","3")]),
              new_node ("4","T2",[a("OBJECT","4")])],
             [new_edge ("1-2","T1",[],"1","2"),
              new_edge ("1-3","T2",[],"1","3"),
              new_edge ("1-4","T3",[],"1","4")]))
```

Now you can send a visualization rule to the API to draw *all* nodes of type T2 in red and another one to draw *all* edges of type T2 in blue:

```
visual(new_rules([nr("T2",[a("COLOR","red")])]))
visual(new_rules([er("T2",[a("EDGECOLOR","blue")])]))
```

You can even overwrite the *daVinci* defaults for attributes with visualization definitions. Please refer to the concepts document and to the API command category Visual to read more about visualization definitions.

Pop-Up menus for nodes and edges (API)

Graph nodes and edges may now have a pop-up menu which is displayed if the mouse pointer is currently over a particular node or edge and the user selects the right mouse button. For comfortable use, pop-up menus should be specified with visualization rules for particular types of nodes and edges (see above) and are implemented with a new kind of node/edge attribute.

For example: The following API command sends a graph to *daVinci* that contains one node with a pop-up menu (remove line breaks before sending commands to API):

```
graph(new([l("1",n("T",[a("OBJECT","Node 1"),
                        m([menu_entry("ID","Entry")])],[]))]))
```

Please refer to the description of the term representation to read more about popup menus.

Drag and Drop for nodes and edges (API)

For a better support of user interaction we have added the drag and drop functionality to the *daVinci* API. With the new commands you can set the API into dragging mode. This switches off the two dimensional scrolling feature, you achieve normally by pressing the middle mouse button. Then you can click with the middle mouse button somewhere in the *daVinci* window and the API informs the application, or you can select a node with the middle mouse button and drag a line from this node to another node. The application is informed if you release the button over a node or over the free area. Moreover you can grab a node and drop it on some other node.

Please refer to the concepts document and to the API command category Drag and Drop to read more about drag and drop.

All open windows accessible via API commands

Many people asked for full access to all open windows in the multi view mode of the API. Implementing this feature results in a syntax change for the API answers. Therefore we make this mode available via a command line option -window api. You'll get context_window(...) answers instead of context(...) answers in the new mode.

Please refer to the API command category Multi to read more about this new multi-window mode.

Some more enhancements to the API

- The graph(update(...)) command and the graph(update_and_change_attr(...)) now accept updates for nodes and edges in one list.
- The new command graph(change_type(...)) can be used to change the type (or class) of nodes and edges.
- The command "menu(navigation(select_childs(...)))" has been renamed to menu(navigation(select_children(...))) but the old command is still available for backward compatibility.

New command-line options

With the command line option -init <file> you can supply initial API commands to *daVinci*, independent whether you've connected an application or not. You can execute all API commands except those of categories multi, drag_and_drop and app_menu, which would make no sense, if no application is connected. With this feature, you can send for example the new visualiziation rules to *daVinci* or load graphs in the API update format to *daVinci*.

With the command line option -log <file> you can supply a file, in which all commands and answers of the API send to or send by *daVinci* are logged.

With the command line option -version *daVinci* will print it's current version and terminate.

New Attributes for Nodes and Edges

Some attributes for nodes and edges have been added or changed. For nodes:

- _GO: new graphical object 'triangle' and 'scissors'
- BORDER: this attribute can be set to 'none', 'double' and 'single', the first value is new and especially usefull for _GO icon.
- _CGO, CCOLOR, CICONFILE are new. See below for an explanation of these attributes.

For edges:

- HEAD: With this new attribute you can specify the head of an edge, which can only be a solid arrow in V2.0.x and former versions. Now the following values are allowed:
  - farrow: default behaviour as before (filled arrow)
  - arrow: arrow only drawn by its border lines
  - fcircle: filled circle
  - circle: normal circle not filled

Up to now the visualization of nodes abstracted by the menu or API command Hide Subgraph was a scissors-icon with the color of the node. Now you can define the appearance of these nodes by setting the attributes:

- _CGO: graphical object for abstracted nodes like _GO, default: scissors
- CCOLOR: color of abstracted nodes, default: value of COLOR for the node
- CICONFILE: icon file, if _CGO is 'icon'

Please remember, that it could be very difficult to find out which nodes are abstracted, if you make a bad choice for these attributes.

grapheditor supports all new API features, including drag and drop

The new grapheditor supports all the new features of the API and all new node and edge attributes (except those for abstracted nodes). The major improvement is the drag and drop mode, which is the default mode of the grapheditor now. In this mode, you can create new nodes by clicking in the *daVinci* window with the middle mouse button and new edges by dragging the mouse with pressed middle mouse button from one node to the other. You can even copy existing nodes. The drag and drop mode can be switched off via the icon bar or a menu. Additionally there are context sensitive pop-up menues for all nodes and edges generated with the grapheditor. The grapheditor now uses the incremental layout of *daVinci* and you'll definitely see the changes.

Additionally the graph editor now supports the editing of several graphs and the loading of status files.

Please refer to the grapheditor manual to read more about the new grapheditor features.

Several bug fixes and reduced memory footprint

- Correct navigation in cyclic graphs. The selection functions of menu Navigation now work as one might expect from the layout: A child is the node where the arrow of an edge is pointing to and a parent is the node where the arrow is not pointing to. There are situations where a node is both a parent and a child.
- The layout algorithm and the drawing component have been rewritten, to improve performance (see above) and reduce the memory usage of *daVinci*. Especially the resources allocated in the X-server are reduced by 50 percent, because the pixmap caching mechanism of *daVinci* now needs only one pixmap instead of two for each part of the graph.
- All Bugs which hadn't been fixed in V2.0.3 are now fixed.

Shortcomings in daVinci *V2.1.x*

The following shortcomings are present in the current release:

- Visualization definitions can be defined only via the API.
- Edge labels are still not supported. There is a work-around for edge- labels by using graphical object *text*.

---

daVinci V2.1 Online Documentation - Page update: June 15, 1998

Overview -> Reference

daVinci - Online Documentation V2.1

Reference

This chapter covers detailed background information about the *daVinci* graph format (term representation), application programmer interface (API), PostScript interface for printing graphs, the graph editor, tools bundled with the distribution, command-line options, the bibliography, and an index. The following subtitles are available:

- **Term Representation for Graphs**
- **API Reference**
- **PostScript Interface**
- **Graph Editor Application**
- ***daVinci* Tools**
- **Command-Line Options and Environment Variables**
- **Bibliography**
- **Index**

daVinci V2.1 Online Documentation - Page update: June 15, 1998

daVinci Term Representation

daVinci graphs are given in a format called *term representation* which is composed of plain text ASCII characters. Graphs can be created very easily, even with a usual text editor. But in most cases graphs are automatically generated by an application program, for example with the interactive graph editor application which is part of each *daVinci* distribution. All kinds of directed graphs can be specified with a term representation: acyclic and cyclic structures or even graphs with multi-edges (i.e. more than one edge between two nodes) and self-edges (i.e. edges where the parent and child node are the same). The term representation of a graph is either loaded from file with menu File/Open... or sent to the API.

Structure of Term Representation

In general, a term is a structure where a superterm (*parent*) encloses its subterms (*children*), e.g. *parent[child1,child2,child3]*. Brackets *[...]* are used to get a list of comma-separated elements of the same type. This scheme of expressing parent-child relationships can be applied recursively, so each child may have its own children, and so on. Such a notation allows to represent arbitrary tree structures.

To specify graphs, a mechanism of *identifiers* (called *labels* in former releases) and *references* is used in *daVinci*. For example, if a child node has more than one parent node, then in the term representation the corresponding subgraph of the child appears in only one of the parents as a subterm. This subterm is marked with an identifier (in fact, all nodes and edges need to be marked with a unique identifier). All the other parents of the same child do not duplicate the subterm. Instead, they point to the child by using a reference to the identifier. Note that this way it is even possible to describe cyclic graphs. When loading a term representation, *daVinci* will construct an internal graph by resolving these references. The linear order of a node's subterm (where the identifier is declared) and a reference to this node (where the identifier is used) is arbitrary in a term representation, so references can be used before the corresponding identifier and subterm appears in the term representation.

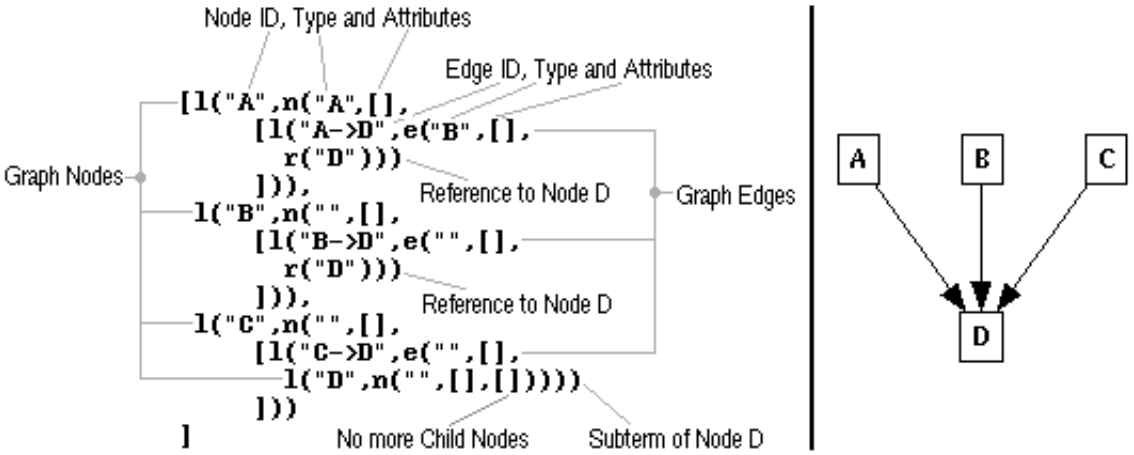
Note: You have to make sure that each node and edge in a term representation has its own unique identifier, i.e. do not use the same ID for two objects. In multi-graph mode, each context (graph) has its own scope, so there is no need to have unique identifiers for the union of all current contexts.

Beside the identifier (a unique string) and the list of child nodes (which is probably empty), each node also has a *type* and a list of *attributes* which are responsible for the image of a node in the visualization. Types and attributes are discussed below. Between a parent and the corresponding child node, there is an edge in the term representation which also has its unique edge identifier, type and attributes. So, in fact the children of a node are edges and each edge has one node or reference as subterm.

Note: In previous V1.x releases of *daVinci*, identifiers (*labels*) for nodes and edges were recommended, but optional. In *daVinci* V2.x, identifiers are now mandatory to support several new features. For example, identifiers will be sent back to a connected application when the user selects

nodes or edges.

The syntax of the term representation (startsymbol is *graphterm*) can be found in the API definition. When a graph is loaded from file, *whitespace* (blanks, tabs or returns) may occur at any position in a term representation. Of course, in strings whitespace is significant. When a graph is sent in term representations to the API, whitespace should be avoided and returns are forbidden. The following figure illustrates a term representation of a small graph and the corresponding visualization in *daVinci*. A more complex example is demonstrated below.



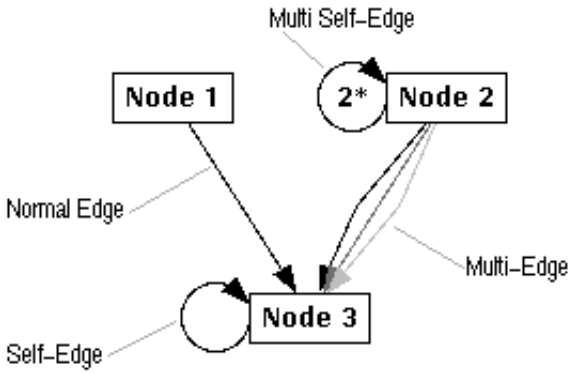
Term Representation of a Graph and its Visualization

Multi-Edges and Self-Edges

Multi-edges and *self-edges* are special kinds of edges that are treated in a different way in the graph layout. Multi-edges are present when there are two or more edges between the same parent and child node. All these edges form one *multi-edge*. The single edges of a multi-edge are spreaded in the middle between the parent and the child (see figure below), whereby the used gap can be adjusted with the Options/Layout Dimensions... dialog window.

Self-edges are edges where the parent and child node are the same, i.e. a node that has an edge pointing to itself. Self-edges are visualized with a circular edge at the left resp. lower side of the node. The user can adjust the radius of self-edges with the Options/Layout Dimensions... dialog window. *Multi self-edges* are also possible when a node uses edges with a reference to itself many times in its list of children. These edges are visualized by writing the number of self-edges in the middle of the circle. The attributes of self-edges are considered for the graph visualization (only in *daVinci* V2.0.2 or higher), but it is undefined which self-edge of a multi self-edge is taken for the attributes. So for example, if you have a multi self-edge containing a red and a blue self-edge, then it is not defined whether the multi self-edge is drawn in red or blue.

The following figure illustrates the visualization of multi- and self-edges in *daVinci*.



Node and Edge Types

In a term representation, all nodes and edges have a *type* which is an arbitrary string. The type assigns nodes and edges to *classes*. It is the intention to define the visualization of all classes in a central file called *visualization definition*. Node and Edge Types are first used in *daVinci* V2.1, but the visualization of the classes can only be defined via API commands of category Visual. For example, it is possible to specify a rule to draw all nodes of some class *X* with a red circle, instead of using attributes for each individual node of the class. The modification of a rule will affect all elements of the class at the same time. So, the central visualization definition is a comfortable method to define a universal visualization for a family of graphs.

Note: With the command-line option -init you can supply API commands at startup of *daVinci*. If you put your definitions into such an initial file, you will be able to use visualization defintions without the API. Unfortunately, you will not be able to change the definitions at runtime.

In one of the next *daVinci* releases, the user will be able to load a file of rules (*visualization definition*) that defines the visual appearance of a class.

Attributes

Attributes are used in the term representation to specify the visualization of individual nodes and edges. Each node and edge has its own attributes which is a list of string pairs with a constructor **a** followed by an attribute name and attribute value in parentheses or a constructor **m** followed by a list of menus. Here is an example for the attributes of a node:

```
[a("OBJECT","hello"),a("COLOR","red"),a("FONTFAMILY","times"),
 m([menu_entry("MenuID","Menu Entry")])
]
```

If a particular attribute is specified for a node or edge, it will overwrite the default value for this attribute which is used otherwise. Please look at the paragraph visualization defintion of the concepts document to find out more about the evaluation order for attributes. Node and edge attributes are distinguished from eachother. For example, there are different attributes (more precisely: attribute names) to set the color of nodes ("COLOR") and edges ("EDGECOLOR"). Edge attributes cannot be used for nodes and vice versa. Unknown attribute names and values are simply ignored without any warning. This way, the term representation is both up- and backward compatible with previous and future releases of *daVinci*. Some attributes, starting with character '_', are needed for internal purposes, but can also be used in a term representation. The usage of attributes is demonstrated in the

example below.

Supported Node Attributes

Node Attribute "OBJECT"

This is the most important attribute for a graph visualization, used to specify the text that appears inside a node. Usually, each node should have this attribute. The text, i.e. the attribute value, may be an arbitrary string, whereby international characters are supported (ISO 8bit). The extension of a node in the graph visualization will automatically follow the size of the text by considering the current font, so nodes may have any size. Even multi-line text is supported by using a '\n' inside the attribute value. The **default** value for this attribute is the empty string "".

Node Attribute "FONTFAMILY" and "FONTSTYLE"

These two attributes are used to visualize the text of a node in a different font. Supported values of attribute "FONTFAMILY" are: "lucida" (**default**), "times", "helvetica", and "courier". Values of attribute "FONTSTYLE" are: "normal", "bold" (**default**), "italic", and "bold_italic". For a node, both font attributes do not need to be used pairwise at the same time.

Node Attribute "COLOR"

Can be used to define the background color of a node. The value of this attribute may be any X-Window colorname (see file *lib/rgb.txt* in your X11 directory) or any RGB color specification in a format like "#0f331e", where 0f is the hexadecimal value for the red part of the color, 33 is the green part and 1e is the blue. Hence, a pallet of 16.7 million colors is supported. The **default** color for nodes is "white".

Node Attribute "CCOLOR"

Same as attribute "COLOR" but this color is used, if the subgraph of the node is hidden. The **default** value for this attribute is the value of the attribute "COLOR" or its default value.

Node Attribute "_GO" (Graphical Object)

This attribute specifies the geometrical object used to draw a node. Nine different objects are currently implemented: "box" (**default**), "circle", "ellipse", "rhombus", "triangle", "text", "invisible", "icon" and "scissors".

Graphical object "text" can be used to simulate edge labels which are currently not available in *daVinci*. "text" is very similar to object "box", in fact it is a box without the black border. An example graph with simulated edge labels is file *example_graphs/edge_labels.daVinci* of the *daVinci* distribution. The trick with simulated edge lables is to use a regular node with graphical object "text" to show an edge label.

With object "icon", a user can define to use a monochrome image for a node in the visualization. Unfortunately, "icon" is more complicated than the other graphical objects. The bitmap file with the image of the icon need to be specified with a second attribute "ICONFILE" (see below), so for icons you need two attributes: "_GO" *and* "ICONFILE".

Note: The geometrical object "scissors" is normally used to display nodes with hidden subgraphs. So it is not recommended to use this object, because the user may get confused.

Node Attribute "_CGO" (Graphical Object)

Same as attribute "_GO" but this geometrical object is used, if the subgraph of the node is hidden. The **default** value for this attribute is "scissors". You also need the attribute "CICONFILE", if you choose the geometrical object "icon" as the attribute value, as described for "_GO".

Node Attribute "ICONFILE"

This is used as a supplement for attribute "_GO" when its value is set to "icon". Attribute "ICONFILE" specifies the (relative) filename of the bitmap used to draw the node as icon. The file must be in X-Window bitmap format (*.xbm*). Absolute filenames are not allowed as a value for this attribute. *daVinci* is looking for the file in all directories given by environment variable *$DAVINCI_ICONDIR*. So for example, when the bitmap of an icon is in file */home/me/images/my_image.xbm*, then add */home/me/images* to the search path of environement variable *$DAVINCI_ICONDIR* before starting *daVinci*, set attribute "_GO" to "icon" and attribute "ICONFILE" to "my_image.xbm" for a particular node. The **default** value for this attribute is "".

Notes about attribute "ICONFILE":

- Using attribute "ICONFILE" without setting "_GO" to "icon" will have no effect. Do not forget to set both attributes when you want to have icons in a graph visualization.
- Icon images are not saved in a status file, so when you like to transfer the status file of a graph with icons, make sure to transfer the bitmap files, too!
- If *daVinci* does not find the specified bitmap file of an icon in the directories defined by environment variable *$DAVINCI_ICONDIR*, then a default image with a question mark will be used instead.

Node Attribute "CICONFILE"

Same as attribute "ICONFILE" but this file is used, if the subgraph of the node is hidden. The **default** value for this attribute is "". This is used as a supplement for attribute "_CGO" when its value is set to "icon".

Node Attribute "HIDDEN"

Attribute "HIDDEN" is used to hide the subgraph of a node initially. This is the same operation available in menu Abstraction/Hide Subgraph. Supported values for this attribute are "true" and "false" ("false" is the **default**).

Node Attribute "BORDER"

This attribute defines the border of the geometrical object (see attribute "_GO"). The values of this attribute may be: "none", "single" (**default**) and "double". The following geometrical objects support all values: "box", "circle", "ellipse" "rhombus", and "triangle". Objects "icon" and "scissors" can't have "double" borders and objects "text" and "invisible" have no border at all. The **default** value is "single" to have a normal border.

Node Menu Attribute

This attribute defines the popup menu of a node. This is done with a different syntax than for all the other node attributes. Instead of *a(...,...)*, menus are defined with *m(menus)*. The popup menu is displayed, if the user presses button BMenu over the node. This attribute is only usefull, if an application is connected. Please have a look at the description in the command category Application

Menu for a further description of the parameter *menus*.

Supported Edge Attributes

Edge Attribute "EDGECOLOR"

In the same way as nodes, the color of an edge can be specified using this attribute. The value of this attribute may be any X-Window colorname (see file *lib/rgb.txt* in your X11 directory) or any RGB color specification in a format like "#0f331e", where 0f is the hexadecimal value for the red part of the color, 33 is the green part and 1e is the blue. Hence, a pallet of 16.7 million colors is supported. The **default** color for edges is "black".

Edge Attribute "EDGEPATTERN"

The pattern style of an edge is defined by this attribute. Possible values are: "solid" (**default**), "dotted", "dashed", "thick", and "double". Note: Thick edges should be used carefully because users might confuse with selected solid edges (both are drawn with the same line thickness!).

Edge Attribute "_DIR"

This attribute is used to control the arrow of an edge. In a graph visualization, each edge usually has an arrow pointing to the child node. This attribute can be used to let the arrow be drawn inverse (i.e. pointing to the parent), to get an arrow at both sides of an edge or to suppress arrows for a particular edge. The supported attribute values are: "normal" (**default**), "inverse", "both", and "none".

Edge Attribute "HEAD"

daVinci will draw a little image at no, one or both ends of an edge. This behaviour is controlled with the attribute "_DIR". With the attribute "HEAD" you can define which image this should be. Normally it's an arrow, therefore we talk about arrows in the rest of the documentation. The possible values are: "farrow" (**default**), "arrow", "fcircle", and "circle", where a leading 'f' means filled.

Edge Menu Attribute

This attribute defines the popup menu of an edge. This is done with a different syntax than for all the other edge attributes. Instead of *a(...,...)*, menus are defined with *m(menus)*. The popup menu is displayed, if the user presses button BMenu over the edge. This attribute is only usefull, if an application is connected. Please have a look at the description in the command category Application Menu for a further description of the parameter *menus*.

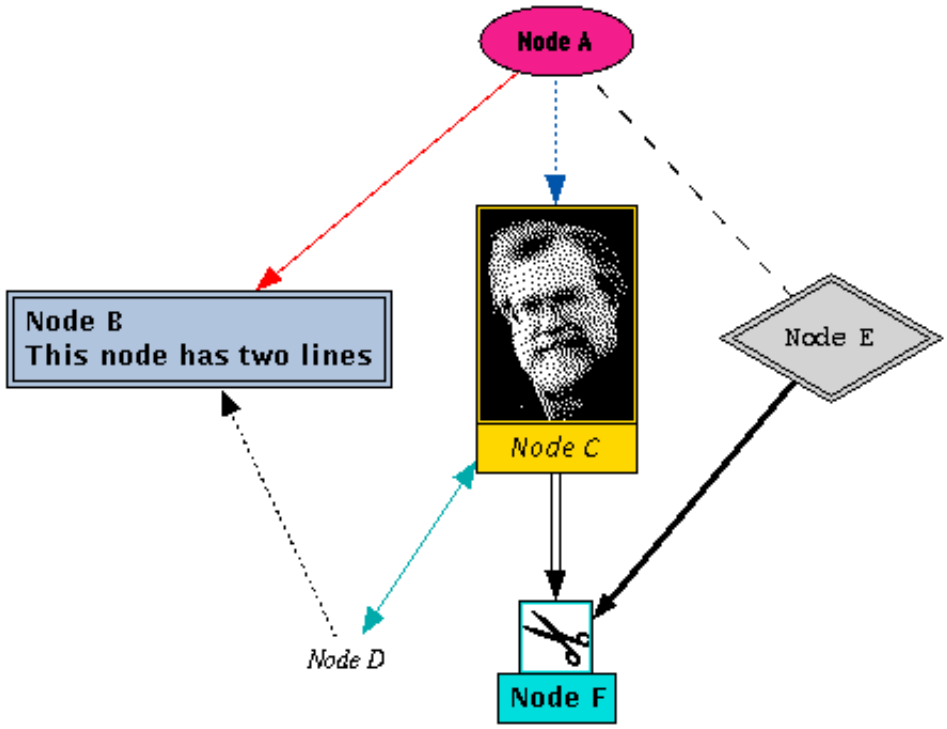
Example of a Term Representation

This example demonstrates a complex term representation of a graph, where nearly all available node and edge attributes are used. The figure below shows the resulting graph visualization in *daVinci* V2.1.x.

```
[
l("Node A",n("Module",[a("COLOR","#f31d8b"),
                       a("OBJECT","Node A"),
                       a("_GO","ellipse"),
                       a("FONTFAMILY","helvetica")],
   [
   l("Edge A->B",e("",[a("EDGECOLOR","red")],
    l("Node B",n("anything",[a("COLOR","light steel blue"),
```

```
                            a("BORDER","double"),
                            a("OBJECT","Node B\nThis node has two lines")],
        [
        l("Edge B->D",e("",[a("EDGEPATTERN","dotted"),
                            a("_DIR","inverse")],
         r("Node D")))
        ])))),
    l("Edge A->C",e("",[a("EDGEPATTERN","dotted"),
                        a("EDGECOLOR","#0054aa")],
     l("Node C",n("anything",[a("_GO","icon"),
                            a("ICONFILE","bkb.xbm"),
                            a("COLOR","gold"),
                            a("OBJECT","Node C"),
                            a("FONTSTYLE","italic")],
        [
        l("Edge C->D",e("anything",[a("EDGECOLOR","#00aaaa"),
                                    a("_DIR","both")],
         l("Node D",n("Module",[a("OBJECT","Node D"),
                                a("_GO","text"),
                                a("FONTFAMILY","times"),
                                a("FONTSTYLE","italic")],[])))),
        l("Edge C->F",e("anything",[a("EDGEPATTERN","double")],
         r("Node F")))
        ])))),
    l("Edge A->E",e("",[a("EDGEPATTERN","dashed"),
                        a("_DIR","none")],
     l("Node E",n("anything",[a("COLOR","LightGray"),
                            a("FONTFAMILY","courier"),
                            a("FONTSTYLE","normal"),
                            a("BORDER","double"),
                            a("OBJECT","Node E"),
                            a("_GO","rhombus")],
        [
        l("Edge E->F",e("anything",[a("EDGEPATTERN","thick")],
         l("Node F",n("anything",[a("COLOR","#00dddd"),
                                  a("OBJECT","Node F"),
                                  a("HIDDEN","true")],
            [
            l("Edge F->G",e("anything",[],
             l("Node G",n("anything",[a("OBJECT","Node G")],[])))))
            ])))))
    ]))
]
```

Here is the visualization of this term representation:



---

daVinci V2.1 Online Documentation - Page update: June 15, 1998

daVinci API

The basic concept of *daVinci* is to provide a generic graph user interface that can be used on top of any application program. The application is able to access all operations of *daVinci* by using the API (Application Programmer Interface). So, the programmer of an application who might want to have a powerful graphical user interface to display the applications graph data structures does not need to re-invent the wheel. Instead, the application can simply plug *daVinci* to get state-of-the-art graph visualization.

Note: The syntax of the API has been completely revised with transition from *daVinci* V1.x to V2.x. An automatic backward compatibility mode supports all commands of the former V1.x API. Programmers of applications written for *daVinci* V1.x should read the Transition Guide. Some command names have changed from *daVinci* V2.0.x to *daVinci* V2.1.x, but all commands available in *daVinci* V2.0.x are still available for backward compatibility. This means your *daVinci* V2.0.x application should still work without any changes.

The following subtitles are available in this chapter:

- **Getting started: Introduction to the API**
- **How to connect an Application**
- **Programmers Guide for API Users**
- **Transition Guide for *daVinci* V1.x Applications**

- **Overview: API Commands**
  - **Commands of Category Graph**
  - **Commands of Category Multi**
  - **Commands of Category Menu**
  - **Commands of Category Application Menu**
  - **Commands of Category Set**
  - **Commands of Category Window**
  - **Commands of Category Tcl**
  - **Commands of Category Special**
  - **Commands of Category Visual**
  - **Commands of Category Drag and Drop**
- **Overview: API Answers**
  - **All Answers**

daVinci API - Introduction

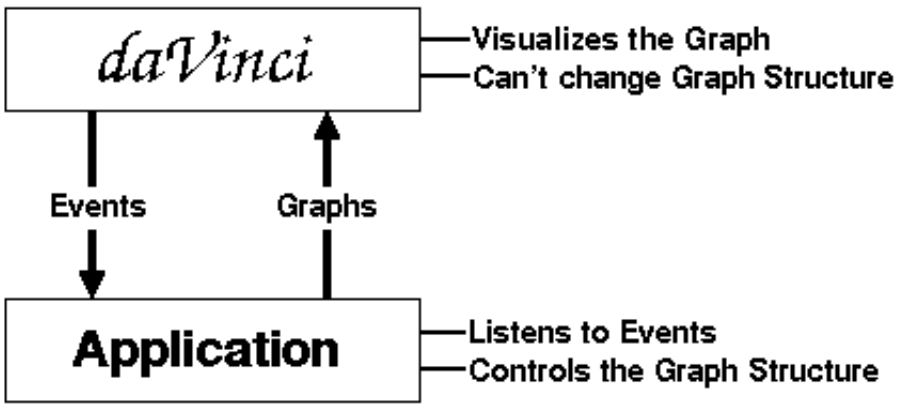
After using *daVinci* for a while, you might wish to have a similar user interface for the application program you have developed. Until now, you have been afraid of the cost of such an expensive implementation. But imagine how the user will be able to handle your complicated graph data structure by easily browsing an elegant and intuitive graph drawing of high quality. How about controlling your application by simply clicking on nodes and edges in a graph visualization? This is the typical szenario where you want to start using the API to plug *daVinci* on top of your application.

daVinci does not control the structure of a displayed graph. It is a pure visualization system and is even not able to alter the graph structure on its own. So unlike usual graph editors (which may take control on the development of a graph by removing or adding nodes and edges without any notion about the specific underlying semantic), *daVinci* let your application decide what operations need to be applied when nodes and edges are clicked or menus and icons are selected.

The basic philosophy of *daVinci* is to have two levels of responsibility:

1. The visualization layer (*daVinci*), responsible for graphical representation, but without permission to manipulate the graph structure on its own.
2. The application layer (some application program connected to the API) which decides what to do if an operation should be applied that modifies the structure of a graph.

The following figure illustrates the data flow between an application program and *daVinci*:



An application is connected with *daVinci's* API by using two communication channels: One to send commands (e.g. graphs) and one to receive the answers (e.g. events). With API commands, the application is able to send graphs for visualization and attach its own dialog elements (i.e. menus and icons) to the *daVinci* user interface. Afterwards, the application will wait for answers from *daVinci*. These are events triggered by user interaction in *daVinci's* graph visualization windows. Based on

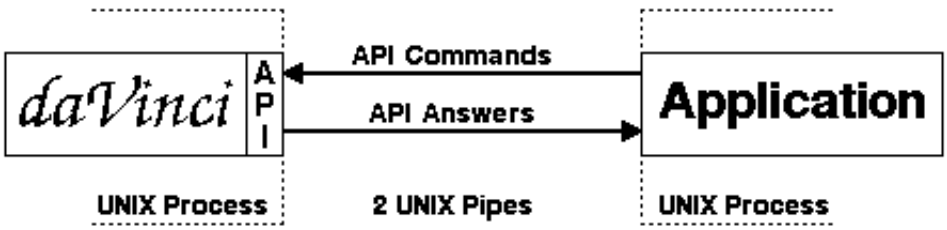
daVinci API - Introduction

these event, for example a particular menu selection in the context of a previous node selection, the application can decide what operations need to be applied on its own local graph data structure, for example deleting the specific node if the semantic of the selected menu is "removal". After modifying its own data structure, the application has to sent the new graph (or an increment) to *daVinci* to update the graph visualization.

---

daVinci API - How to Connect an Application

From a technical point of view, communication between *daVinci* and a connected application program is realized with two UNIX pipes: One to send commands to the *daVinci* API (e.g. to transmit graphs, to attach menus/icons or to call menu functions) and one to receive the answers (e.g. return values, user events). The following figure illustrates this approach:



Only one application program can be connected with *daVinci* at a time, but this process is able to visualize many graphs in different windows (see multi-graph mode). If there is a need to have many applications communicating with *daVinci* in a system architecture, then the application programmer has to provide a single "concentrator process" which is between *daVinci* and all the applications to synchronize communication.

Choose which way to go

First of all, there are two opportunities to connect an application with *daVinci's* API. Either the application let *daVinci* do this job (this is the most easiest way) or the application connects *daVinci* to get more flexibility. The point is whether *daVinci* or the application is the parent process of the two. The advantage of the first solution, where *daVinci* is responsible for establishing the connection, is that the application programmer can leave the UNIX process- and pipe programming manuals on the book shelf. In this case, all an application has to do is sending commands to the API by printing on *stdout* and receiving the answers from the API by reading from *stdin*. Further, if *daVinci* launches the application process, the user will be able to disconnect the application later in *daVinci* to connect another one.

The disadvantage of this solution is that the application's *stdin* and *stdout* are exclusively reserved for communication with *daVinci*. If more flexibility is needed, e.g. to use other file descriptors than *stdin* and *stdout*, the application has to establish the pipe communication with *daVinci* on its own, read below for details.

Either daVinci *connects the Application...*

daVinci is able to connect an application program at start-up time using command-line option -startappl or at run-time with menu File/Connect Application.... By connecting an application, *daVinci* forks a new UNIX process, creates two pipes for communication, binds *stdin* and *stdout* of the new process to the end of the pipes and finally starts the specified application which has to wait for an initial ok answer before start sending commands to the API by simply printing them on *stdout*. Answers from the API are available for the application by reading from *stdin*.

At any time, the user is able to shut-down the currently connected application by selecting menu File/Disconnect Application. But this only works in case of support by the application (it has to terminate after receiving answer disconnect).

... or the Application connects daVinci

When the more comfortable connection method (described above) is not appropriate, the application program needs to start *daVinci* on its own. To do so, the application has to perform the following tasks:

1. Forking a new UNIX process. This is the child process for *daVinci*.
2. Creating two UNIX pipes, one to send commands and the other to receive answers from the API.
3. Redirecting the file descriptors: *stdin* and *stdout* of the child process (*daVinci*) has to be connected with one side of both pipes. The parent process (application) has to connect two free file descriptors to the other end of both pipes.
4. Calling *daVinci* with command-line option *-pipe* in the forked child process.

Note: The application has to set line-buffered I/O before start communication with *daVinci* via pipes. For example, in C this can be done with *setbuf(stdout,NULL)* and *setbuf(stdin,NULL)*.

After establishing connection, *daVinci* sends an initial ok answer for synchronization. The application should not start sending commands to the API before receiving this answer. *daVinci* is not able to disconnect the application if it has not established the connection before. So, menu File/Disconnect Application is always deactivated for the connection method described in this paragraph.

Transmitting Commands

After establishing connection between your application and *daVinci*, you should test the communication by sending command *nothing* several times to the API. The connection works if you receive an answer *ok* for each sent command. *daVinci* only accepts valid commands according to the API syntax, all other messages are treated as a syntax error by sending answer communication error(...) back to the application. Unlike previous V1.x releases, *daVinci* V2.x has now a robust API to continue communication after a syntax error is detected. Tip: You can play with the API by starting *daVinci* in a shell with option *-pipe*. In this case, you are able to enter (or better: using copy/paste) commands into the shell and send them to the API with one final return behind the command. The answers from the API are also displayed in the shell in this case. You can find some API examples in your *daVinci* distribution.

Note: API commands **must not have** newline or return characters inside. They **must have** exactly one newline at the end of the command to flush the write buffer. No other characters, even no spaces, must follow this concluding return, except of the next complete command. If these restrictions are not considered, pipe communication with the *daVinci* API will likely fail.

---

daVinci V2.1 Online Documentation - Page update: June 15, 1998

daVinci API - Programmers Guide

This document is an overview of the API for programmers who want to connect their application with *daVinci*.

Commands of the API

After connecting your application program with *daVinci* (section "How to connect an Application" explains this topic in detail), you can start sending commands to the API. The "Overview of Commands" is a useful reference to browse the command syntax. The commands are divided into ten categories with respect to their semantic. They are build hierarchically with parenthesis according to the category they belong to, e.g. command *menu(file(exit))*. The following categories are available:

- **graph(...)**
  The graph category covers commands to send and update graphs. You can send a complete graph in term representation format or update the structure of an already loaded graph by adding and deleting nodes and edges. Further, it is possible to change the attributes of nodes and edges (more information about attributes) and to change the type of nodes and edges (more information about types). Graph updates and attribute modifications can be combined to be performed in one step.

- **multi(...)**
  The multi-graph category covers commands to work with many graphs in different windows at the same time. Commands are available to open a new context (i.e. to get a new empty window, where an additional graph can be visualized). Each context (graph) has its own unique context-ID. In multi-graph mode, *daVinci* has a current context which can be set with a command of this category. The current context specifies the graph that is currently affected by all the other commands sent to the API. On the other side, in multi-graph mode, each answer from the API has a preluding context answer to specify the graph where the event happened. Multi-graph mode is entered as soon as the first context is opened. If you intend to use more than one graph later in a session, you should open the first context at initialization time.

  With this concept you can only access each context (graph) on its own, but you have no access to the windows opened in a context. Especially the commands of category Window are sometimes useless, because they work only for the first window of a context. Therefore we have introduced the Window-API mode in *daVinci* V2.1. With the command-line option -window_api, you can switch the behaviour of the API. We have to keep the old behaviour as the default, because we have to guarantee, that all old applications still work without changes. In multi-window mode, the current context must be set with a command of this category. The current context and window specifies the graph and its window that is currently affected by all the other commands sent to the API. On the other side, in multi-window mode, each answer from the API has a preluding context_window answer to specify the graph and window where the event happened.

- **menu(...)**
  The menu category covers commands to call the menu functions of *daVinci's* user interface. All these commands are further divided into sub-categories with respect to the menu they belong to: File, View, Navigation, Abstraction, and Layout.

- **app_menu(...)**
  The application menu category covers commands to let the application attach its own menus and icons to the *daVinci* user interface. Application menus are always attached to the Edit menu and application icons are placed on a separate plane of the icon bar. Application menus and icons, not applicable in the current state, should be disabled by the application. This can be done with the activation commands which specify the currently enabled menus/icons, all the other menus/icons will automatically be disabled this way. To interpose the I/O operations, the application is able to take control on *daVinci's* file menu events, e.g. to store its own data structure when the user selects the 'File/Save' menu in *daVinci*.

- **set(...)**
  The set category covers commands to set the options of *daVinci* which can also be done by the user with the Option menu. Commands are available to set the accuracy of the graph layout algorithm, to keep the graph nodes at their current level with the next start of the graph layout algorithm, to specify fontsize and distances between nodes and edges in the graph visualization (*gap*), to control the scrolling behaviour of survey views, to define the animation speed used for scrolling and to switch off the pixmap cache when there is not enough memory available.
  One option, that can't be set by the user via menus, is the precedence of visualization definitions in the evaluation of attribute values for nodes and edges. You can set this option only via API, because visualization definitions can only be defined via the API.

- **window(...)**
  The window category covers commands to manipulate *daVinci's* base window. There are commands to set the window title, to show messages and status informations in the footer area, to set the window position and size, to raise the window (i.e. bring it to the front), and to iconify or deiconify it. With the commands deactivate and activate, all user input can be blocked for a time. Finally, there is a command to let the application use the *daVinci* file-browser for its own purposes.

- **tcl(...)**
  The Tcl category covers commands to access *daVinci's* Tcl/Tk interface. The user interface of *daVinci* V2.x is implemented in Tcl/Tk. This way, the application is able to execute Tcl7.4/Tk4.0 scripts in *daVinci*, for example to implement dialog windows which pop up when application menus or icons are selected. The *daVinci* Tcl/Tk interface can be used in the same way as a *wish* shell, but has an additional Tcl object *daVinci* that accepts API commands sent directly from inside the Tcl/Tk-script and makes it possible for the script to send strings back to the application. **Note**: *daVinci's* user interface and graph visualization cannot be manipulated with the Tcl/Tk interface, so *daVinci* is not a Tcl object in the common sense. *daVinci* can only be accessed by using the commands of the API.

- **special(...)**
  The special category covers commands to set the current node and egde selection, to scroll to a particular node with and without animation and to display a particular HTML-page (URL) in the Netscape Navigator WWW-browser.

- **visual(...)**
  The visual category covers commands to add new rules or modify rules of the visualization definitions.

- **drag_and_drop(...)**
  The Drag and Drop category covers commands to switch drag and drop on or off. Further commands are available to react on drag and drop events by adding a new node or a new node and edge at the drop position.

Answers from the API

All commands sent to the API are confirmed after execution by returning an answer which is usually ok, but communication error(...) when the command failed. Some commands (listed here) return another confirmation answer instead of *ok*. It is very important to wait for confirmation before sending the next command, because in rare situations, the API accepts new commands even if the previous command is still in process and not yet confirmed, so the confirmation answers may not appear in the right order.

The rest of the answers (listed here) are events that are triggered by the user (e.g. to inform the application about selection of a graph node). Events can occur at **any** time, except when an API command is currently executed and not yet confirmed.

> **VERY IMPORTANT**: Due to the strange effects of asynchronous communication, an event must also be expected by the application after sending a command to the API and before receiving the corresponding confirmation answer! *daVinci* guarantees that events are not propagated during execution of an API command, but if the event occured during transfer time of the command (i.e. *after* it is sent by the application and *before* it is received by the API), then the event answer may interfere and will arive at the application side in the time interval after sending the command and before receiving the corresponding confirmation answer. So never expect that the confirmation answer will come back directly after sending an API command. Your application should accept events at any time!

Your application is responsible for handling an event answer. The event may trigger an operation in the application or can simply be ignored.

Important Notes about the API

Here are some important points that have to be considered when using the *daVinci* API:

- **No Return characters inside API commands**
  API commands ***must not*** have newline (NL, ASCII=10) or return characters (CR, ASCII=13) inside. They must have exactly one newline at the end of the command ("end-of-line") to flush the write buffer. No other characters, even no spaces, should follow after this concluding return, except of the next complete command. When these restrictions are not considered, pipe communication with the *daVinci* API will likely fail. For example, in C you can send one command to *daVinci's* API this way, whereby `dv_pipe` is the file descriptor of the command pipe to *daVinci* (the API command is highlighted in bold face):

  ```
  fprintf(dv_pipe,"window(show_message(\"Hi\"))\n");
  ```

  Note that there is a newline at the end of the C string and that there is another API string inside the C string (which is an API command). Quotes inside C strings have to be escaped with a backslash character '\'. The same is true for *daVinci's* API strings, read the next list item.

- **No unescaped double quotes inside API commands**
  Some API commands have a string argument which is enclosed by double quotes, for example command window(show_message("This is a string")). If you have to use double quotes *inside* a string, make sure that they are always escaped with a backslash character '\'. Example: window(show_message("Here is a quote \" inside.")). But do not escape the outermost double quotes of the string!

- **Use only balanced braces ('{' and '}') in API commands**
  As a result of using Tcl/Tk for the implemantation of the user interface, some characters have a special meaning for Tcl. So we have used '{' and '}' as delimiters for sending strings to Tcl. Therefore Tcl complains about unbalanced braces, if somewhere in a string only one of both is used (i.e. "}" or "{", whereas "{}" is allowed). So please avoid unbalanced braces in strings or you'll get Tcl/Tk error messages on command-line.

- **Initial ok:**
  After connecting an application to *daVinci's* API, the application program has to wait for the initial **ok** answer before sending any commands to *daVinci*.

- **Disconnection:**
  An application will receive the answer disconnect if the user selects menu File/Disconnect. The appropriate action for this event is to terminate the application to let the user connect another application in *daVinci*.

- **Multi-Graph Mode:**
  In multi-graph mode, **each** answer has a preluding context(...) answer to specify the context (graph) where the event happened. So, your application has to receive **two** answers at a time after entering multi-graph mode.

- **multi-window Mode:**
  In multi-window mode, **each** answer has a preluding context_window(...) answer to specify the context (graph) and window where the event happened. So, your application has to receive **two** answers at a time after entering multi-window mode. The application is informed about opening (answer open_window) and closing (answer close_window(...) of windows.

- **Close and Exit:**
  If the application controls *daVinci's* File menu events, it will receive (a preluding answer context("*ID*") or context_window("*ID*") for multi-window mode and then) an answer menu_selection("#%close") each time the user wants to close a particular context (graph) by using menu File/Close. The appropriate action in this case is to set the current context (and window) to the given *ID* (*ID's*) and to close it by sending command menu(file(close)).
  In the same way, the application will receive answer menu_selection("#%exit") when the user wants to exit *daVinci* by using menu File/Exit. The appropriate action in this case is to send command menu(file(exit)) to terminate *daVinci*.

- **Migration to the V2.x API:**
  If you have an existing application using commands of the V1.x API, you should migrate to the V2.x API as soon as possible. Guidelines for the neccessary tasks can be found in the Transition Guide. Do not mix commands of the V1.x and V2.x API!

daVinci V2.1 Online Documentation - Page update: June 15, 1998

daVinci API - Transition Guide

This document is a migration guide for transition to the *daVinci* V2.x API. It is intended for programmers who have implemented applications for the previous release *daVinci* V1.x.

Term Representation of Graphs

The syntax of *daVinci* graphs (term representation) has not been modified in V2.x, but the following important restriction has to be considered:

> **In *daVinci* V2.x, all nodes and edges of a graph must have a unique node/edge-ID in the term representation.**

Node- and edge-ID's were called *labels* in former releases. A node/edge-ID is the **l** constructor in front of each **n** and **e** term in a term representation (see syntax). So please consider that the ID's are now mandatory.

New API Syntax

With version V2.x, the former V1.x API was subject to a major revision. Unlike the answers, which have been left untouched, all commands have got a new syntax and are now grouped into categories with respect to their semantic. For example, all commands dealing with graph transmission and updates are member of the graph category. Commands for calling the menu functions of the user interface can be found in the menu category and are further divided into sub-categories to reflect the menu they belong to, e.g. menu(layout(improve_all)) or menu(file(exit)).

This drastic modification of the API was neccessary to realize new features in *daVinci* V2.x (e.g. the multi-graph mode) and to simplify the use by structuring the API with intuitive categories. The former V1.x API has had a grown structure without a clear concept, so the time has come to put this in order.

Applications written for the *daVinci* V1.x API are still supported in backward compatibility mode which is automatically activated if an obsolete V1.x command is sent to the API. So usually, all V1.x applications should be able to communicate with *daVinci* V2.x without any modification.

Transform V1.x to V2.x API Commands

As soon as an application would like to use new features of *daVinci* V2.x, all commands sent to the API have to be in V2.x syntax, because most new features will not work in the compatibility mode.

Transition to the new API is easy. Unless the application does not use any new feature of *daVinci* V2.x (e.g. application icons, multi-graph mode, etc.), it will not receive new kinds of answers. The syntax of the answers that already exist in the previous V1.x API is left untouched. Moreover, the parameters of the commands, for example the term representation of a graph or the specification of application menus, have not been modified, too. So, the only task an application programmer has to do

to migrate to the new API is transforming the commands to the new V2.x syntax according to the following rules. The list shows all V1.x API commands (in alphabetic order) and their replacement in the V2.x API.

- **activate_menu_entries(***menu_ids***)**
  V2.x API: **app_menu(activate_menus(***menu_ids***))** Details.
- **activation**
  V2.x API: **window(activate)** Details.

- **change_node_color(***node_id***,***string***)**
  V2.x API: **graph(change_attr([node(***node_id***,[a("COLOR",***string***)])]))** Details.
- **close_window**
  V2.x API: **window(iconify)** Details.
- **compact_graph**
  V2.x API: **menu(layout(compact_all))** Details.
- **confirmer(***string***,***strings***,***string***)**
  Not available in V2.x API. Use *daVinci* Tcl/Tk interface for arbitrary dialogs.
- **create_menu(***menus***)**
  V2.x API: **app_menu(create_menus(***menus***))** Details.

- **deactivation**
  V2.x API: **window(deactivate)** Details.

- **focus_node(***node_id***)**
  V2.x API: **special(focus_node(***node_id***))** Details.
- **focus_node_animated(***node_id***)**
  V2.x API: **special(focus_node_animated(***node_id***))** Details.

- **hide_or_show_edges**
  Not available in V2.x API. Use **menu(abstraction(hide_edges(...)))** or **menu(abstraction(show_edges(...)))** with current selection instead. Details.
- **hide_or_show_subgraph**
  Not available in V2.x API. Use **menu(abstraction(hide_subgraph(...)))** or **menu(abstraction(show_subgraph(...)))** with current selection instead. Details.
- **hide_subgraphs(***node_ids***)**
  V2.x API: **menu(abstraction(hide_subgraph(***node_ids***)))** Details.

- **load_graph(***filename***)**
  V2.x API: **menu(file(open_graph(***filename***)))** Details.
- **load_graph_placed(***filename***)**
  V2.x API: **menu(file(open_graph_placed(***filename***)))** Details.
- **load_status(***filename***)**
  V2.x API: **menu(file(open_status(***filename***)))** Details.

- **new_term(***graphterm***)**
  V2.x API: **graph(new(***graphterm***))** Details.
- **new_term_placed(***graphtrees***)**
  V2.x API: **graph(new_placed(***graphterm***))** Details.
- **nothing**
  V2.x API: not changed, still **nothing**

- **open_window**
  V2.x API: **window(deiconify)** Details.

- **place_all_nodes**
  V2.x API: **menu(layout(improve_all))** Details.
- **place_visible_nodes**
  V2.x API: **menu(layout(improve_visible))** Details.

- **question_boolean(***string***,***strings***,***string***,***string***,***boolean***)**
  Not available in V2.x API. Use *daVinci* Tcl/Tk interface for arbitrary dialogs.
- **question_string(***string***,***strings***,***string***)**
  Not available in V2.x API. Use *daVinci* Tcl/Tk interface for arbitrary dialogs.
- **quit**
  V2.x API: **menu(file(exit))** Details.

- **restore_edges**
  V2.x API: **menu(abstraction(restore_all_edges))** Details.
- **restore_subgraphs**
  V2.x API: **menu(abstraction(restore_all_subgraphs))** Details.

- **save_graph(***filename***)**
  V2.x API: **menu(file(save_graph(***filename***)))** Details.
- **save_postscript(***filename***)**
  V2.x API: **menu(file(print(***filename***)))** Details.
- **save_status(***filename***)**
  V2.x API: **menu(file(save_status(***filename***)))** Details.
- **select_nodes_labels(***node_ids***)**
  V2.x API: **special(select_nodes(***node_ids***))** Details.
- **set_accuracy_phase_1(***integer***)**
  V2.x API: **set(layout_accuracy(***new_integer!***)** Details.
  Note: The mechanism to set the accuracy has been simplified to use only one value in the range between 1 and 5 instead of individual values for phase1 and phase2. Consider the following relations between the new value and the former phase1/phase2 values to choose an appropriate setting. Other phase1/phase2 values are no longer supported.

V2.x API	V1.x API (phase1,phase2)
1	(1,2)
2	(2,3)
3	(3,4)
4	(4,8)
5	(4,12)

- **set_accuracy_phase_2(***integer***)**
  V2.x API: **set(layout_accuracy(***new_integer!***))** Details.
  Note: Read comments for previous command **set_accuracy_phase_1** above!
- **set_font_size(***integer***)**

V2.x API: **set(font_size(***integer***))** Details.

- **set_gap_height(***integer***)**
  V2.x API: **set(gap_height(***integer***))** Details.
- **set_gap_width(***integer***)**
  V2.x API: **set(gap_width(***integer***))** Details.
- **set_multiple_edge_gap(***integer***)**
  V2.x API: **set(multi_edge_gap(***integer***))** Details.
- **set_scale(***integer***)**
  V2.x API: **menu(view(scale(***integer***)))** Details.
- **set_self_refering_edge_radius(***integer***)**
  V2.x API: **set(self_edge_radius(***integer***))** Details.
- **set_window_position(***integer***,***integer***)**
  V2.x API: **window(position(***integer***,***integer***))** Details.
- **set_window_size(***integer***,***integer***)**
  V2.x API: **window(size(***integer***,***integer***))** Details.
- **set_window_title(***string***)**
  V2.x API: **window(title(***string***))** Details.
- **show_message(***string***)**
  V2.x API: **window(show_message(***string***))** Details.
- **show_status(***string***)**
  V2.x API: **window(show_status(***string***))** Details.
- **show_subgraphs(***node_ids***)**
  V2.x API: **menu(abstraction(show_subgraph(***node_ids***)))** Details.

---

daVinci API Definition - Overview of Commands

Help about syntax

```
daVinci_commands  ::=          /*List of API commands for option -init.*/
    [daVinci_commands1].
daVinci_commands1 ::=
    daVinci_command | daVinci_command,daVinci_commands1.
daVinci_command ::=            /*Commands of the API (top-level). */
    graph(graph_cmd)                    /*Graph category. Details.*/
  | multi(multi_cmd)                    /*Multi category. Details.*/
  | menu(menu_cmd)                      /*Menu category. Details.*/
  | app_menu(app_menu_cmd)              /*App_menu category. Details.*/
  | set(set_cmd)                        /*Set category. Details.*/
  | window(window_cmd)                  /*Window category. Details.*/
  | tcl(tcl_cmd)                        /*Tcl category. Details.*/
  | special(special_cmd)                /*Special category. Details.*/
  | visual(visual_cmd)                  /*Visual category. Details.*/
  | drag_and_drop(drag_and_drop_cmd)    /*Drag and Drop category. Details.*/
  | nothing.                            /*No operation, for syncronization.*/
graph_cmd ::=                  /*Send and update graphs. Details.*/
    new(graphterm)                      /*Send new graph. Details.*/
  | new_placed(graphterm)               /*Dito, better layout. Details.*/
  | update(node_updates,edge_updates)   /*Send graph updates. Details.*/
  | update(mixed_updates)               /*Send graph updates. Details.*/
  | change_attr(attr_changes)           /*Change attributes. Details.*/
  | update_and_change_attr(node_updates,edge_updates,attr_changes)
                                        /*Combination of both. Details.*/
  | update_and_change_attr(mixed_updates,attr_changes)
                                        /*Combination of both. Details.*/
  | change_type(type_changes).          /*Change type. Details.*/

multi_cmd ::=                  /*For multi-graph mode. Details.*/
    new_context                         /*Open graph context. Details.*/
  | open_context(context_id)            /*Dito, but ID is given. Details.*/
  | set_context(context_id).            /*Switch to context. Details.*/
  | set_context(context_id,window_id).  /*Switch to context and window. Details.*/

menu_cmd ::=                   /*Call functions of menu. Details.*/
    file(file_menu_cmd)                 /*File menu category. Details.*/
  | view(view_menu_cmd)                 /*View menu category. Details.*/
  | navigation(navigation_menu_cmd)     /*Navigation menu category. Details.*/
  | abstraction(abstraction_menu_cmd)   /*Abstraction menu category. Details.*/
  | layout(layout_menu_cmd).            /*Layout menu category. Details.*/

file_menu_cmd ::=              /*File menu functions. Details.*/
    new                                 /*Clear graph. Get Details.*/
  | open_graph(filename)                /*Load graph from file. Details.*/
  | open_graph_placed(filename)         /*Dito, better layout. Details.*/
  | open_status(filename)               /*Load status from file. Details.*/
  | save_graph(filename)                /*Save graph as term. Details.*/
  | save_status(filename)               /*Save graph as status. Details.*/
  | print(filename)                     /*Save as PostScript. Details.*/
  | print                               /*Open print dialog. Details.*/
  | close                               /*Close graph window. Details.*/
  | exit.                               /*Exit daVinci. Details.*/
```

```
view_menu_cmd ::=               /*View menu functions. Details.*/
    open_new_view                       /*Open additional view. Details.*/
  | open_survey_view                    /*Open survey view. Details.*/
  | full_scale                          /*Set scale to 100%. Details.*/
  | fit_scale_to_window                 /*Set scale to fit. Details.*/
  | scale(integer)                      /*Set scale to integer. Details.*/
  | scale                               /*Open Scale dialog. Details.*/
  | graph_info                          /*Open Graph Info dialog. Details.*/
  | daVinci_info.                       /*Open daVinci Info dialog. Details.*/

navigation_menu_cmd ::=         /*Navigation menu functions. Details.*/
    select_parents(node_ids)            /*Select parents of nodes. Details.*/
  | select_siblings(node_ids)           /*Select siblings of nodes. Details.*/
  | select_children(node_ids)           /*Select children of nodes. Details.*/
  | navigator(node_id,direction,boolean)/*Navigate in graph. Details.*/
  | navigator                           /*Open Navigator dialog. Details.*/
  | find(string,boolean,boolean)        /*Find a node. Details.*/
  | find.                               /*Open Find dialog. Details.*/

abstraction_menu_cmd ::=        /*Abstraction menu functions. Details.*/
    hide_subgraph(node_ids)             /*Hide subgraphs of nodes. Details.*/
  | show_subgraph(node_ids)             /*Show subgraphs of nodes. Details.*/
  | restore_all_subgraphs               /*Show all hidden subgr. Details.*/
  | hide_edges(node_ids)                /*Hide edges of nodes. Details.*/
  | show_edges(node_ids)                /*Show edges of nodes. Details.*/
  | restore_all_edges.                  /*Show all hidden edges. Details.*/

layout_menu_cmd ::=             /*Layout menu functions. Details.*/
    improve_all                         /*Start layout algorithm. Details.*/
  | improve_edges                       /*Dito, only dummy nodes. Details.*/
  | improve_selected                    /*Dito, only selected and dummy nodes. Details.*/
  | improve_nodes(node_ids)             /*Dito, only supplied and dummy nodes. Details.*/
  | improve_spacing                     /*Compact graph layout. Details.*/
  | orientation(orientation).           /*Switch orientation. Details.*/

app_menu_cmd ::=                /*Create menus/icons. Details.*/
    create_menus(menus)                 /*Add menus in Edit. Details.*/
  | create_icons(icons)                 /*Add icons in icon-bar. Details.*/
  | activate_menus(menu_ids)            /*Enable menus. Details.*/
  | activate_icons(icon_ids)            /*Enable icons. Details.*/
  | control_file_events.                /*Get events of File menu. Details.*/

set_cmd ::=                     /*Set options. Details.*/
    layout_accuracy(integer)            /*Layout algorithm params. Details.*/
  | keep_nodes_at_levels(boolean)       /*Keep nodes at levels. Details.*/
  | font_size(integer)                  /*Node font size. Details.*/
  | gap_width(integer)                  /*Min. node distance. Details.*/
  | gap_height(integer)                 /*Min. level distance. Details.*/
  | multi_edge_gap(integer)             /*Distance for multi-edges. Details.*/
  | self_edge_radius(integer)           /*Distance for self-edges. Details.*/
  | scrolling_on_selection(boolean)     /*Auto focusing node. Details.*/
  | animation_speed(integer)            /*Speed of animation. Details.*/
  | no_cache(boolean)                   /*Control pixmap caching Details.*/
  | rules_first(boolean).               /*Control attribute evaluation. Details*/

window_cmd ::=                  /*Control windows. Details.*/
    title(string)                       /*Set window title. Details.*/
  | show_message(string)                /*Left footer message. Details.*/
  | show_status(string)                 /*Right footer message. Details.*/
  | position(integer,integer)           /*Window origin x/y. Details.*/
  | size(integer,integer)               /*Window width/height. Details.*/
  | raise                               /*Raise window. Details.*/
  | iconify                             /*Iconify window. Details.*/
  | deiconify                           /*Deiconify window. Details.*/
  | activate                            /*Enable interaction. Details.*/
  | deactivate                          /*Disable interaction. Details.*/
  | file_browser(boolean,string,string,string,string,btypes,boolean).
```

```
                                        /*Show file browser. Details.*/

tcl_cmd ::=                         /*Tcl/Tk interface. Details.*/
    eval(string)                            /*Eval Tcl/Tk script. Details.*/
  | eval_file(filename).                    /*Dito, from file. Details.*/

special_cmd ::=                     /*Special commands. Details.*/
    select_nodes(node_ids)                  /*Select specified nodes. Details.*/
  | select_edge(edge_id)                    /*Select specified edge. Details.*/
  | focus_node(node_id)                     /*Scroll to specified node. Details.*/
  | focus_node_animated(node_id)            /*Dito, with animation. Details.*/
  | show_url(string).                       /*Display HTML-page. Details.*/

visual_cmd ::=                      /*Visualization definitions. Details.*/
    new_rules(visual_rules)                 /*Define new rules. Details.*/
  | add_rules(visual_rules).                /*Add or modify rules. Details.*/

drag_and_drop_cmd ::=                       /*Drag and Drop. Details.*/
    dragging_on                             /*Drag and Drop on. Details.*/
  | dragging_off                            /*Drag and Drop off. Details.*/
  | new_node_at_coord(node_update)          /*Insert node at drop position. Details.*/
  | new_edge_and_node_at_coord(node_update,edge_update).
                                        /*Insert node and edge at drop position. Details.*/
```

---

daVinci V2.1 Online Documentation - Page update: June 15, 1998

daVinci API - Syntax

Commands and answers of the *daVinci* API are terms which can be build hierarchically by applying substitution rules of the syntax definition. The syntax of commands and answers is defined in Backus-Naur form. The syntax grammar has a set of substitution rules, where each rule may have different alternative cases on the right hand side, separated by bars (|). Terminal symbols of the grammar are written in bold face and need to occur in the term. Non-terminal symbols are links to other rules and need to be substituted by one of the cases of the particular rule. This substitution process continues until no non-terminal symbols remain. To say it in another word: Commands and answers are exclusively composed of terminal symbols. The start symbol for the commands is rule daVinci_command and for the answers daVinci_answer. For example, consider the following rules:

```
daVinci_command ::=
    tcl(tcl_cmd)
  | (... other cases deleted ...)

tcl_cmd ::=
    eval(string)
  | eval_file(filename).

filename ::= string.
```

Here are some examples for valid commands with respect to this grammar (but please do not try to send such Tcl-scripts to *daVinci*). :-)

```
tcl(eval("Hello"))
tcl(eval_file("daVinci"))
```

Usually, whitespace (i.e. space, tab or return/newline characters between non-terminal symbols) is allowed, but should not be used in a term to reduce the amount of transmitted data. **Return or newline characters must not be used in a term if send to the API**. The reason for this restriction is that a newline character is needed to signal the end of a command if sent on a pipe, i.e. the buffer will flush as soon as a newline appears. So, with a newline inside the term, it will be split at this position, resulting in a syntax error for the fragment. If you need to have newline characters inside of a string (e.g. to visualize a node with multiline text), you can use the well known C escape mechanism of ("\n").

Note: As a result of using Tcl/Tk for the implemantation of the user interface, some characters have a special meaning for Tcl. So we have used '{' and '}' as delimiters for sending strings to Tcl. Therefore Tcl complains about unbalanced braces, if somewhere in a string only one of both is used (i.e. "}" or "{", whereas "{}" is allowed). So please avoid unbalanced braces in strings or you'll get Tcl/Tk error messages on command-line.

daVinci API Definition - Parameters

Help about syntax

```
--- Term Representation for Graphs: ----------------------------------------
graphterm     ::= [graphterm1] | [].    /*Graph with root nodes. Details.*/
graphterm1    ::= node | node,graphterm1.
node          ::=
    l(node_id,n(type,attributes,edges)) /*Node with ID/type/attr/children.*/
  | r(node_id).                         /*Reference to a node.*/

edges         ::= [edges1] | [].        /*Edges with ID/type/attr/child.*/
edges1        ::= edge | edge,edges1.
edge          ::=
    l(edge_id,e(type,attributes,node)).

attributes    ::= [attributes1] | [].   /*Attributes (key/value). Details.*/
attributes1   ::= attribute | attribute,attributes1.
attribute     ::=
    a(string,string)                    /*Normal key/value attribute pair.*/
  | m(menus).                           /*Popup menu attribute for a node/edge.*/



--- Graph Updates: ---------------------------------------------------------
mixed_updates  ::= [mixed_updates1] | [].       /*Delete or remove nodes and edges. Details.*/
mixed_updates1 ::= mixed_update | mixed_update,mixed_updates1.
mixed_update   ::= node_update | edge_update.

node_updates  ::= [node_updates1] | []. /*Delete or remove nodes. Details.*/
node_updates1 ::= node_update | node_update,node_updates1.
node_update   ::=
    delete_node(node_id)
  | new_node(node_id,type,attributes).

edge_updates  ::= [edge_updates1] | []. /*Delete or remove edges. Details.*/
edge_updates1 ::= edge_update | edge_update,edge_updates1.
edge_update   ::=
    delete_edge(edge_id)
  | new_edge(edge_id,type,attributes,node_id,node_id).
  | new_edge_behind(edge_id,edge_id,type,attributes,node_id,node_id).

attr_changes  ::= [attr_changes1] | []. /*Change attributes. Details.*/
attr_changes1 ::= attr_change | attr_change,attr_changes1.
attr_change   ::=
    node(node_id,attributes)
  | edge(edge_id,attributes).

type_changes  ::= [type_changes1] | []. /*Change types. Details.*/
type_changes1 ::= type_change | type_change,type_changes1.
type_change   ::=
    node(node_id,type)
  | edge(edge_id,type).


--- Application Menus and Icons: -------------------------------------------
menus      ::= [menus1] | [].           /*Create Menus. Details.*/
menus1     ::= menu_entry | menu_entry,menus1.
menu_entry ::=
    menu_entry(menu_id,menu_label)
  | menu_entry_mne(menu_id,menu_label,menu_mne,menu_mod,menu_acc)
  | menu_entry_disabled(menu_id,menu_label)
  | submenu_entry(menu_id,menu_label,menus)
```

```
  | submenu_entry_mne(menu_id,menu_label,menus,menu_mne)
  | submenu_entry_disabled(menu_id,menu_label,menus)
  | blank.

icons      ::= [icons1] | [].          /*Create Icons. Details.*/
icons1     ::= icon_entry | icon_entry,icons1.
icon_entry ::=
    icon_entry(icon_id,filename,string)
  | blank.


--- Visualization definitions: ------------------------------------------
visual_rules  ::= [visual_rules1] | []. /*Visualization definitions. Details.*/
visual_rules1 ::= visual_rule | visual_rule,visual_rules1.
visual_rule   ::=
    nr(type,attributes)
  | er(type,attributes).


--- String Sorts: -------------------------------------------------------
node_ids   ::= [node_ids1] | [].        /*Many node ID's.*/
node_ids1  ::= node_id | node_id,node_ids1.
menu_ids   ::= [menu_ids1] | [].        /*Many menu ID's.*/
menu_ids1  ::= menu_id | menu_id,menu_ids1.
icon_ids   ::= [icon_ids1] | [].        /*Many icon ID's.*/
icon_ids1  ::= icon_id | icon_id,icon_ids1.

node_id    ::= string.                  /*Unique node ID. Details.*/
edge_id    ::= string.                  /*Unique edge ID. Details.*/
menu_id    ::= string.                  /*Unique menu ID. Details.*/
menu_label ::= string.                  /*Text of menu entry. Details.*/
menu_mne   ::= string.                  /*Motif mnemonic char. Details.*/
menu_acc   ::= string.                  /*Motif accelerator key. Details.*/
icon_id    ::= string.                  /*Unique icon ID. Details.*/
type       ::= string.                  /*Arbitrary type. Details.*/
filename   ::= string.                  /*Valid filename.*/
context_id ::= string.                  /*Context ID. Details.*/
window_id ::= string.                   /*Window ID. Details.*/
string     ::= ... /*Informally: Character sequence without newline,
                     carriage return and double quotes, enclosed by double
                     quotes ("). For example: "This is a string".
                     Special characters can be written in C-notation, e.g.
                     \n for newline or \" for a quote inside the string.*/


--- Basic Sorts: --------------------------------------------------------
integer    ::= ... /*Informally: Non-floating positive number.*/
boolean    ::= true | false.
orientation::= top_down | bottom_up | left_right | right_left.
direction  ::= up | down | left | right.
btypes     ::= [btypes1].               /*Types for file browser. Details.*/
btypes1    ::= btype | btype,btypes1.
btype      ::= bt(string,string,string)./*Text, pattern and title postfix.*/
menu_mod   ::= alt | shift | control | meta | none.
                                        /*Motif modifier key. Details.*/
```

---

daVinci V2.1 Online Documentation - Page update: June 15, 1998

daVinci API Definition - Graph Commands

On this page, API commands of the graph category are described. This category covers all commands to send graphs in term representation and updates on graphs. The commands of this category are enclosed by keyword **graph(...)**. Click here to get an overview of all API commands.

graph(new(graphterm))

Sends a graph in term representation *graphterm* to *daVinci* for visualization. This is the same format used for file representation of *daVinci* graphs (i.e. the .daVinci files). **Note:** Unlike the file format, newline characters are not allowed in term representations if sent to the API.

After transmitting a graph by using this command, the layout algorithm for edge crossing minimization will not be started automatically. You can either do this later using command menu(layout(improve_all)) or by combining both steps with command graph(new_placed(...)), see below. The advantage of the second method is that the user will not see an unplaced graph without an applied crossing minimization.

If you do not like to send the whole graph via API, you can save it to a file and use command menu(file(open_graph(...))) instead for loading the graph from file.

graph(new_placed(graphterm))

This command combines graph(new(...)) (i.e. transmitting a graph) and menu(layout(improve_all)) (i.e. starting the layout algorithm to minimize edge crossings) in one step. The advantage of using this command (instead of the previous one) is that the user will not see an unplaced graph without an applied crossing minimization. The parameter *graphterm* is the term representation of the graph which should be visualized.

If you do not like to send the whole graph via API, you can save it to a file and use command menu(file(open_graph_placed(...))) instead for loading the graph from file.

graph(update(node_updates,edge_updates))
graph(update(mixed_updates))

These commands can be used to update the structure of a current graph. You can remove existing nodes and edges, add new nodes or connect nodes with new edges. For the first command a graph update consist of two lists: One for the node and one for the edge updates. For the second command, updates for nodes and edges are supplied in one list. After sending this command, incremental layout will be used to integrate the update into an existing layout to ensure visual stability. After an update, all previously selected nodes (and edges) are always deselected *without* sending API answer node_selections_labels(...)

We describe the possible updates divided into node and edge updates. If you use the **mixed_updates** form of the command, you can use elements of both lists, which can be mixed in arbitrary order.

For the list of **node_updates**, there are two different kinds of list elements which can be mixed in arbitrary order:

- **delete_node(**node_id**)**
  Delete the node with given *node_id*. The ID must exist in the current graph. All edges of this node, from the parents and to the children, are implicitly removed as well.

- **new_node(**node_id**,**type**,**attributes**)**
  Insert a new node in the graph with given *node_id*, type and node attributes. The node ID must be unique for the set of nodes in the current graph and update. If the new node is not connected with new edges in the same update (see below), it will be visualized as an unconnected root node.

For the list of **edge_updates**, there are three different kinds of list elements which can be mixed in arbitrary order:

- **delete_edge(**edge_id**)**
  Delete the edge with given *edge_id*. The ID must exist in the current graph. **Note**: The edges from the parents and to the children of a node removed in the same update must not be deleted, because they are removed implicitly.

- **new_edge(**edge_id**,**type**,**attributes**,**node_id**,**node_id**)**
  Insert a new edge in the graph with given *edge_id*, type and edge attributes. The edge ID must be unique for the set of edges in the current graph and update. The first *node_id* is the parent and the second *node_id* is the child of the edge. Both, the child and parent, have to be ID's of existing nodes in the current graph or new nodes of the update. **Note**: You cannot delete the specified parent and child node in the same update.

- **new_edge_behind(**edge_id**,**edge_id**,**type**,**attributes**,**node_id**,**node_id**)**
  Same as **new_edge** (read directly above), but this will enter the new edge behind a left neighbour edge which is specified by the additional first *edge_id*. So, the new edge will be the right neighbour of the edges specified by the first *edge_id*. The left neighbour edge has to be the ID of an existing edge in the current graph or a new edge of the update. At the moment, there is no need to insert an edge at a specific position, because the order of child nodes is currently not significant. This may change in a future version of *daVinci* when visualization definitions will be available.

graph(change_attr(attr_changes))

This command can be used to update the attributes of nodes and edges that are already present in the current graph. The semantic of this operation is either to complete or overwrite the existing attributes of a particular node or edge. For example, in the current graph, a node is specified to be drawn in red with text "test", i.e. attributes are:

[a("COLOR","red"), a("OBJECT","test")]

Now consider the following attribute update:

[a("OBJECT","new test"),a("FONTFAMILY","times")].

After sending this update, the node will be still drawn in red (the COLOR attribute is not overwritten), but with text "new test" in font "times".

The parameter of the command is a list of attribute changes with two different kinds of list elements. Both types of elements can be mixed in arbitrary order:

- **node**(node_id**,**attributes**)**
  Complete or overwrite the attributes for the node specified by *node_id* which has to be the ID of an existing node in the current graph.

- **edge**(edge_id**,**attributes**)**
  Complete or overwrite the attributes for the edge specified by *edge_id* which has to be the ID of an existing edge in the current graph.

graph(update_and_change_attr(node_updates,edge_updates,attr_changes))
graph(update_and_change_attr(mixed_updates,attr_changes))

(Only available in *daVinci* V2.0.3 or higher.) This command combines the API commands graph(update(...)) and graph(change_attr(...)) (described above) to update the structure of a graph and modify attributes of existing nodes in one step. The benefit of this command instead of using the other two commands directly after eachother is that the graph is not displayed after the update and before attribute modification. First the *..._updates* are applied to the graph and afterwards the *attr_changes* are applied to the updated graph. So for example it is not allowed to modify attributes of nodes in *attr_changes* that are removed in *..._updates*.

graph(change_type(type_changes))

This command can be used to update the type of nodes and edges that are already present in the current graph.

The parameter of the command is a list of type changes with two different kinds of list elements. Both types of elements can be mixed in arbitrary order:

- **node**(node_id**,**type**)**
  Change the type for the node specified by *node_id* which has to be the ID of an existing node in the current graph.

- **edge**(edge_id**,**type**)**
  Change the type for the edge specified by *edge_id* which has to be the ID of an existing edge in the current graph.

This command is especially usefull, if you've supplied several visualization rules for different node and edge types. Than you can change the appearance of a node or edge by changing its type.

---

daVinci API Definition - Multi Commands

On this page, API commands of the multi category are described. This category covers all commands to work with more than one graph at a time. The commands of this category are enclosed by keyword **multi(...)**. Click here to get an overview of all API commands.

Multi-Graph Background:

daVinci V2.x is able to visualize more than one graph at a time (*multi-graph mode*). Read about multi-graph in the concept document before going on here.

In multi-graph mode, each loaded graph is shown in a different base window. Operations on one graph do not affect the other graphs. To deal with different graphs at the API level, the notion of a *context* is introduced in *daVinci* V2.x. Each context has its own scope, so it is not neccessary to keep identifiers (e.g. for nodes and edges in a term representation or for menus) unique between two contexts, but of course all identifiers have to be unique in one context.

For each graph, there is a context which has a unique context_id. The *context_id* is either choosen and returned by *daVinci* when a new context is opened with command multi(new_context) or can be choosen by the application when the new context is opened with the alternative command multi(open_context(...)). After opening a context, which pop up a fresh and empty base window, all commands send to the API are applied to this current context until the current context is set to another one. So for example, by sending command menu(file(open_graph(...))) directly after opening a new context, the current context has been automatically switch to the new context before and the graph will be displayed in the new window. If the application wants to send an API command to another graph that has been previously opened, then it has to set the current context to the context of this graph first. The current context which specifies the graph currently affected by all the other API commands can be set with command multi(set_context(...)).

Two Answer Concept for Multi-Graph

On the other side, after entering multi-graph mode (i.e. by sending the first multi(new_context) or multi(open_context(...)) command), each answer from the API will have a preluding context(...) answer which specifies the context where the event happened. So, an application will **always** get **two answers** with each event in multi-graph mode: first the context and then the event itself. Note: The preluding context answer does not switch the current context which can only be done by the application itself.

To ensure full compatibility with applications designed for *daVinci* V1.x (who do not expect a preluding context), the context answer is only used after entering multi-graph mode. At startup time, *daVinci* is in single-graph mode by default. The user cannot enter the multi-graph mode if an application is connected to the API, only the application can decide to do so.

Enter the Multi-Graph Mode at Initialization Time

If the application would like to use the multi-graph mode later in a session, it should open the first context at initilization time. For consistency reasons, *daVinci* will remove a probably loaded graph as soon as the first context is opened (i.e. when the multi-user mode is entered).

The application is able to close an opened context by sending command menu(file(close)). This is the equivalent of menu File/Close. After closing a context, all of the windows of the particular graph will be removed, and the corresponding *context_id* cannot be used any further. Afterwards, the application need to open a new context or specify one of the remaining contexts to be the new current context. All other commands will be confirmed with an error message, because the current context is undefined as long as a new one is not specified. It is even possible to close the last context without quitting *daVinci*. After closing the last context, no window will be visible any longer, but the *daVinci* process is still running. In this situation, the application can open a new first context and a new base window will come back.

Note: The user is also able to close a context at any time by using menu File/Close. The API will send anwer close after the user or the application has decided to close a context. If it is not appropriated to let the user closing a context, then the application has the opportunity to interpose *daVinci's* File menu to catch these events.

Multi-View Background:

Another issue is the *multi-view mode* where different base windows are used to visualize the same graph. In opposite to multi-graph, multi-view windows are based on the same graph structure, so working with one view manipulates all the other views of a graph, too. Multi-graph and multi-view can occur at the same time, e.g. several graphs can be loaded and each one may have different views. Multi-view is a feature of the user interface and cannot be controlled from the API, except of the opportunity to open a new view with command menu(view(open_new_view)). So it is not possible for an application to close a particular view. All views (base windows) of a context (graph) are removed if the context is closed with command menu(file(close)).

The user is able to open and close new views for a graph at any time. The first (i.e. oldest) view of a context is called *master view*. The master view can be closed by the user without closing the whole context as long as there is one other opened view of the same graph which will become the new master view. For example, when the user opens one additional view with menu View/Open New View, then he can work with the same graph in two different base windows. Now, he/she is able to close the first view (master view) with menu File/Close without closing the whole graph. The second remaining view will serve as the new master view in this situation.

Master View

Most of the commands dealing with one base window rather than a graph/context (e.g. setting the scale, focusing nodes or setting window position/extension, etc.) are applied only to the *master view* window of a context and not to all of its views. The master view is the oldest view of a context. The other views cannot be manipulated with API commands, because multi-view is a feature of the user interface rather than the API.

Multi-Window Background:

Usually the application has no access to individual windows (views) of a context (graph). In *daVinci* V2.1 we have the multi-window mode for the API, in which an application can access all base-windows opened in *daVinci*. You can set *daVinci* into this new mode with the command-line option -window_api. Than the multi-window mode is entered the same way as it is described above for the multi-graph mode.

After entering multi-window mode (i.e. by sending the first multi(new_context) or multi(open_context(...)) command), each answer from the API will have a preluding context_window(...) answer which specifies the context and the window where the event happened. So, an application will **always** get **two answers** with each event in multi-window mode: first the context and then the event itself. Note: The preluding context_window answer does not switch the current context and window which can only be done by the application itself.

To ensure full compatibility with applications designed for *daVinci* V1.x (who do not expect a preluding context_window), the context_window answer is only used after entering multi-window mode. At startup time, *daVinci* is in single-graph mode by default.

In multi-window mode the application is informed by the events open_window and close_window(...) about the opening or closing of a window.

Documentation of the Category Commands:

multi(new_context)

This command opens a new context by operating in multi-graph mode mode. If *daVinci* has been in single graph mode before and a graph was loaded in this situation, then the graph will be automatically removed to always start with an empty window for a new context (read note above). Otherwise, if *daVinci* has already been in multi-graph mode, then an additional empty base window will appear. In both cases, the current context (which specifies the graph that is currently affected by all other commands sent to the API) will be the new context until this is changed with command multi(set_context(...)) or another context is opened.

As with most commands, answer ok is sent back to the application to confirm this command. In multi-graph mode (always true after using this command), each answer sent back to the application has a preluding context(...) (or context_window(...)) answer that specifies the graph where the event happened, read above. If a new context is opened, *daVinci* will choose a new context id (and window id) which will be returned to the application in the preluding *context(...)* (or *context_window(...)*) answer. This means, the command always returns two answers, *context(...)* (or *context_window(...)*) with the ID of the new context (and the new window) and then *ok*). Your application has to store this *context_id* (and *window_id*) to apply commands to this context (and window) later.

The *context_id* (and *window_id*), returned by this command, is used in the preluding context (context_window) answer of each answer that is sent back to the application in multi-graph (multi-window) mode. Further, the *context_id* (and *window_id*) can be used by the application to set the current context with command multi(set_context(...)).

multi(open_context(context_id))

This command opens a new context in the same way like multi(new_context) do (see above), but the *context_id* is not choosen by *daVinci*, but given by the application as parameter. **Note**: An initial underscore ('_') is not allowed for *context_id's* given by the application.

The command returns answer ok with a preluding context(...) (or context_window(...)) answer, where the *context_id* is the one used in the parameter of the command. The current context will be the new one until this is changed with command multi(set_context(...)) or another context is opened.

multi(set_context(context_id))
multi(set_context(context_id,window_id))

The first form of the command is used in multi-graph, the second in multi-window mode to set the current context to *context_id* which specifies one of the graphs in multi-graph mode (and *window_id*, which specifies one of the windows of the context). All commands sent to the API are applied to the current context (and window), so to manipulate different graphs (and windows), the application has to switch between the contexts (and windows) by using this command. The *context_id* parameter of the command must be one of the identifiers returned from the API after opening a context. The *window_id* parameter of the command (second form) must be one of the identifiers returned from the API after opening a context or a window. The specified context (and window) needs to be still open and the context must not be closed by the user or the application (this is propagated with a close answer).

---

daVinci API Definition - Menu Commands

On this page, API commands of the menu category are described. This category covers all commands to call the functions of the menus which can be found in the *daVinci* base window. The commands of this category are enclosed by keyword **menu(...)**. The menu commands are further divided into sub-categories with respect to the menu they belong to, so for example, you can find the functions of menu File in the **menu(file(...))** sub-category, and so on. Here is a list of all sub-categories on this page:

- File Menu Category
- View Menu Category
- Navigation Menu Category
- Abstraction Menu Category
- Layout Menu Category

Click here to get an overview of all API commands.

File Menu Category:

The commands of the file category are responsible for input and output, i.e. loading, saving, and printing. Further, there are commands to close the current context (i.e. the actual graph in multi-graph mode) and to terminate the *daVinci* process. The file category is the API equivalent of the File menu of the user interface.

menu(file(new))

Remove the current graph and clear the window.

menu(file(open_graph(filename)))

Loads a new graph from file. The parameter *filename* (should not be relative!) specifies a file containing a *daVinci* graph in term representation.

After loading a graph with this command, the layout algorithm for edge crossing minimization is not started automatically. You can either do this later using menu(layout(improve_all)) or by combining both steps with command menu(file(open_graph_placed(...))), see below. The advantage of the second method is that the user will not see an unplaced graph without having applied crossing minimization.

As an alternative, you can avoid file access by sending the graph in term representation directly to the API using command graph(new(...)).

menu(file(open_graph_placed(filename)))

This command combines menu(file(open_graph(...))) (i.e. loading a graph) and menu(layout(improve_all)) (i.e. starting layout algorithm to minimize edge crossings) in one step. The advantage of using this command (instead of the previous one) is that the user will not see an unplaced graph without applied crossing minimization. The parameter *filename* (should not be relative!) specifies a file containing a *daVinci* graph in term representation.

As an alternative, you can avoid file access by sending the graph in term representation directly to the API using command graph(new_placed(...)).

menu(file(open_status(filename)))

Loads a status from file *filename* (should not be relative!). The file has to exist and must contain a status previously saved by *daVinci*. A status is a graph together with layout informations and user interface settings.

menu(file(save_graph(filename)))

Saves the current graph in term representation to the specified file *filename* (should not be relative!). The filename should have the prefix .daVinci. By saving a term representation, only the graph structure will be stored and the layout informations will be lost.

menu(file(save_status(filename)))

Saves the status of the current graph to the specified file *filename* (should not be relative!). The filename should have the prefix *.status*. By saving a status, the graph structure will be stored as well as layout informations and user interface settings.

menu(file(print(filename)))

Saves a PostScript representation of the current graph to the specified file *filename* (should not be relative!). The filename should have the prefix *.ps*. The generated file is EPSF compliant (encapsulated PostScript) and can either be printed on a PostScript printer or imported with appropriated software. Read more about the PostScript interface in the reference.

The options of the PostScript interface, available in the print dialog window of the user interface, are not accessible from the API. This command will always use PostScript scale *1:1* without centering.

menu(file(print))

Opens the print dialog window if a graph is loaded. When no graph is loaded in the current context, nothing will happen.

menu(file(close))

-> Returns answer close!

Closes the current context in multi-graph multi-window mode. This will remove all windows which shows the graph of the current context. Afterwards, the closed context cannot be used any further. To proceed, the application has to open a new context with command multi(new_context) or specify one

of the remaining contexts to be the current context with command multi(set_context(...)). **All other commands will be ignored as long as a new current context is not specified!**

This command is not available in single-graph mode. Even the last context may be closed without quitting *daVinci*. When closing the last context, no window will be visible, but the *daVinci* process is still running and accepts commands to open a new first context.

menu(file(exit))

Quits *daVinci*. This will immediately terminate the *daVinci* process.

View Menu Category:

The commands of the view category are used to open additional views (windows) to the current graph, to set the scale and to show some windows with information. The view category is the API equivalent of the View menu of the user interface.

menu(view(open_new_view))

Opens a new view of the current graph which is an additional base window showing the same graph (multi-view). Each view of a graph is based on the same logical structure, so selection of a node or fine-tuning operations in one view will manipulate all the other views of the same graph, too. One exception is the scale, so each view may have its own scale.

From a technical point of view, multi-view windows are not the same as multi-graph windows which show different graphs and are independent from eachother. Multi-view is a feature of the user interface, so an application is not able to control multi-view windows (e.g. to close one particular window). But the application has the opportunity to open additional views by using this command. All views (i.e. base windows) of a context (i.e. graph) will be removed when the context is closed, either by the user or the application with command menu(file(close)). The user may open and close new views for a graph at any time.

Note: In multi-view mode, all commands dealing with one base window rather than a graph (e.g. setting the scale, focusing nodes or setting the window position) are always applied only to the master view and not to all views of the current context. The master view is the oldest view of a context.

In multi-window mode this command returns answer open_window. All commands dealing with one base window rather than a graph (e.g. setting the scale, focusing nodes or setting the window position) are always applied to the current window.

menu(view(open_survey_view))

This command can be used to open a special survey view showing the whole graph in reduced scale, whereby the currently visible part of the master view is visualized with a blue rectangle. This rectangle is updated as soon as scrolling is used in the master view. When the user selects a node in the survey view, *daVinci* will automatically scroll to this node in the master view.

Refer to command menu(view(open_new_view)) for general information about multi-view.

In multi-window mode this command returns answer open_window.

menu(view(full_scale))

Sets the scale to 100% such that all details are visible. In multi-view mode, this operation will only affect the master view of the current context.

menu(view(fit_scale_to_window))

Sets the scale to the maximal value such that the graph is completely visible. In multi-view mode, this operation will only affect the master view of the current context.

menu(view(scale(integer)))

Sets the scale to the given parameter *integer*. The *integer* is a percent value and has to be in the range between 1 and 100. So, with a scale of *20* the graph is drawn five times smaller than usual. In scales smaller than 100, the text of nodes is not displayed and other details are dropped as well. If you like to have the graph in a smaller scale *and* need to see the text of nodes, you can set a smaller font size or gap width/height by using the API commands set(font_size(...)), set(gap_width(...)) and set(gap_height(...)). In multi-view mode, this operation will only affect the master view of the current context.

menu(view(scale))

Opens the scale dialog window if a graph is loaded. When no graph is loaded in the current context, nothing will happen.

menu(view(graph_info))

Opens the graph info dialog window if a graph is loaded. When no graph is loaded in the current context, nothing will happen.

menu(view(daVinci_info))

Opens the *daVinci* info dialog window.

Navigation Menu Category:

The commands of the navigation category are changing the selection, i.e. the set of currently selected nodes. There are commands to highlight the parent-, sibling- and child nodes for a given set of nodes. Further, there are commands to navigate to related nodes and find particular nodes based on textual search. The navigation category is the API equivalent of the Navigation menu of the user interface.

All commands of this category do not return answer ok for confirmation. Instead, they return the new selection by using answer node_selections_labels(...). Of course the corresponding operation of a navigation command may fail, for example when none of the nodes match with the string used in command menu(navigation(find(...))). A failed navigation command will not modify the selection and will return the current (former) selection as result. So, to see if the command was successful or not, you have to compare the node selection before and after sending the navigation command. If both

selections are the same, then the navigation has failed, otherwise it was successful.

Note: If you are looking for an API command to select a specified set of nodes and edges in the graph, then this category is obviously the wrong place (because nodes are *not* selected via menus in *daVinci*, but with the mouse). You can select particular nodes and edges with the API commands found in the special(...) category.

menu(navigation(select_parents(node_ids)))

-> Returns answer node_selections_labels(...)!

Selects all parent nodes of the nodes specified by list *node_ids*. The returned answer is either the list of parents (which is the new current selection) or the former selection when this operation fails (i.e. if there are no parents for the given nodes).

menu(navigation(select_siblings(node_ids)))

-> Returns answer node_selections_labels(...)!

Selects all sibling nodes (i.e. children or all parents) of the nodes specified by list *node_ids*. The returned answer is either the list of siblings (which is the new current selection) or the former selection when this operation fails (i.e. if there are no siblings for the given nodes).

menu(navigation(select_children(node_ids)))

-> Returns answer node_selections_labels(...)!

(Only available in *daVinci* V2.1 or higher. This command was menu(navigation(select_childs(..))) in daVinci V2.0.x which is still supported by V2.1.x for backward compatibility.)
Selects all child nodes of the nodes specified by list *node_ids*. The returned answer is either the list of children (which is the new current selection) or the former selection when this operation fails (i.e. if there are no children for the given nodes).

menu(navigation(navigator(node_id,direction,boolean)))

-> Returns answer node_selections_labels(...)!

Selects the nearest neighbour node according to the given *direction* by starting at the specified node *node_id*. If the *boolean* parameter is *false*, geometrical navigation is used. This ignores the relations (edges) in the graph, so for example, if you navigate upwards, you will go to the nearest node at the previous level. On the other side, by using *true* for the *boolean* parameter, structural navigation is used which takes the relations of a graph into account. Navigating upwards with structural navigation will select the nearest parent node, according to the center x-coordinates of all parents.

Note: The navigation command will always consider the current layout orientation which can be set with command menu(layout(orientation(...))). So, *down* in a graph with left-to-right layout will go to the same node as *left* in a top-down layout of the same graph.

The returned answer is either the node the navigation command has moved to (which is the new current selection) or the former selection when this operation fails (e.g. by navigating upwards starting at the topmost node in a top-down layout).

menu(navigation(navigator))

Opens the navigator dialog window.

menu(navigation(find(string,boolean,boolean)))

-> Returns answer node_selections_labels(...)!

Searches for a node with text *string*. The node text is the value of the node attribute OBJECT in the term representation. The first *boolean* parameter of the command is a flag to consider the case or not. If this parameter is false, then upper and lower case characters (e.g. 'A' and 'a') are not distinguished from eachother for the search. The second *boolean* parameter is a flag to use exact match or not. If this parameter is false, then for example the search string "Vin" will match a node with text "daVinci", otherwise not. If exactly one node is selected before using this commands, then search will start at this node. Otherwise, the search will start at the leftmost and topmost node (for top-down layouts).

The returned answer is either the first node found by the search operation (which is the new current selection) or the former selection when this operation fails (e.g. when there is no node in graph that matches the string).

menu(navigation(find))

Opens the find dialog window.

Abstraction Menu Category:

The commands of the abstraction category are responsible for hiding particular details in a graph visualization. An abstraction does not manipulate the internal graph structure, it only has a temporary influence on the visualization of the graph. At the moment, there are two abstractions implemented in *daVinci*: Subgraph and edge hiding. After hiding subgraphs or edges, it will be possible to undo these operations later. The abstraction of a particular node can be undone with the *show...* commands of this category. All abstractions can be undone in one step using the *restore...* commands. The abstraction category is the API equivalent of the Abstraction menu of the user interface.

menu(abstraction(hide_subgraph(node_ids)))

Hides the subgraphs of the nodes specified by *node_ids*. This abstraction removes all children of a specified node from the drawing that can only be reached by starting at the specified node. The hiding process stops as soon as there is a node with an additional parent which is not hidden at the same time. Nodes on which this abstraction is applied are visualized with an icon showing a pair of scissors. *node_ids* of already hidden nodes can be used here, but of course this will have no effect.

menu(abstraction(show_subgraph(node_ids)))

Undoes a subgraph abstraction for the nodes specified by *node_ids*. *node_ids* of already shown nodes can be used here, but of course this will have no effect.

menu(abstraction(restore_all_subgraphs))

Undoes all subgraph abstractions in the graph in one step.

menu(abstraction(hide_edges(node_ids)))

Hides the edges of the nodes specified by *node_ids*. This abstraction removes all incoming and outgoing edges of a specified node from the drawing. Nodes on which this abstraction is applied are visualized with a surrounding rectangular border. *node_ids* of already hidden nodes can be used here, but of course this will have no effect.

menu(abstraction(show_edges(node_ids)))

Undoes an edge abstraction for the nodes specified by *node_ids*. *node_ids* of already shown nodes can be used here, but of course this will have no effect.

menu(abstraction(restore_all_edges))

Undoes all edge abstractions in the graph in one step.

Layout Menu Category:

The commands of the layout category are manipulating the layout of a graph. There are commands to start the layout algorithm (i.e. edge crossing minimization), to move nodes closer together, and to set the orientation of a graph layout. The layout category is the API equivalent of the Layout menu of the user interface.

menu(layout(improve_all))

This command starts the layout algorithm to improve the visualization quality for the whole graph by minimizing edge crossings and edge bends. Processing this algorithm may take a lot of time, depending on the size of the graph and the accuracy of the layout algorithm which can be set by using command set(layout_accuracy(...)). This is the reason for not applying the layout algorithm initially after loading a graph. The user or the application has to do this on its own.

menu(layout(improve_edges))

Same as command menu(layout(improve_all)) above, but this command optimizes only the dummy nodes (i.e. it calculates a better edge routing. The order of the regular nodes on the levels is left unchanged.

menu(layout(improve_selected))

Same as command menu(layout(improve_all)) above, but this command optimizes only the currently selected regular nodes and the dummy nodes of the graph. The order of all other nodes is left unchanged.

menu(layout(improve_nodes(node_ids)))

Same as command menu(layout(improve_selected)) above, but this command uses the nodes supplied in the parameter instead of the selected nodes.

menu(layout(improve_spacing))

Calculates a new layout by moving the nodes closer together if possible. Usually, one will use this operation after hiding subgraphs with command menu(abstraction(hide_subgraph(...))) to get a more compact graph layout.

menu(layout(orientation(orientation)))

Sets the layout orientation of a graph, according to parameter *orientation*. The default orientation is *top-down*, where root nodes of the graph are drawn at the top and edges are leading downwards. The other three directions *bottom-up*, *left-to-right* and *right-to-left* are supported as well.

---

daVinci V2.1 Online Documentation - Page update: June 15, 1998

daVinci API Definition - Application Menu Commands

On this page, API commands of the app_menu category are described. This category covers all commands to attach dialog elements (i.e. menus and icons) of an application to the *daVinci* user interface. The commands of this category are enclosed by keyword **app_menu(...)**. Click here to get an overview of all API commands.

Why attaching Menus and Icons to the daVinci *User Interface?*

By attaching its own menus and icons to *daVinci*, the application can offer the opportunity to start operations, processed by the application itself, from the *daVinci* user interface. So, the user has the same kind of menu access to both the visualization operations executed by *daVinci* and to operations that modify the graph structure and need to be executed by the connected application.

For example, an application might add a *"Delete Node"* menu. If the user selects this menu in *daVinci's* user interface, the application will be informed about this event by receiving a menu_selection(...) answer from the API. With respect to the current node selection, which has been propagated by the API with a prior node_selections_labels(...) answer, the application can remove the specific node(s) in its own data structure. Afterwards, the application has to send one of the commands of the graph category back to the API to reflect the modifications in *daVinci* by updating the graph visualization. With the trivial tasks of this example, *daVinci* has been extended by the functionality of a simple graph editor without the need to provide the necessary code inside the visualization system. This way, *daVinci* is as flexible and generic as possible to be extended by any kind of functionality, depending on the behaviour of an application connected to the API.

Background of Application Menus and Icons

An application is able to attach menus in the *daVinci* user interface at any time by sending command app_menu(create_menus(...)). It is even possible to add more menus later in a session, although this is not a recommended practice with respect to human computer interaction guidelines. *daVinci* will always attach these application menus in the Edit menu hierarchy, on the assumption that these operations should usually deal with modification of the graph structure. Each time the user selects an entry of a menu provided by the application, answer menu_selection(...) will be sent back by the API to inform about this event. Afterwards, the application can perform any appropriated task to handle the event, e.g. manipulating the own data and sending commands to the API.

Application icons, added by using command app_menu(create_icons(...)), are attached on a separate plane of the icon bar. The other (default) plane of the icon bar, visible at startup time, contains *daVinci's* icons which are shortcuts to frequently used menu operations. The user can switch between the *daVinci* plane and the application plane by clicking on the arrow at the lower side of each plane. As soon as the application attaches new icons, the application plane and the arrow become visible

automatically. Each time the user selects an icon provided by the application, answer icon_selection(...) will be sent back by the API to inform about this event. Afterwards, the application can perform any appropriated task to handle the event, e.g. manipulating the own data and sending commands to the API.

In multi-graph mode, each context (i.e. graph) has its own menus and icons. To guarantee a consistent user interface, application menus and icons cannot be removed later. If particular menu entries or icons are not applicable in a current situation (e.g. with respect to the current node or edge selection in *daVinci*), the application should deactivate them by using commands app_menu(activate_menus(...)) and app_menu(activate_icons(...)). Both activation commands enable the specified menus (or icons) and automatically disable all remaining application menus (or icons).

A closer Integration of I/O by interposing the File Menu Events

The application is not able to attach its menus in *daVinci* to other places than Edit. This might be inappropriated for I/O operations, for example if the applications would like to have a menu to save its own data structure. With respect to most style guides, such an operation should usually be available in the File menu. After connecting an application to the API, *daVinci* will disable most of the entries in the File menu to aviod situations where the user is able to load a new graph without permission of the application.

If desired, the application can use command app_menu(control_file_events) to take control on all entries of *daVinci's* File menu, except of File/Connect Application and File/Disconnect Application. After doing so, the application will receive events with the menu_selection(...) answer as soon as any entry of the File menu is selected by the user. After loosing control on the File menu, *daVinci* will not perform any operation if these menu entries are selected by the user. The application has to decide on its own what to do. To simulate *daVinci's* behaviour on these events, read the recommended actions below. The entries of *daVinci's* File menu have special predefined menu_ids (with prefix #%) which are used for the menu_selection(...) answer in case of selection. These reserved *menu_ids* cannot be used as identifier for application menus:

- "#%new" -> user has selected menu File/New
- "#%open" -> user has selected menu File/Open...
- "#%save" -> user has selected menu File/Save
- "#%saveas" -> user has selected menu File/Save As...
- "#%print" -> user has selected menu File/Print...
- "#%close" -> user has selected menu File/Close
- "#%exit" -> user has selected menu File/Exit

These *menu_ids* can also be used with command app_menu(activate_menus(...)) to enable resp. disable the corresponding entries in the File menu. Of course, an unapplicable File menu events can simply be ignored by the application, but usually it has to deactivate menu entries that are not applicable at a time.

Note: Each time command *app_menu(activate_menus(...))* is sent to the API, all the other menus, not listed in the command's parameter *menu_ids*, will be automatically deactivated, even in the File menu (but of course only if the application has taken control on the File menu). So, an application programmer must always consider the entries of the File menu with each menu activation, because otherwise the user might not be able to exit *daVinci*, if menu File/Exit has been disabled

automatically.

Very important: After taking control on the File menu events, the application is responsible for closing a context and quitting *daVinci*, if the user selects the corresponding menu (e.g. File/Close resp. File/Exit). In this case, the answers *menu_selection("#%close")* resp. *menu_selection("#%exit")* are sent to the application which should usually be handled by performing the recommended actions listed below. The user will not be able to close windows or exit *daVinci*, if the application ignores these events.

Recommended Actions on File Events

If the application controls *daVinci's* File menu (read above) and the user selects an entry of this menu, then *daVinci* does not perform any operation. Instead, the event is forwarded to the application by using the API answer menu_selection(...). At this time, the application has to decide what to do. For example, if menu File/New is selected, the application will receive answer *menu_selection("#%new")*. An appropriate behaviour to handle this event might be to clear the applications internal graph data structure and send an empty graph back to *daVinci*.

If the application would like to simulate *daVinci's* behaviour on file events, it can do so by sending the following API commands back to *daVinci*:

- Menu event *File/New*, i.e. receiving answer *menu_selection("#%new")*:
  Clear the graph in *daVinci* by sending command menu(file(new)).

- Menu event *File/Open...*, i.e. receiving answer *menu_selection("#%open")*:
  Use an "open" file browser with command window(file_browser(...)) and wait for answer browser_answer(...). Load the choosen *daVinci* file by using command menu(file(open_graph(...))) or menu(file(open_status(...))) with the filename returned by the file browser.

- Menu event *File/Save*, i.e. receiving answer *menu_selection("#%save")*:
  Save the status of the graph currently visualized in *daVinci* with command menu(file(save_status(...))).

- Menu event *File/Save As...*, i.e. receiving answer *menu_selection("#%saveas")*:
  Use a "save" file browser with command window(file_browser(...)) and wait for answer browser_answer(...). Save the graph or status by using command menu(file(save_graph(...))) or menu(file(save_status(...))) with the filename returned by the file browser.

- Menu event *File/Print...*, i.e. receiving answer *menu_selection("#%print")*:
  Open *daVinci's* print dialog window with command menu(file(print))

- Menu event *File/Close*, i.e. receiving answer *menu_selection("#%close")*:
  Close the current context (i.e. graph) by sending command menu(file(close)). This menu should only be available in multi-graph mode. Never close a context in single-graph mode!

- Menu event *File/Exit*, i.e. receiving answer *menu_selection("#%exit")*:
  Terminate *daVinci* with command menu(file(exit)) and exit your application, too.

Note: In multi-graph mode, you also need to consider the context of the menu event, sent to the application in a preluding context(...) answer. The current context must be set using command multi(set_context(...)) (to the *context_id* returned by the event) before sending any commands.

Otherwise, your reaction may affect other graphs than the one where the File menu event actually happened.

Documentation of the Category Commands:

app_menu(create_menus(menus))

Creates application menus, specified by list *menus*, to be attached to *daVinci's* Edit menu. Read above for details about application menus. The menus are build hierarchically, so they may contain submenus to realize cascades. The menus will appear in the same order in the user interface as the entries in the list. The following menu entry elements can be used for list *menus*:

- **menu_entry(**menu_id**,**menu_label**)**
  A single menu entry, whereby *menu_id* is the identifier sent back to the application with answer menu_selection(...) if the user selects this entry. Further, the *menu_id* can be used with command app_menu(activate_menus(...)) to enable (resp. disable) this menu entry in the user interface. *menu_label* is the text of the entry which will appear in the menu.
  Note: This entry does not have a Motif mnemonic. It is intended for backward compatibility with the V1.x API. Please use the mnemonic version *menu_entry_mne* (directly below) instead of this one to support keyboard operation.

  Popup-menu: For popup menus, this is the recommended element to insert an entry into a popup menu, because popup menus can't have mnemonics. The entry is always active in the popup menu.

- **menu_entry_mne(**menu_id**,**menu_label**,**menu_mne**,**menu_mod**,**menu_acc**)**
  A single menu entry, whereby *menu_id* is the identifier sent back to the application with answer menu_selection(...) if the user selects this entry. Further, the *menu_id* can be used with command app_menu(activate_menus(...)) to enable (resp. disable) this menu entry in the user interface. *menu_label* is the text of the entry which will appear in the menu. *menu_mne* is a string with only one character, specifying the Motif mnemonic for this entry. This character will be underlined in the menu for keyboard usage. The last two parameters are used to specify an optional Motif keyboard accelerator for frequently used menu entries. *menu_mod* is the modifier key, either **alt**, **shift**, **control**, **meta** or **none** (do not quote, e.g. *no* **"alt"**!). *menu_acc* is the accelerator key which is a *X-Window keysym* without the prefix "XK_" (keysyms are listed in file *include/keysymdef.h* in your X11 directory). If you do not want to have an accelerator for a particular menu entry, then use **none** for the *menu_mod* and **""** for the *menu_acc*. So for example, if the modifier ist **alt** and the accelerator is **"Home"**, then the user is also able to select this menu entry from the keyboard by pressing **Alt-Home** in the *daVinci* base window. The accelerator will be listed automatically in the menu.
  Note: Existing accelerators must not be overloaded, expecially not for the built-in *daVinci* accelerators (i.e. do not use **control** together with O,S,P,C,X,N,L,T,F,A,I)!

  Popup-menu: For popup menus, this element is not recommended, because popup menus can't have mnemonics. But if you use it, it behaves as *menu_entry* above.

- **menu_entry_disabled(**menu_id**,**menu_label**)**
  This element behaves as *menu_entry* above for application menus. It should be used only for popup-menus.

Popup-menu: For popup menus, this is the element to insert a disabled entry into a popup menu. The entry is always deactivated in the popup menu. We have introduced it, to give the application the chance to keep popup menus consistent for all nodes and edges, because popup menus can't be deactivated. If the application want's to activate the menu entry, it has to exchange the popup menu by changing the node or edge attribute or the corresponding visualization rule.

- **submenu_entry(**menu_id**,**menu_label**,**menus**)**
  A submenu (i.e. cascade) which contains another list of menu entries, specified by parameter *menus*. Identifier *menu_id* can be used with command app_menu(activate_menus(...)) to enable (resp. disable) this submenu. Submenus can be opened by the user, but not selected, so their *menu_id* will never be sent back to the application with a menu selection event. *menu_label* is the text of the submenu which will appear in the menu.
  Note: This submenu does not have a Motif mnemonic. It is intended for backward compatibility with the V1.x API. Please use the mnemonic version *submenu_entry_mne* (directly below) instead of this one to support keyboard operation.

  Popup-menu: For popup menus, this is the recommended element to insert a submenu into a popup menu, because popup menus can't have mnemonics. The entry is always active in the popup menu.

- **submenu_entry_mne(**menu_id**,**menu_label**,**menus**,**menu_mne**)**
  A submenu (i.e. cascade) which contains another list of menu entries, specified by parameter *menus*. Identifier *menu_id* can be used with command app_menu(activate_menus(...)) to enable (resp. disable) this submenu. Submenus can be opened by the user, but not selected, so their *menu_id* will never be sent back to the application with a menu selection event. *menu_label* is the text of the submenu which will appear in the menu. *menu_mne* is a string with only one character, specifying the Motif mnemonic for this entry. This character will be underlined in the menu for keyboard usage.

  Popup-menu: For popup menus, this element is not recommended, because popup menus can't have mnemonics. But if you use it, it behaves as *submenu_entry* above.

- **submenu_entry_disabled(**menu_id**,**menu_label**,**menus**)**
  This element behaves as *submenu_entry* above for application menus. It should be used only for popup-menus.

  Popup-menu: For popup menus, this is the element to insert a disabled submenu into a popup menu. The submenu is always deactivated in the popup menu. We have introduced it, to give the application the chance to keep popup menus consistent for all nodes and edges, because popup menus can't be deactivated. If the application want's to activate the submenu, it has to exchange the popup menu by changing the node or edge attribute or the corresponding visualization rule.

- **blank**
  A separator line in the menu to group particular menu entries.

app_menu(create_icons(icons))

Creates application icons, specified by list *icons*, to be attached to the application plane of the icon bar. Read above for details about application icons. The icons will appear in the same order in the user interface as the entries in the list. The following icon entry elements can be used for list *icons*:

- **icon_entry**(icon_id,filename,string)
  A single icon entry, whereny *icon_id* is the identifier sent back to the application with answer icon_selection(...) if the user selects this icon. Further, the *icon_id* can be used with command app_menu(activate_icons(...)) to enable (resp. disable) this icon in the user interface. *filename* is the file with the bitmap of the icon. The file must be a X-Window Bitmap file (*.xbm* format) of size 18x18 pixel. Relative *filenames* are searched in the directories specified by environment variable $DAVINCI_ICONDIR, whereas absolute *filenames* are loaded directly. The third parameter *string* should be a short description of the semantic of the icon which will be displayed in the footer area of the *daVinci* base window as soon as the mouse-pointer is over the icon.

- **blank**
  A separator gap in the application plane of the icon bar to group particular icon entries.

app_menu(activate_menus(menu_ids))

Activates the application menu entries specified by *menu_ids*. All the other entries, previously created by sending command app_menu(create_menus(...)) and not member of list *menu_ids*, are deactivated at the same time, such that they cannot be selected by the user. So, the application only need to enable the menus that should be available at a time without deactivating the other ones. Read above for details about application menus.
Note: If the application has taken control on *daVinci's* File menu (read above for details), then it also has to activate/deactivate the menu entries of the File menu, which have a special *menu_id*.

app_menu(activate_icons(icon_ids))

Activates the application icons specified by *icon_ids*. All other icons, previously created by sending command app_menu(create_icons(...)) and not member of list *icon_ids*, are deactivated at the same time, such that they cannot be selected by the user. So, the application only need to enable the icons that should be available at a time without deactivating the other ones. Read above for details about application icons.

app_menu(control_file_events)

By sending this command, the application declares to take control on the events of *daVinci's* File menu. Afterwards, entries of the File menu can be enabled (resp. disabled) with command app_menu(activate_menus(...)) just like the other application menus. The entries of the File menu have a special *menu_id*, listed above. In this case, *daVinci* will not perform any operation, if an entry of the File menu is selected by the user. Instead, an event is sent by the API with answer menu_selection(...) to let the application decide what to do. Read above for details about controlling the File menu.

---

daVinci API Definition - Set Commands

On this page, API commands of the set category are described. This category covers all commands to set options in *daVinci*. The commands of this category are enclosed by keyword **set(...)**. Click here to get an overview of all API commands.

set(layout_accuracy(integer))

Sets the accurary of the graph layout algorithm and thereby the processing cost. The integer parameter of the command need to be in the range between 1 and 5, whereby higher values mean better layout quality. The default value is 3. A higher value may result in fewer edge crossings after using the layout algorithm, but the processing time will rise substantially. So, manipulate this option with caution, especially for very large graphs This setting operation can also be done by the user with menu Options/Layout Algorithm....

set(keep_nodes_at_levels(boolean))

(Only available in *daVinci* V2.0.2 or higher.) Specifies whether or not nodes should be kept at their current level when the graph layout algorithm is performed. The default value is *true*. Using argument *false* will place the nodes back to their initial level with the next run of the layout algorithm (e.g. by using menu Layout/Improve All). This setting operation can also be done by the user with menu Options/Layout Algorithm....

set(font_size(integer))

Sets the font size which is used for drawing the text of nodes in the graph. The size *integer* is specified in 'point' (pt), the default value is 12. The given *integer* need to be a supported font size, either 6, 8, 10, 12, 14, 18, 24 or 36. Setting the font size to a smaller value is useful to reduce the scale of a graph without loosing the text of nodes. This setting operation can also be done by the user with menu Options/Layout Dimensions....

set(gap_width(integer))

Sets the minimal gap distance between nodes at the same level to *integer*. The value need to be in the range between 4 and 300, the default value is 20. This setting operation can also be done by the user with menu Options/Layout Dimensions....

set(gap_height(integer))

Setting the minimal gap distance between the levels to *integer*. The value need to be in the range between 4 and 300, the default value is 60. This setting operation can also be done by the user with menu Options/Layout Dimensions....

set(multi_edge_gap(integer))

Sets the distance used for drawing multi-edges to *integer*. The value need to be in the range between 2 and 40, the default value is 10. Multi-edges are used if there is more than one edge between two nodes. The single edges of a multi-edge are spreaded by using this gap. This setting operation can also be done by the user with menu Options/Layout Dimensions....

set(self_edge_radius(integer))

Sets the radius used for drawing self-edges to *integer*. The value need to be in the range between 8 and 60, the default value is 16. Self-edges are used if the parent and child node of an edge are the same. Self-edges are drawn at the left or lower side of a node using a circle with a radius specified here. This setting operation can also be done by the user with menu Options/Layout Dimensions....

set(scrolling_on_selection(boolean))

(Only available in *daVinci* V2.0.2 or higher.) This command is used to switch the auto-selection behaviour. If auto-selection is on (*true*) and the user selects a node in a survey view, then *daVinci* automatically scrolls to the same node in the corresponding detail view (details about multi-view can be found in the concept document). This behaviour can be switched off using parameter *false*, the default value is *true*. This setting operation can also be done by the user with menu Options/General Settings....

set(animation_speed(integer))

Sets the speed used for animation to *integer*. The value need to be either in the range between 10 and 100 or 0 (0 is a special case to switch the animation off). The default value is 25. Animation is used to focus a new node, for example to show the result of command menu(navigation(find(...))) or menu(navigation(navigator(...))). The smaller the value, the slower the movement from an old point of focus to a new one. This setting operation can also be done by the user with menu Options/General Settings....

set(no_cache(boolean))

Sets the flag for controlling the pixmap cache to *boolean*. The default value is false. Set this to true if you have problems with memory on your computer.

For efficency reasons, *daVinci* will never draw a graph visualization directly on the canvas of a window. Instead, the canvas is partitioned into a grid of squares. For each element of the grid, there is a separate pixmap which is only drawn by need if the particualar grid element becomes visible, e.g. after using scrolling. This way, *daVinci* is even able to handle very large graphs which would usually require a hugh canvas. Already drawn pixmaps are kept in a cache to avoid another redrawing if they will become visible again after they have been invisible before.

You can observe this feature in *daVinci*: The first time you scroll into a new area of the graph, you will notice a delay which is needed to draw the portion of the graph on the pixmap of the corresponding grid element. Afterwards, the pixmap will be copied to the window canvas. The next time you scroll into this area, you will not notice any delay. The pixmap was cached and only need to be copied into the canvas without any redrawing.

Unfortunately, pixmap caching requires a lot of memory, especially for large graphs. This may crash *daVinci* with a message *X Error: BadAlloc* on systems with a small amount of RAM as soon as no more memory is available. You can try to work without the cache in these situations. The disadvantage of turning off caching is a reduced scrolling performance. This setting operation can also be done by the user with menu Options/General Settings....

set(rules_first(boolean))

Controls the attribute evaluation order for the use of visualization definitions as described in the concepts document. This attribute can be set only with this API command. Set this to *false* to achieve the default behaviour.

---

daVinci V2.1 Online Documentation - Page update: June 15, 1998

daVinci API Definition - Window Commands

On this page, API commands of the window category are described. This category covers all commands to manipulate windows in *daVinci*. The commands of this category are enclosed by keyword **window(...)**. Click here to get an overview of all API commands.

window(title(string))

Sets the title of the base window to *string*. A string "*daVinci* V2.1 - " is a permanent prefix of all window titles and cannot be changed. In multi-view and multi-window mode, this operation will affect all views of the current context.

window(show_message(string))

Displays a message *string* in the left footer area of the base window. This command should be used to show temporal information such as "saving done". For permanent messages, command window(show_status(...)) should be used instead. In multi-view and multi-window mode, this operation will affect all views of the current context.

window(show_status(string))

Displays a message *string* in the right footer area of the base window. This command should be used to show permanent informations such as a current status. For temporal messages, command window(show_message(...)) should be used instead. In multi-view and multi-window mode, this operation will affect all views of the current context.

window(position(integer,integer))

Sets the x/y position of the base window to the given values. The first *integer* is the new x-coordinate, the second is the new y-coordinate of the window. Do not use negative values or values larger than the screen. In multi-view mode, this operation will only affect the master view of the current context.

window(size(integer,integer))

Sets the width and height of the base window to the given values. The first *integer* is the new width, the second is the new height of the window. Do not use negative values or values larger than the screen. In multi-view mode, this operation will only affect the master view of the current context.

window(raise)

Raises the base window to the front such that it lays on top of all the other windows. In multi-view mode, this operation will only affect the master view of the current context.

window(iconify)

Iconifies the base window. The inverse command is window(deiconify). In multi-view and multi-window mode, this operation will affect all views of the current context.

window(deiconify)

Deiconifies the base window. The inverse command is window(iconify). In multi-view and multi-window mode, this operation will affect all views of the current context.

window(activate)

Enables user input in *daVinci* again after this has been disabled before with command window(deactivate). In multi-graph and multi-view mode, this operation will affect all opened windows of *daVinci* and not only the views of the current context.

window(deactivate)

Disables **all** user input in *daVinci*. This will deactivate the whole system, such that the user is not able to do anything in *daVinci*, and no answers are sent to the application until it will be enabled again with command window(activate). In multi-graph and multi-view mode, this operation will affect all opened windows of *daVinci* and not only the views of the current context.

window(file_browser(boolean,string,string,string,string,btypes,boolean))

-> Returns answer browser_answer(...)!

By using this command, the application has the opportunity to use *daVinci's* file browser for its own purpose. This file browser does not start any operation in *daVinci* after the user has selected a file. Instead, the selected file (and type) is returned to the application by sending answer *browser_answer(...)*. Due to the manifold features of the *daVinci* file browser, the application need to specify several parameters which have the following semantic:

1. *boolean* (first one)
   Flag for opening a file. Should be *true* if the browser is used to open a file, *false* otherwise. There are two differences between an opening and saving browser: The opening browser does not ask the user with a confirm window if the selected file exist (which is done by the saving browser to avoid accidental overwriting) and the opening browser does not allow to choose a not existing file (whereas the saving browser does).

2. *string* (first one)
   Window title of the file browser.

3. *string* (second one)
   Short label text of the left activation button that is responsible for closing the browser. Should be usually "Open", "Save" or something like that.

4. *string* (third one)
   Initial directory of the browser. If empty (i.e. ""), the home directory of the user is used as default.

5. *string* (forth one)
   Initial file of the browser in the initial directory, may be empty (i.e. ""). The entry of the selection field of the browser at start-up will be composed of the initial directory and the initial file.

6. *btypes*
   This list specifies the types (i.e. file patterns), available in the **Types** menu of the browser. The list need to have at least one entry to retrieve the default type of the browser which is always the first type of the list. An empty list is not allowed and will force an error. The elements of list *btypes* are three-tuples with the following syntax:

   bt(string**,**string**,**string**)**

   1. string: A plain text description of the type, e.g. "PostScript Files"
   2. string: The pattern for the type in shell notation, e.g. "*.ps"
   3. string: An optional postfix for the window header, e.g. "PostScript"

   The header postfix (3. string) will be added to the window title of the browser each time the particular type is selected by the user. So, if the window has title "Save" and there are two types with postfix "X" and "Y", then the title of the browser will be either "Save X" or "Save Y". For example, here is a valid *btypes* parameter for a browser with a C pattern and a generic pattern for all files:

   [bt("C Files","*.c","C"),bt("All Files,"*","")]

7. *boolean* (second one)
   Flag to show hidden files initially. Should be usually *false* to filter the hidden files. The user will be able to modify this setting at any time by using the **Show hidden** check button of the browser.

The application file browser is modal, in opposite to the file browsers used by *daVinci*. So, after opening an application file browser, all user interaction in *daVinci* will be disabled until the browser is closed by the user. At this time, the application will be informed about the selected file with API answer browser_answer(string,string), whereby the first string is the absolute file name of the selected file and the second string is the plain text of the selected type (i.e. the first string of the corresponding *bt(...)* three-tuple). Both strings are empty if the user has canceled the file browser.

---

daVinci API Definition - Tcl/Tk Commands

On this page, API commands of the tcl category are described. This category covers all commands to use the *daVinci* Tcl/Tk interface. The commands of this category are enclosed by keyword **tcl(...)**. Click here to get an overview of all API commands.

The daVinci *Tcl/Tk Interface*

General informations about Tcl/Tk are provided by SunScript.

The *daVinci* Tcl/Tk Interface can be used by application programmers to create their own application specific windows. The *daVinci* Tcl/Tk interface is able to execute any Tcl 7.4/Tk 4.0 script in the same way as a *wish* shell would do. So, the application program does not need to start its own *wish* (or another interpreter) to use Tcl/Tk, because "*daVinci* is the *wish*". This way, the application does not need to deal with another X-Window toolkit to realize its graphical front-end. All it has to do to implement a graphical user interface is sending API commands and Tcl/Tk scripts to the *daVinci* API.

Tcl/Tk scripts can either be sent to *daVinci* with API command tcl(eval(string)), whereby *string* is the Tcl/Tk script, or loaded from file with API command tcl(eval_file(filename)). Command *tcl(eval(string))*, i.e. sending the complete Tcl/Tk script via pipe, is only useful for small scripts or Tcl/Tk function calls, because the *string* argument has to fulfill the conventions for API strings, especially double quote characters *inside* the string have to be escaped (i.e. **\"**) and newline or return characters are not allowed (use **\n** instead). This restriction is not required for Tcl/Tk scripts loaded from file with API command tcl(eval_file(filename)). For both commands, the result of the Tcl/Tk script will be immediately returned to the application with answer tcl_answer(...) as soon as the script is executed.

A possible scenario for using the Tcl/Tk interface is to save a script of Tcl/Tk functions in a file and execute it with API command *tcl(eval_file(filename))* at startup time of *daVinci*. During runtime, the particular functions can be called by sending API command *tcl(eval(string))*.

The daVinci *command in Tcl*

The *daVinci* Tcl/Tk interpreter, accessible by the API, has all the features of a usual Tcl/Tk interpreter, e.g. the one in the *wish* shell, plus one additional object (or function) *daVinci* which supports the two methods (or parameters) *tcl_answer* and *command*, see below. The *daVinci* Tcl-object can be used in a Tcl/Tk script to send strings back to the application via the API or to send API commands directly to *daVinci* from inside the script.

Note: *daVinci's* user interface and graph visualization cannot be manipulated with the Tcl/Tk interface, so *daVinci* is not a Tcl object in the common sense. The *daVinci* system can only be accessed by using the commands of the API.

Tcl command daVinci tcl_answer string

This Tcl command can be used in a Tcl/Tk script to send an arbitrary string back to the application. *daVinci* will redirect the argument *string* via the API to the application which will receive an answer tcl_answer(...) from the API. This feature is useful to implement callbacks.

Example:

tcl(eval("button .b -text Hello -command {daVinci tcl_answer Pressed} ; pack .b"))

By sending this API command (containing a Tcl/Tk script) to *daVinci*, a window with a button *Hello* will appear. The API will send answer *tcl_answer("Pressed")* back to the application as soon as the user presses the button.

Note: It is currently not possible to use Tcl delimiters (i.e. '{' and '}') inside the *string* argument of command *tcl_answer*. E.g. *"...-command {daVinci tcl_answer {Pressed}}"* is **not** allowed in the example above, although it is a valid Tcl expression.

Tcl command daVinci command api-command

This Tcl command can be used in a Tcl/Tk script to send API commands directly to *daVinci* without detour through the application. The result of the Tcl command in the Tcl/Tk script is the answer of the executed API command, so the script may need to forward this result to the application (e.g. by using Tcl command *daVinci tcl_answer {...}*), if the application need to be informed about this event.

Example:

tcl(eval("button .b -text Clear -command {daVinci command menu(file(new))} ; pack .b"))

By sending this API command (containing a Tcl/Tk script) to *daVinci*, a window with a button *Clear* will appear. After pressing this button, the API command menu(file(new)) will be executed by *daVinci* which will clear a probably loaded graph.

Documentation of the Category Commands:

tcl(eval(string))

-> Returns answer tcl_answer(...)!

Sends a Tcl/Tk script *string* to *daVinci's* Tcl/Tk interface and waits for the result (i.e. return value) of this script which can be found in the string of the API answer *tcl_answer(...)*, sent back to the application as soon as the script has finished. *daVinci* is blocked as long as the Tcl-script is executed.
Note: The Tcl/Tk script in *string* must not contain return or newline characters and all double quotes inside the string need to be escaped (i.e. **\"**).

tcl(eval_file(filename))

-> Returns answer tcl_answer(...)!

Same as API command tcl(eval(...)) above, but the Tcl/Tk script is read from the specified file rather than taken from the string parameter. So, the Tcl/Tk script does not need to be transfered via pipe. Use this API command for large scripts or if you cannot fulfill the API convention for strings (i.e. no return/newline characters and no unescaped double quotes in the Tcl/Tk script).

Overview | Intro | Reference | Tutorial | Questions & Answers | New Features | Index

Overview -> Reference -> API Reference -> Commands of Category Special

daVinci API Definition - Special Commands

On this page, API commands of the special category are described. This category covers all commands to select or scroll to particular nodes. The commands of this category are enclosed by keyword **special(...)**. Click here to get an overview of all API commands.

special(select_nodes(node_ids))

-> Returns answer node_selections_labels(...)!

This command can be used for setting the selection in a graph to the nodes specified by *node_ids*. The new selection is returned by the node_selections_labels(...) answer of the command.

special(select_edge(edge_id))

-> Returns answer edge_selection_label(...)!

This command can be used for setting the selection in a graph to the edge specified by *edge_id*. The new selection is returned by the edge_selection_label(...) answer of the command.

special(focus_node(node_id))

Scrolls to the node specified by *node_id*. Usually, it is a better practice to use the next command special(focus_node_animated(...)) which takes advantage of animation to scroll smoothly to the new point of focus. In multi-view mode, this operation will only affect the master view of the current context, where in multi-window mode it will affect the current window.

special(focus_node_animated(node_id))

Scrolls to the node specified by *node_id* with animation, i.e. the window with the graph visualization will scroll smoothly to the new point of focus. The speed of animation can be set using command set(animation_speed(...)). If animation is not required, command special(focus_node(...)) can be used instead. In multi-view mode, this operation will only affect the master view of the current context, where in multiwindow mode it will affect the current window.

special(show_url(string))

Displays a particular HTML-page in Netscape Navigator specified by the URL given in the string parameter. If this string contains a ':/' substring then the URL is expected to be global and passed to Netscape as is. Otherwise, if no substring ':/' occurs, the URL is expected to be a relative filename in the *docs/* directory of the *daVinci* distribution. In the second case, *daVinci* adds the content of the **Online Documentation:** field of the general settings dialog as prefix to the given string before passing the URL to Netscape.

To use this command, *daVinci* has to be configured to be able to access the online documentation according to these rules. This is usually true with the initial installation of the *daVinci* system.

---

daVinci V2.1 Online Documentation - Page update: June 15, 1998

daVinci API Definition - Visual Commands

On this page, API commands of the visual category are described. This category covers all commands to work with visualization definitions. The commands of this category are enclosed by keyword **visual(...)**. Click here to get an overview of all API commands.

Further details about visualization definitions can be found in the concepts document.

Documentation of the Category Commands:

visual(new_rules(visual_rules))

The parameter of the command is a list of visualization rules with two different kinds of list elements, which make up the visualization definition. Both types of elements can be mixed in arbitrary order:

- **nr**(type,attributes)
  Define the visualization rule (attributes) for all nodes of the given type. The usable attributes are explained in the term representation reference.

- **er**(type,attributes)
  Define the visualization rule (attributes) for all edges of the given type. The usable attributes are explained in the term representation reference.

This command first removes all defined visualization rules and than inserts the newly defined rules.

You can overwrite the default values of *daVinci* if you supply rules for type "".

visual(add_rules(visual_rules))

This command has the same behaviour as the command *visual(new_rules(...))*, but it doesn't remove the visualization rules before adding the new ones. This means, that rules for a type already present are overwritten and all other rules are added to the visualization definition.

You can overwrite the default values of *daVinci* if you supply rules for type "".

---

daVinci V2.1 Online Documentation - Page update: June 15, 1998

daVinci API Definition - Drag and Drop Commands

On this page, API commands of the drag and drop category are described. This category covers all commands to implement drag and drop features in an application. The commands of this category are enclosed by keyword **drag_and_drop(...)**. Click here to get an overview of all API commands.

Possible Drag and Drop scenarios

Create a new node

In this case, the user presses the button BTransfer somewhere in the graph area and no node is below the mouse pointer. The API sends the answer create_node(...) and the application should react with the command *drag_and_drop(new_node_at_coord(...))* to insert a new node. *daVinci* will try to insert the new node at the coordinate where the mouse button was released.

Create a new node connected to a parent node by a new edge

In this case, the user presses the button BTransfer somewhere in the graph area and the mouse pointer is over a node. The mouse button is released somewhere but **not** over a node. The API sends the answer create_node_end_edge(...) and the application should react with the command *drag_and_drop(new_edge_and_node_at_coord(...))* to insert a new node and connect it as a child node to the parent node received in the event. *daVinci* will try to insert the new node at the coordinate where the mouse button was released. While dragging the mouse pointer with the button pressed, *daVinci* draws an abstract edge from the first selected node to the position of the mouse pointer. If the mouse pointer is over a node, this node is inverted to inform the user about the secondary selection.

Create a new edge

In this case, the user presses the button BTransfer somewhere in the graph area and the mouse pointer is over a node. The mouse button is released somewhere over a node. The API sends the answer create_edge(...) and the application should react with the command *graph(update(...))* to insert one new edge connecting the two nodes received in the event, whereas the first node is the parent and the second node is the child. While dragging the mouse pointer with the button pressed, *daVinci* draws an abstract edge from the first selected node to the position of the mouse pointer. If the mouse pointer is over a node, this node is inverted to inform the user about the secondary selection. If the mouse pointer is released over the node where the mouse button was pressed and dragged in the meantime, both nodes received in the event are the same. The application could in this case ignore the event or if this is allowed insert a self-edge.

Drop a node into a window

In this case, the user presses the button BTransfer somewhere in the graph area together with the Shift-Key and the mouse pointer is over a node. The mouse button is released somewhere even in another *daVinci* base-window but **not** over a node. The API sends the answer the event drop_node(...) and the application should react with some command appropriate for this action, i.e. *drag_and_drop(new_node_at_coord(...))* to insert a copy of the node. *daVinci* will try to insert the new node at the coordinate where the mouse button was released. While dragging the mouse pointer with the button pressed, *daVinci* changes its mouse pointer to a target, but the

selected node is not moved together with the mouse-pointer, because we can't get the appropriate events from Tcl/Tk. The first selected node is drawn inverted to mark the node where the dragging started. The event contains the node_id,context_id and window_id of the node where the dragging started. The second node_id is "" in this case, because the node wasn't dropped on another node.

Drop a node onto another node

In this case, the user presses the button BTransfer somewhere in the graph area together with the Shift-Key and the mouse pointer is over a node. The mouse button is released somewhere even in another *daVinci* base-window over a node. The API sends the answer drop_node(...) and the application should react with some command appropriate for this action, i.e. *drag_and_drop(new_edge_and_node_at_coord(...))* to insert a copy of the node as a child of the node the first node was dropped onto. While dragging the mouse pointer with the button pressed, *daVinci* changes its mouse pointer to a target, but the selected node is not moved together with the mouse-pointer, because we can't get the appropriate events from Tcl/Tk. The first selected node is drawn inverted to mark the node where the dragging started. The event contains the node_id,context_id and window_id of the node where the dragging started. The second node_id contains the identifier of the node over which the mouse button was released. The context_id and window_id of this node can be retrieved from the context(...) or context_window(...) answer immediately before the *drop_node(...)* answer.

Further details about drag and drop can be found in the concepts document.

Documentation of the Category Commands:

drag_and_drop(dragging_on)

With this command an application can switch on the Drag and Drop mode, in which case the 2D-scrolling with mouse button BTransfer is switched off. After sending this command, the application has to be prepared to receive the API events:

- create_node(...)
- create_node_and_edge(...)
- create_edge(...)
- drop_node(...)

drag_and_drop(dragging_off)

With this command an application can switch off the Drag and Drop mode (**default**), in which case the 2D-scrolling with mouse button BTransfer is switched on.

drag_and_drop(new_node_at_coord(node_update))

This command is the recommended action after receiving the API event create_node(...). If the *daVinci* API sends the answer, the coordinate at which the mouse button was released, is stored and reused, if this command is received, to insert the new node defined in the parameter *node_update*. Therefore only a new_node(...) update is allowed in the parameter.

daVinci tries to insert the new node at the stored coordinate, but it maybe not possible to do so, because of layout restrictions. Please have a look at the concepts document to read more about graph layout.

If no create_node(...), create_node_and_edge(...) or drop_node(...) event was received before and therefore no coordinate was stored, the command behaves like a graph(update(...)) command with only one node update specified.

drag_and_drop(new_edge_and_node_at_coord(node_update,edge_update))

This command is the recommended action after receiving the API event create_node_and_edge(...). The *daVinci* API sends the answer, the coordinate at which the mouse button was released, is stored and reused, if this command is received, to insert the new node defined in this parameter *node_update*. Therefore only a new_node(...) update is allowed in the parameter. The new edge specified in the parameter *edge_update* should connect the new node to the node that was received with the event, because the user expects that a new child node of this node is inserted. Therefore only a new_edge(...) or new_edge_behind(...) update should be used in this parameter, although this is not checked by *daVinci*.

daVinci tries to insert the new node at the stored coordinate, but it maybe not possible to do so, because of layout restrictions. Please have a look at the concepts document to read more about graph layout.

If no create_node(...), create_node_and_edge(...) or drop_node(...) event was received before and therefore no coordinate was stored, the command behaves like a graph(update(...)) command with only one node and one edge update specified.

---

daVinci API Definition - Overview of Answers

Help about syntax

```
daVinci_answer ::=              /*Answers from the API. Details.*/
    ok                          /*Positive confirmer. Details.*/
  | communication_error(string) /*Negative confirmer. Details*/
  | node_selections_labels(node_ids) /*Labels of sel. nodes. Details.*/
  | node_double_click           /*Sel. node double-clicked. Details.*/
  | edge_selection_label(edge_id) /*Label of sel. edge. Details.*/
  | edge_selection_labels(node_id,node_id)/*Dito, parent/child. Details.*/
  | edge_double_click           /*Sel. edge double-clicked. Details.*/
  | menu_selection(menu_id)     /*ID of selected menu. Details*/
  | icon_selection(icon_id)     /*ID of selected icon. Details*/
  | popup_selection_node(node_id,menu_id) /*ID of selected popup menu for node. Details.*/
  | popup_selection_edge(edge_id,menu_id) /*ID of selected popup menu for edge. Details.*/
  | context(context_id)         /*Other context (graph). Details. */
  | context_window(context_id,window_id)  /*Other context (graph) and window. Details.*/
  | open_window                 /*User opened a window. Details.*/
  | close_window(window_id)     /*ID of closed window. Details. */
  | tcl_answer(string)          /*Answer from Tcl script. Details.*/
  | browser_answer(string,string) /*File browser result. Details*/
  | create_node                 /*Drop new node. Details.*/
  | create_node_and_edge(node_id) /*Drop node and edge for parent ID. Details.*/
  | create_edge(node_id,node_id) /*Drop edge between nodes. Details.*/
  | drop_node(node_id,context_id,window_id,node_id)
                                /*Drop node from context and window on node. Details.*/
  | disconnect                  /*Termination request. Details*/
  | close                       /*Context (graph) closed. Details.*/
  | quit.                       /*daVinci terminated. Details*/
```

---

daVinci V2.1 Online Documentation - Page update: June 15, 1998

daVinci API Definition - Answers

On this page, all answers of the *daVinci* API are described. These answers are sent back to an application connected with the API. An overview of all answers is also available.

Answers at Initialization Time:

daVinci can be connected with an application program either by starting *daVinci* from the application process with option -pipe or by starting the application from the *daVinci* process using menu File/Connect Application... or option -startappl, refer to page How to Connect an Application for details. After connecting *daVinci* with the application, an initial answer ok is sent to the application to signal preparedness for receiving commands. An application should not start sending commands to *daVinci* before receiving the inital *ok* as the first answer from the API.

Answers for confirming API Commands:

In general, commands sent to the API will be confirmed with an answer ok after the command was executed successfully. All commands that do not return *ok* are listed below. If an error occured while executing a command, *daVinci* will send answer communication_error(...) instead of *ok*. Usually, this exception should not happen in a running system, if the application meets all rules and restrictions of the API. The application can find the reason for the error in the string parameter of answer *communication_error(...)*.

Commands that do not return the *ok* answer are:

1. menu(file(close)): Returns close
2. menu(navigation(select_parents(...))): Returns node_selections_labels(...)
3. menu(navigation(select_siblings(...))): Returns node_selections_labels(...)
4. menu(navigation(select_children(...))): Returns node_selections_labels(...)
5. menu(navigation(navigator(...))): Returns node_selections_labels(...)
6. menu(navigation(find(...))): Returns node_selections_labels(...)
7. window(file_browser(...)): Returns browser_answer(...)
8. tcl(eval(...)): Returns tcl_answer(...)
9. tcl(eval_file(...)): Returns tcl_answer(...)
10. special(select_nodes(...)): Returns node_selections_labels(...)
11. special(select_edge(...)): Returns edge_selection_label(...)

In multi-window mode, the following two commands don't answer open_window instead of ok:

1. menu(view(open_new_view))
2. menu(view(open_survey_view))

Answers for Event Notification:

The rest of the answers may occur at any time. These are event notifications which are triggered by the user. A *daVinci* application must be prepared to receive these answers at any time, even in the time interval between sending a command and receiving the corresponding confirmation answer. The following events may be sent by the API:

1. node_selections_labels(...): Some nodes have been selected in the graph.
2. node_double_click: The previously selected node has been clicked twice.
3. edge_selection_label(...): An edge has been selected in the graph.
4. edge_selection_labels(...): Obsolete answer for compatibility to *daVinci* V1.x API.
5. edge_double_click: The previously selected edge has been clicked twice.
6. menu_selection(...): An application menu entry of the Edit menu has been selected.
7. icon_selection(...): An application icon of the icon bar has been selected.
8. popup_selection_node(...): A popup-menu for a node has been selected.
9. popup_selection_edge(...): A popup-menu for an edge has been selected.
10. tcl_answer(...): Return value from a Tcl/Tk script sent to *daVinci*.
11. create_node(...): Create a new node in Drag and Drop mode.
12. create_node_and_edge(...): Create a new node and edge starting at a parent in Drag and Drop mode.
13. create_edge(...): Create a new edge between two nodes in Drag and Drop mode.
14. drop_node(...): Drop a node on another node in Drag and Drop mode.
15. disconnect: The user has selected menu File/Disconnect Application.
16. close: In multi-graph mode, the current context has been closed.
17. quit: *daVinci* has terminated.

18. open_window: User opened a window in multi-window mode.
19. close_window(...): User closed a window in multi-window mode.

VERY IMPORTANT: Due to the strange effects of asynchronous communication, an event must also be expected by the application after sending a command to the API and before receiving the corresponding confirmation answer! *daVinci* guarantees that events are not propagated during execution of an API command, but if the event occured during transfer time of the command (i.e. *after* it is sent by the application and *before* it is received by the API), then the event answer may interfere and will arive at the application side in the time interval after sending the command and before receiving the corresponding confirmation answer. So never expect that the confirmation answer will come back directly after sending an API command. Your application should accept events at any time!

Answers in Multi-Graph Mode:

If an application is leaving the default single-graph mode to enter the multi-graph mode, then the behaviour for confirming commands and sending event notification will slightly change. In multi-graph mode, **all** answers from *daVinci* **always** have a preluding answer context(...) to specify the context (graph) where the event happened. So, in multi-graph mode, an application will always get two answers instead of one: The first one is the context and the second one is the event itself.

Answers in Multi-Window Mode:

If an application is leaving the default single-graph mode to enter the multi-window mode by setting the command-line option -window_api, then the behaviour for confirming commands and sending event notification will slightly change. In multi-window mode, **all** answers from *daVinci* **always** have a preluding answer context_window(...) to specify the context (graph) and window where the event happened. So, in multi-window mode, an application will always get two answers instead of one: The first one is the context/window and the second one is the event itself. Additionally the answers of the commands menu(view(open_new_view)) and menu(view(open_survey_view)) change to open_window instead of ok.

Documentation of all API Answers:

ok

This answer is returned by the API to confirm the successful execution of a command sent to the API. Exceptions: The commands listed above do not return *ok*, they return another answer instead. If the execution of a command has failed, then answer communication_error(...) is sent instead of *ok*.

For synchronization, *ok* is also sent to the application at initialization time of the API after the application is successfully connected with daVinci.

communication_error(string)

This answer will be returned by the API after the execution of an API command has failed. In the *string* parameter, the application can find an error message describing the reason for the failure. Usually, such an exception cannot happen if the application meets all rules and restrictions of the commands sent to the API.

node_selections_labels(node_ids)

Answer *node_selections_labels(...)* is returned by the API to inform the application about selection of nodes in the graph. This is done to confirm the following API commands:

- menu(navigation(select_parents(...)))
- menu(navigation(select_siblings(...)))
- menu(navigation(select_children(...)))
- menu(navigation(navigator(...)))
- menu(navigation(find(...)))
- special(select_nodes(...))

Further, answer *node_selections_labels(...)* will be sent to the application as an event after the user has selected a set of nodes in the graph visualization. In the parameter *node_ids* of the answer, the application can find the identifiers of all nodes which are currently selected. These identifiers are specified in the term representation of the graph. **Note**: Many nodes may be selected in *daVinci* at a time which is reflected by the list of identifiers returned by this answer. The list will be empty if the user has deselected all nodes. After loading a graph, no node will be selected by default.

node_double_click

This answer will be returned by the API after the user has selected a node twice within a specific short time interval. Note that the answer has no parameter because it refers to the last node selection. So, if the user clicks on a node with *node_id* "something" twice, then after the first click *daVinci* will send answer node_selections_labels(["something"]). If the second mouse click will be detected within the time interval, then *daVinci* will send answer *node_double_click* to inform that the last selection was a double click.

edge_selection_label(edge_id)

Answer *edge_selection_label(...)* is returned by the API to inform the application about selection of a single edge in the graph. This is done to confirm the following API command:

- special(select_edge(...))

Further, answer *edge_selections_label(...)* will be sent to the application as an event after the user has selected an edge in the graph visualization. In the parameter *edge_id* of the answer, the application can find the identifier of the edge which is currently selected. This identifier is specified in the term representation of the graph. **Note**: Unlike nodes, there can only be *one* edge selected at a time in *daVinci*, so the parameter of this answer is not a list.

edge_selection_labels(node_id,node_id)

This is an obsolete alternative of answer edge_selection_label(...) to inform an application about selection of an edge, too. This answer is present in the V2.x API for backward compatibility. The first *node_id* parameter of the answer is the parent and the second *node_id* is the child node of the currently selected edge. Read the description of answer edge_selection_label(...) above for details about edge selection.

Note: This is an obsolete answer which is only used if the selected edge has no specified edge_id in the graph. In previous V1.x releases of *daVinci*, *edge_id's* were recommended, but not required for the term representation of a graph. In *daVinci* V2.x, an *edge_id* is now mandatory for all edges. This obsolete answer will be supported for some time to ensure backward compatibility.

edge_double_click

This answer will be returned by the API after the user has selected an edge twice within a specific short time interval. Note that the answer has no parameter because it refers to the last edge selection. So, if the user clicks on an edge with *edge_id* "something" twice, then after the first click *daVinci* will send answer edge_selection_label(["something"]). If the second mouse click will be detected within the time interval, then *daVinci* will send answer *edge_double_click* to inform that the last selection was a double click.

menu_selection(menu_id)

This answer is returned by the API to inform the application about selection of an application menu. The application program is able to attach its own menu entries to *daVinci's* Edit menu by using the API command app_menu(create_menus(...)). The *menu_id*, found in the parameter of this answer, is the identifier specified at creation time of the selected menu.

If the application would like to have a closer integration of I/O by taking control on the events of the *daVinci* File menu events, then it is able to do so by using command app_menu(control_file_events). In this case, answer *menu_selection(...)* is also used to inform the application about selection events in the File menu. Click here to get more information about controlling events of the File menu and the recommended actions to handle these events.

icon_selection(icon_id)

This answer is returned by the API to inform the application about selection of an application icon. The application program is able to attach its own icons to the application plane of *daVinci's* icon bar by using the API command app_menu(create_icons(...)). The *icon_id*, found in the parameter of this answer, is the identifier specified at creation time of the selected icon.

popup_selection_node(node_id,menu_id)

This answer is returned by the API to inform the application about selection of a popup menu of a node. The application program is able to attach its own menu entries to each node by setting the popup-menu attribute. The *node_id*, found in the parameter of this answer, is the node identifier of the node over which the right mouse button was pressed and the menu was selected. The *menu_id*, found in the parameter of this answer, is the identifier specified at creation time of the selected menu.

popup_selection_edge(edge_id,menu_id)

This answer is returned by the API to inform the application about selection of a popup menu of an edge. The application program is able to attach its own menu entries to each edge by setting the popup-menu attribute. The *edge_id*, found in the parameter of this answer, is the edge identifier of the edge over which the right mouse button was pressed and the menu was selected. The *menu_id*, found in the parameter of this answer, is the identifier specified at creation time of the selected menu.

context(context_id)

This answer is returned by the API only in multi-graph mode to specify the context (i.e. graph) where an event happened. The event answer itself will be sent afterwards. The *context_id*, found in the parameter of answer *context(...)*, is the context identifier which has been assigned at creation time (by using commands multi(new_context) or multi(open_context(...))). If your application does not want to deal with more than one graph at a time, i.e. when it does not enter multi-graph mode by opening a context, it will never receive the *context* answers.

In multi-graph mode, each answer of the API sent back to the application has always a preluding *context(...)* answer to specify the context where the event happened. So in fact *daVinci* sends two answers at a time. Read above for details.

context_window(context_id,window_id)

This answer is returned by the API only in multi-window mode to specify the context (i.e. graph) where an event happened. The event answer itself will be sent afterwards. The *context_id*, found in the parameter of answer *context_window(...)*, is the context identifier which has been assigned at creation time (by using commands multi(new_context) or multi(open_context(...))). The *window_id* found in the parameter of answer *context_window(...)*, is the window identifier, which is automatically assigned by *daVinci* if the user opens a new base window. You'll be informed about the window identifier the

first time, if you get the answer open_window. If your application does not want to deal with more than one graph at a time, i.e. when it does not enter multi-window mode by opening a context, it will never receive the *context_window* answers.

In multi-window mode, each answer of the API sent back to the application has always a preluding *context_window(...)* answer to specify the context and window where the event happened. So in fact *daVinci* sends two answers at a time. Read above for details.

open_window

This answer is returned by the API only in multi-window mode to inform the application about a new open base-window for a context (graph). This is done to confirm the following API commands:

- menu(view(open_new_view))
- menu(view(open_survey_view))

Further this answer will be sent to the application as an event after the user has opened a new base-window. The window_id which is automatically assigned to the new window, can be taken from the preluding context_window(...) answer which is sent prior to each answer in multi-window mode.

close_window(window_id)

This answer is returned by the API only in multi-window mode as an event after the user has closed a base-window. The window_id parameter is the identifier of the closed window. After closing a window, the *window_id* cannot be used any further. Afterwards, the application needs to open a new window or specify one of the remaining windows to be the new current window. All other commands will be ignored, because after closing, the current window will be undefined as long as a new one is not specified.

tcl_answer(string)

This answer is returned by the API to pass results or informations from Tcl/Tk scripts back to the application. These scripts can be sent to the *daVinci* Tcl/Tk interface by using the following API commands:

- tcl(eval(...))
- tcl(eval_file(...))

Both commands return a *tcl_answer(...)* for confirmation rather than the usual ok answer. The *string* in this answer is the return value of the Tcl/Tk-script after finishing execution. The whole *daVinci* user interface is disabled as long as a Tcl-script is executed.

On the other side, *tcl_answer(...)* may also occur as an event at any time. The *daVinci* Tcl/Tk interface offers the opportunity to send arbitrary messages back to the application via the API by using the Tcl-command daVinci tcl_answer <string> inside a Tcl/Tk script. These messages are forwarded by *daVinci* to the connected application with an API-answer *tcl_answer(...)*.

browser_answer(string,string)

This answer is returned by the API to confirm the API command window(file_browser(...)). In the two parameters of the answer, the application can find the results of the file browser, i.e. the file and type the user has choosen. The first *string* of the parameters is the selected (absolute) file name and the second *string* is the choosen file type which is one of the first elements (i.e. the plain text of the type) of the three-tuples in the btypes list used as parameter for the *window(file_browser(...))* command. Both strings of the answer are empty if the user has canceled the browser.

create_node

This answer is returned by the API to inform the application if the user released the middle mouse button over the free area and if no node was selected when the middle mouse button was pressed in drag and drop mode. The coordinate at which the mouse button is released is stored internally in *daVinci* and is reused if the API command drag and drop(new node at coord(...)) or drag and drop(new node and edge at coord(...)) is received. The recommended behaviour to react on this event is to insert a new node at the mouse position by using drag and drop(new node at coord(...)).

create_node_and_edge(node_id)

This answer is returned by the API to inform the application if the user released the middle mouse button over the free area and if a node was selected when the middle mouse button was pressed in drag and drop mode. The coordinate at which the mouse button is released is stored internally in *daVinci* and is reused if the API command drag and drop(new node at coord(...)) or drag and drop(new node and edge at coord(...)) is received. The parameter *node_id* of the event is the identifier of the node which was selected when the middle mouse button was pressed. The recommended behaviour to react on this event is to insert a new node at the mouse position that is connected to the existing node supplied in the parameter with a new edge by using drag and drop(new node and edge at coord(...)).

create_edge(node_id,node_id)

This answer is returned by the API to inform the application if the user released the middle mouse button over a node and if a node was selected when the middle mouse button was pressed in drag and drop mode. The first parameter *node_id* of the event is the identifier of the node which was selected when the middle mouse button was pressed. The second parameter *node_id* of the event is the identifier of the node which was selected when the middle mouse button was released. The recommended behaviour to react on this event is to insert a new edge between the two nodes by using graph(update(...)).

drop_node(node_id,context_id,window_id,node_id)

This answer is returned by the API if the user released the middle mouse button over a node or the free area and if a node was selected when the middle mouse button was pressed together with the Shift-Key in drag and drop mode. The coordinate at which the mouse button is released is stored internally in *daVinci* and is reused if the API command drag and drop(new node at coord(...)) or drag and drop(new node and edge at coord(...)) is received. The first parameter *node_id* of the event is the identifier of the node which was selected when the middle mouse button was pressed. The second parameter *context_id* of the event is the identifier of the context (graph) of the node specified

in the first parameter. The third parameter *window_id* of the event is the identifier of the window of the node specified in the first parameter (this parameter is only usefull in multi-window mode). The fourth parameter *node_id* of the event is the identifier of the node which was selected when the middle mouse button was released or "" if it was released over the free area. The context_id (and the window_id) of the node specified in the fourth parameter can be taken from the preluding context(...) answer which is sent prior to each answer in multi-graph mode or from the preluding context_window(...) answer which is sent prior to each answer in multi-window mode. There is no recommended behaviour to react on this event.

disconnect

This answer will be returned by the API after the user has selected menu File/Disconnect. When this event happens, the application is obliged to terminate. *daVinci* V2.x is reentrant (in opposite to previous V1.x releases), so the *daVinci* process will not die after the application has terminated. With this feature, the user has the opportunity to connect another application after disconnecting the old one.

Disconnecting is **only** available if *daVinci* has established the pipe connection to the application before, e.g. by using menu File/Connect Application... or option -startappl. On the other side, if *daVinci* has been started by the application program, then disconnecting is not possible, so the *disconnect* answer will never be sent in this situation.

close

The answer *close* will be returned by the API only in multi-graph or multi-window mode to inform the application that the current context (i.e. graph) has been closed. This can either be done by the application itself using API command menu(file(close)) or by the user with menu File/Close. After closing a context, the corresponding graph and all of its view windows will be removed and its *context_id* cannot be used any further. Afterwards, the application need to open a new context or specify one of the remaining contexts to be the new current context. All other commands will be ignored, because after closing, the current context will be undefined as long as a new one is not specified.

So, answer *close* is either sent as result to confirm command *menu(file(close))* or as an event at any time if the user has selected menu *File/Close* in the last open view window of a context. If it is not appropriated to allow that the user is able to close a context on his own, then the application can control the events of the File menu to interpose before the context is actually closed.

The context_id of the closed context can be taken from the preluding context(...) answer which is sent prior to each answer in multi-graph mode or from the preluding context_window(...) answer which is sent prior to each answer in multi-window mode. (read above for details).

quit

This answer is sent by the API on termination, e.g. if the user has selected menu File/Exit. At this time, the *daVinci* process is going to die and the application should stop sending any further command to the API. If it is not appropriated to allow that the user is able to exit *daVinci* on his own, then the application can control the events of the File menu to interpose before *daVinci* is actually terminated.

daVinci V2.1 Online Documentation - Page update: June 15, 1998

daVinci PostScript Interface

In *daVinci*, the user is able to print and save a graph visualization in PostScript format with menu File/Print.... The PostScript code, generated by *daVinci*, can be printed on a PostScript printer as well as included in another document with appropriated software.

The output, generated by *daVinci*, is PostScript Level-1 code, compliant to Adobe's Document Structuring Conventions (DSC). The guidelines for document exchange in *Encapsulated PostScript Format* (EPSF, often called EPS) are considered, so you can load a *daVinci* PostScript graph in programs that support EPSF import (e.g. FrameMaker). The optional PostScript preview part (*Encapsulated PostScript Interchange*, EPSI) is not generated, so after importing a *daVinci* PostScript file into another application, you will not see the graph on the screen, but of course it will appear on paper after printing. There are third party packages to add the optional preview part to any PostScript file, for example Doug Crabill's *pstoepsi*, available from ftp://ftp.cs.purdue.edu/pub/dgc/pstoepsi-1.2.shar.Z.

daVinci usually places a graph on one PostScript page, regardless of the size of the graph visualization. So multi-page output for getting huge posters is not directly supported by *daVinci*. To split a large *daVinci* PostScript file on several small pages (to stick them together), you can use third party tools such as C.P. Lai's *ps_poster* tool, available on the *daVinci* ftp server: ftp://ftp.tzi.de/tzi/biss/daVinci/tools_from_3rd_party/ps_poster.tar.gz.

Notes about the daVinci *PostScript interface:*

1. If the PostScript scale of a graph, selected by the user in the File/Print... dialog window, is too large, then the image of the graph will not be completely visible on the page after printing. If you intend to send the PostScript file to a printer, choose the correct page size supported by the printer and press the **Fit to Page** button afterwards to avoid such problems.
2. The coordinates of nodes and edges in PostScript are taken from the screen representation of the graph. PostScript fonts and X-Window screen fonts may vary in character width, so for the PostScript graph it is possible that the text of a node is quite smaller than the node itself, in opposite to the screen representation. In rare cases, the text may overlap the boundary of a node.
3. Screen font "Helvetica" is always replaced by the PostScript font "HelveticaNarrow" to avoid problems with overlapping.
4. The *daVinci* default font "Lucida" is not a popular font in PostScript. So, the PostScript files, generated by *daVinci*, will always check if the current PostScript interpreter (e.g. the printer) supports "Lucida" and will use font "Helvetica" otherwise.

---

daVinci V2.1 Online Documentation - Page update: June 15, 1998

Graph Editor Application

The graph editor is an interactive tool to modify an existing graph or to build a new graph from scratch. With the editor, the user is able to create new nodes/edges, delete existing nodes/edges, modify the attributes of node/edges, and so on.

The editor is not an independent program with a separate user interface. Instead, it is a GUI-less application which communicates with *daVinci* by using the API. From a users point of view, all graph manipulation is done inside the *daVinci* base window. To access the graph manipulation functions, the editor adds some extra menus, icons and dialogs to the *daVinci* user interface. Technically, *daVinci* and the editor are two individual processes with different responsibilities: the editor has its own graph data structure, it controls and manipulates this graph depending on menu events and sends the current graph back to *daVinci* for visualization after each update. So the structure of the graph in *daVinci* is never manipulated directly, only the application (i.e. the graph editor in this case) is able to do so. You can read more about *daVinci* applications in the API reference.

Please refer to the graph editor tutorial to learn how to use the graph editor.

Starting the Graph Editor

Before editing a graph, you have to connect the editor to the *daVinci* API. This can be done in *daVinci* at runtime with menu File/Connect Application.... Simply navigate with the file browser and select the graph editor binary *grapheditor* in directory *daVinci_V2.1/grapheditor* of the *daVinci* distribution. **Note:** If you do not know where *daVinci* is installed on your computer, type *echo $DAVINCIHOME* in a UNIX shell to display the location.

Another way to connect the graph editor is by starting *daVinci* and the editor at the same time by calling *daVinci* with commandline option -graphedit.

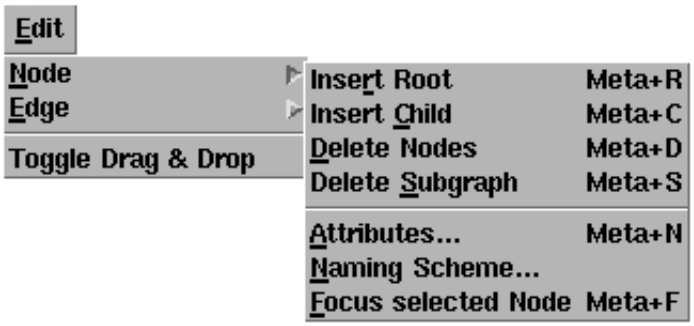
Using the Graph Editor

The graph editor application is some kind of "plug-in" to extend *daVinci's* basic functionality. To access the editing functions, the graph editor adds some extra menus to the *daVinci* user interface which are attached to the Edit menu. There you can find menu operations to manipulate the nodes and edges of a currently loaded graph, see below. For more convenience, the editor adds some icons to the *daVinci* icon bar to provide short-cuts for the most frequently used editing operations, see below.

The graph editor is able to edit several graphs at a time by using the multi-graph mode of *daVinci*. Please choose menu File/New to open a window for a new graph.

For a basic introduction, you can also read the section about the graph editor in the tutorial.

Menu Edit/Node



This menu contains operations to manipulate the nodes of a graph, i.e. inserting nodes, deleting nodes and modifying node attributes.

Note: Menus can also be used mouseless with keyboard commands.

- **Insert Root**
  This operation inserts a new root node in the graph, i.e. a node which has no parent or child node. The new node will have the attributes which are currently set in the attribute dialog (available after selecting menu Edit/Node/Attributes...). The text of the new node is determined by the current content of the naming scheme dialog (available after selecting menu Edit/Node/Naming Scheme...).

- **Insert Child**
  This operation inserts a new child node in the graph, i.e. a node which is connected with its parent node(s) with a directed edge. The new node will become the child of all nodes that are currently selected, so this operations inserts the edges from the parent(s) to the new child node, too. Obviously, this menu operation is disabled when no node is selected. The new node will have the attributes which are currently set in the attribute dialog (available after selecting menu Edit/Node/Attributes...). The text of the new node is determined by the current content of the naming scheme dialog (available after selecting menu Edit/Node/Naming Scheme...).

- **Delete Nodes**
  This operation deletes the currently selected nodes and all of its edges from the direct parents and to the direct children (to avoid incomplete edges with a missing node). This menu operation is only available when one or more nodes are selected in the graph. Deletion of many nodes in one step is only available in the graph editor delivered with *daVinci* V2.0.2 or higher.

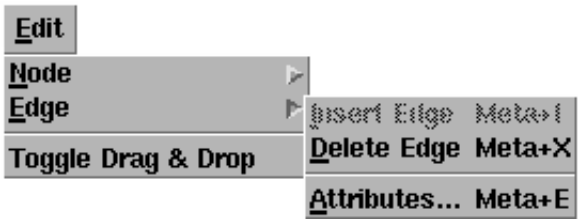
- **Delete Subgraph**
  Same as menu operation Edit/Node/Delete Node above, but removes the subgraph of the currently selected node, too. The subgraph of a node are all children (and their children, and so on) that belong exclusively to this node. So the recursive deletion stops as soon as there is a child node that has at least one parent node which is not deleted at the same time. This menu operation is only available when exactly one node is selected in the graph.

- **Attributes...**
  This operation opens the node attribute dialog to specify the attributes of the currently selected node(s). The attributes specify the visual appearance of a node in the graph visualization. This menu operation is disabled when no node is selected in the graph. You can also open this dialog by double-clicking on a node.

- **Naming Scheme...**
  This operation opens the naming scheme dialog to specify the text pattern of inserted nodes. The text of each new node is assigned automatically by using this pattern.

- **Focus selected Node**
  This operation scrolls the graph visualization to the currently selected node if the node is not already centered in the window. This menu operation is only available when exactly one node is selected in the graph.

Menu Edit/Edge



This menu contains operations to manipulate the edges of a graph, i.e. inserting edges, deleting edges and modifying edge attributes.

Note: Menus can also be used mouseless with keyboard commands.

- **Insert Edge**
  This operation inserts a new edge between two nodes, so it is available when exactly two nodes are selected in the graph (hint: use the **Shift** key to select more than one node at a time in *daVinci*). The node that was selected first will be the parent node and the node selected afterwards will be the child node of the new (directed) edge. The edge will have the attributes which are currently set in the attribute dialog (available after selecting menu Edit/Edge/Attributes...). The menu is also available, if only one node is selected, in which case it inserts a self-edge for the selected node.

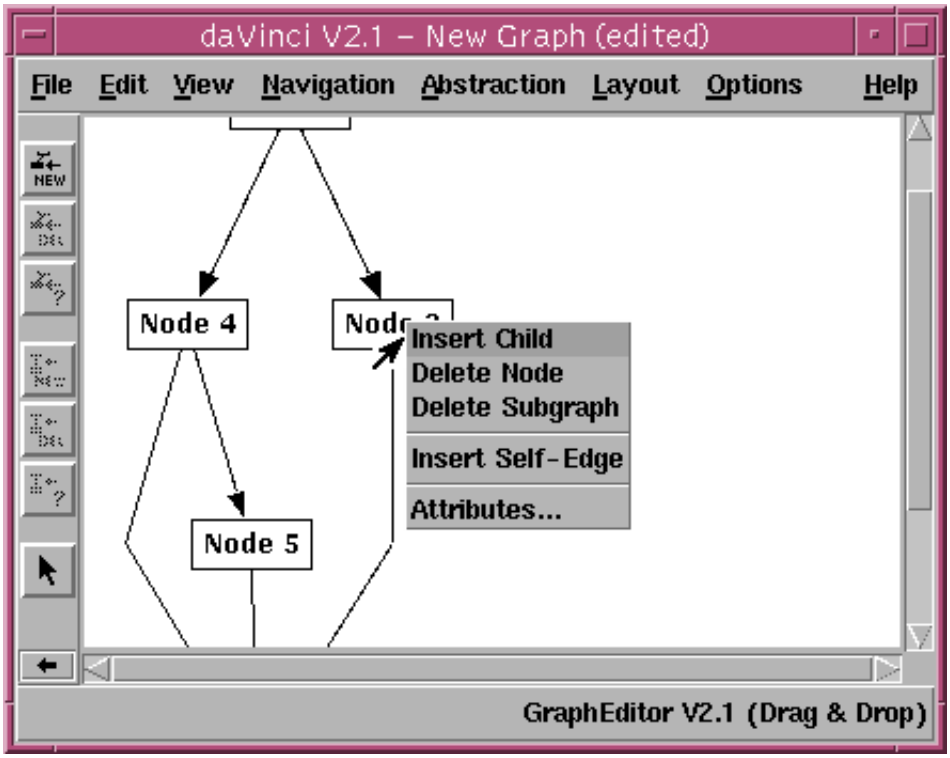
- **Delete Edge**
  This operation deletes the currently selected edge, so it is only available when exactly one edge or self-edge is selected in the graph.

- **Attributes...**
  This operation opens the edge attribute dialog to specify the attributes of the currently selected edge. The attributes specify the visual appearance of an edge in the graph visualization. This menu operation is disabled when no edge is selected in the graph. You can also open this dialog by double-clicking on an edge.

Menu Edit/Toggle Drag & Drop

This operation toggles the Drag and Drop state of the current base window. The current state is written right to the graph editor version number in the status bar of the base window.

Node Popup-Menu



This menu can be accessed, if the mouse button BMenu is pressed over a node in the grapheditor.

- **Insert Child**
  This operation inserts a new child node in the graph, i.e. a node which is connected with its parent node(s) with a directed edge. The new node will become the child of the node over which the menu mouse button was pressed, so this operation inserts the edge from the parent to the new child node, too. The new node will have the attributes which are currently set in the attribute dialog (available after selecting menu Edit/Node/Attributes...). The text of the new node is determined by the current content of the naming scheme dialog (available after selecting menu Edit/Node/Naming Scheme...).

- **Delete Node**
  This operation deletes the node over which the menu mouse button was pressed and all of its edges from the direct parents and to the direct children (to avoid incomplete edges with a missing node).

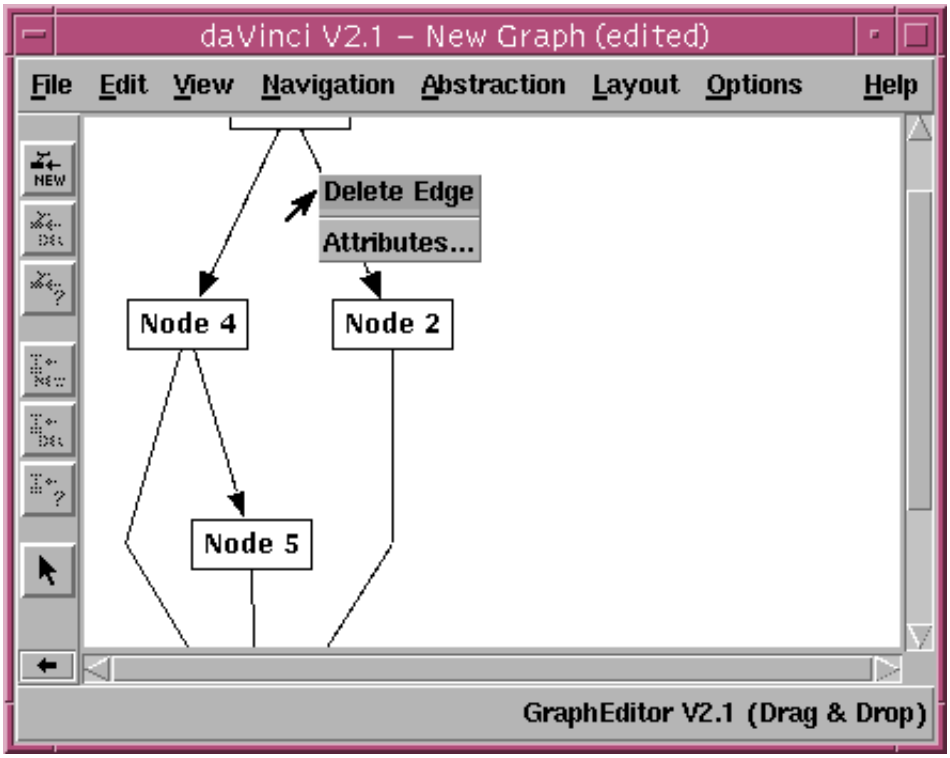
- **Delete Subgraph**
  Same as popup-menu operation Delete Node above, but removes the subgraph of the node over which the menu mouse button was pressed, too. The subgraph of a node are all children (and their children, and so on) that belong exclusively to this node. So the recursive deletion stops as soon as there is a child node that has at least one parent node which is not deleted at the same time.

- **Insert Self-Edge**
  This operation inserts a new self-edge to the node over which the menu mouse button was pressed. The edge will have the attributes which are currently set in the attribute dialog (available

after selecting menu Edit/Edge/Attributes...).

- **Attributes...**
  This operation opens the node attribute dialog to specify the attributes of the node over which the menu mouse button was pressed. The attributes specify the visual appearance of a node in the graph visualization. You can also open this dialog by double-clicking on a node.

Edge Popup-Menu



This menu can be accessed, if the mouse button BMenu is pressed over an edgee in the grapheditor.

- **Delete Edge**
  This operation deletes the edge over which the menu mouse button was pressed.

- **Attributes...**
  This operation opens the edge attribute dialog to specify the attributes of the edge over which the menu mouse button was pressed. The attributes specify the visual appearance of an edge in the graph visualization. You can also open this dialog by double-clicking on an edge.

File Operations

To load and save a graph in the graph editor, simply use *daVinci's* menus File/Open... and File/Save As.... The graph editor application takes advantage of an API feature and controls *daVinci's* File menu to do its own I/O.

Graph Editor Icons on the Icon Bar

The graph editor icons appear on the icon bar of the *daVinci* base window. To switch between the usual *daVinci* icons and the graph editor icons, simply click on the arrow (,) in the lower part of the icon bar. These icons are short-cuts for the most frequently used menu operations of the graph editor, so pressing an icon is the same as selecting the corresponding menu operation which are described above. The following list shows the available icons and the corresponding menu operations.

Edit/Node/Insert {Root | Child} (inserts root if no node is selected, child otherwise)
Edit/Node/Delete Node
Edit/Node/Attributes...
Edit/Edge/Insert Edge
Edit/Edge/Delete Edge
Edit/Edge/Attributes...
Edit/Toggle Drag & Drop

Drag and Drop in the grapheditor

The graph editor uses the Drag and Drop feature of *daVinci* to directly manipulate the graph. It is possible to

- insert a node by pressing the mouse button BTransfer somewhere in the graph area but not over a node.
- insert a new child node of a node by pressing the mouse button BTransfer over the designated parent node and releasing it somewhere over the graph area but not over a node after dragging the abstract edge to the place where you want the child node to appear.
- insert a new edge by pressing the mouse button BTransfer over the designated parent node and releasing it over the designated child node, which is drawn inverted by *daVinci* to indicate the secondary selection.
- insert a copy of a node into the same or into another graph by pressing the mouse button BTransfer together with the Shift-Key over the node you want to copy and than release the mouse button somewhere over the graph area but not over a node in the same or in another graph window.
- insert a copy of a node as a child of another node into the same or into another graph by pressing the mouse button BTransfer together with the Shift-Key over the node you want to copy and than release the mouse button somewhere over the designated parent node in the same or in another graph window.

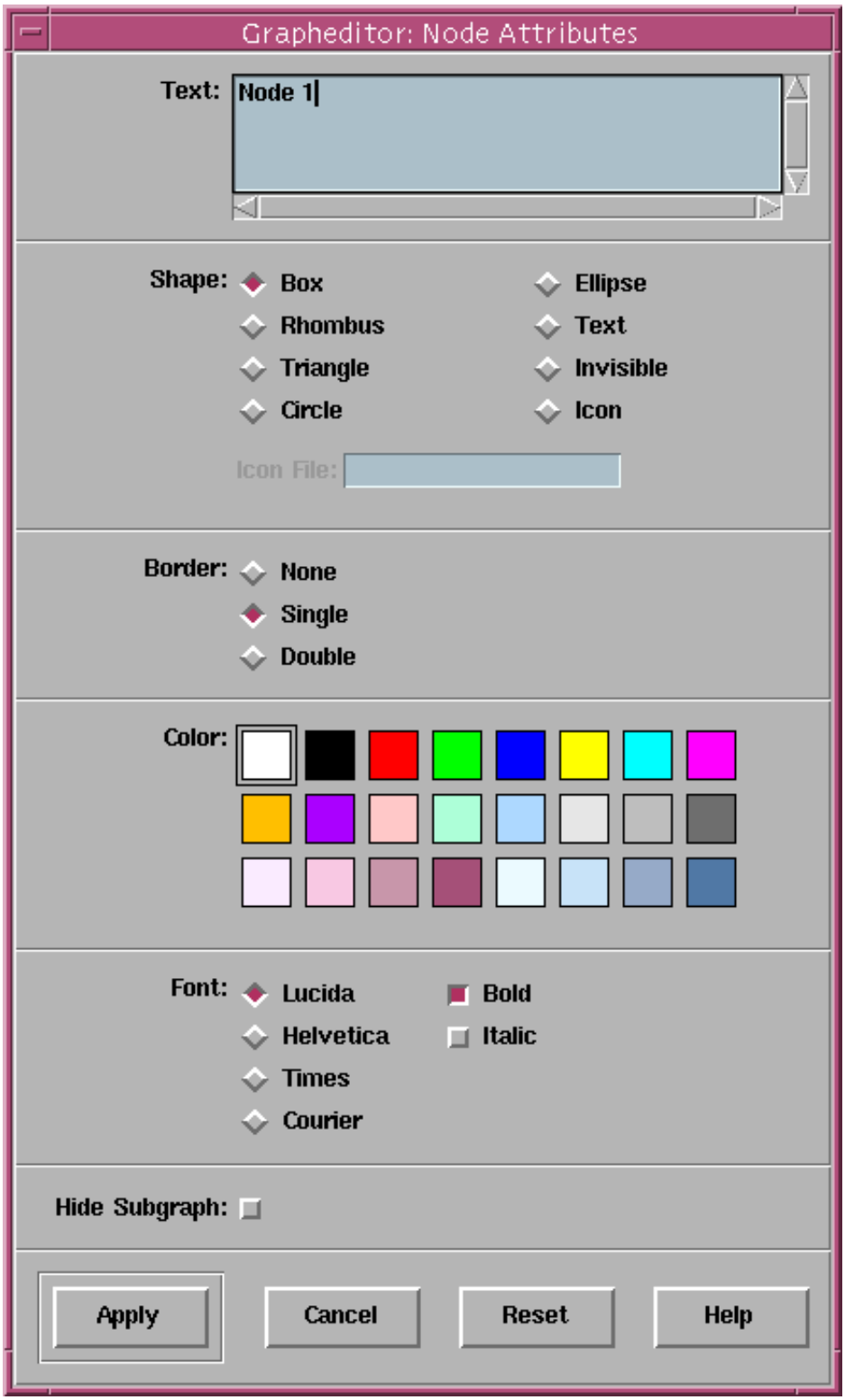
You can switch of Drag and Drop with the menu Edit/Toggle Drag & Drop or with the corresponding icon. The graph editor indicates, if drag and drop is switched on in the footer area of *daVinci*.

Observing the communication between daVinci and the grapheditor

In former graph editor versions, there was a secret option we have given away to some users of *daVinci*. This option has been removed from the graph editor in the current version. But you can still have a look at the communication by using the command-line option -log of *daVinci*.

daVinci V2.1 Online Documentation - Page update: June 15, 1998

Node Attribute Dialog Window



This graph editor dialog window is used to set the attributes of a particular node in the graph (the one which is currently selected or the one over which the popup menu has been opened). It is even possible to set the attributes of several nodes in one step if more than one node is currently selected. The dialog items in this window reflect the currently supported node attributes in *daVinci* (refer to the term representation specification for details about node attributes).

Note: This dialog window can also be used mouseless with keyboard operations.

Dialog Items of the Node Attribute Window

- **Text**
  This item specifies the text which is displayed inside a node in the graph visualization. It is possible to use multi-line text by typing return characters.

- **Shape**
  This item specifies the shape of a node (called *graphical object*). For the objects **Box** (default), **Rhombus**, **Triangle**, **Circle** and **Ellipse**, it is possible to have a double or no line border which can be switched on with the **Border** item. For the objects **Text** and **Invisible**, a border is not supported, whereas the object **Icon** can only have a single or no border.
  Object **Icon** requires to specify a X11 bitmap file (containing the monochrome image of the icon) in text entry field **Icon File**. This file has to be in one of the directories defined by environment variable $DAVINCI_ICONDIR (refer to node attribute "ICONFILE" for more information).

- **Border**
  According to the specification for **Shape** above, a node can have a double, single (default) or no border. The border is defined with this item.

- **Color**
  This item specifies the color of a node which can be choosen from a palette of 24 colors. To modify the colors in the palette, double-click on a color to open the color editor.

- **Font**
  This item specifies the font family and style used to draw the text of a node.

- **Hide Subgraph**
  This item specifies whether or not an initial subgraph abstraction should be applied to a node. Interactive subgraph abstractions (the same feature in the *daVinci* user interface) are available in menu Abstraction. **Note:** this attribute setting does not overwrite an interactive abstraction of a node in *daVinci*. So if the user hides the subgraph of a node interactively with menu Abstraction/Hide Subgraph, then the graph editor will never be able to show this subgraph again by using an attribute setting.

- **Apply**
  Push button to accept the current settings of this dialog window. Without pressing **Apply**, modifications to these options are not considered by *daVinci*. For more convenience, the window remains open after pressing the **Apply** button, so use the **Cancel** button to close the window.

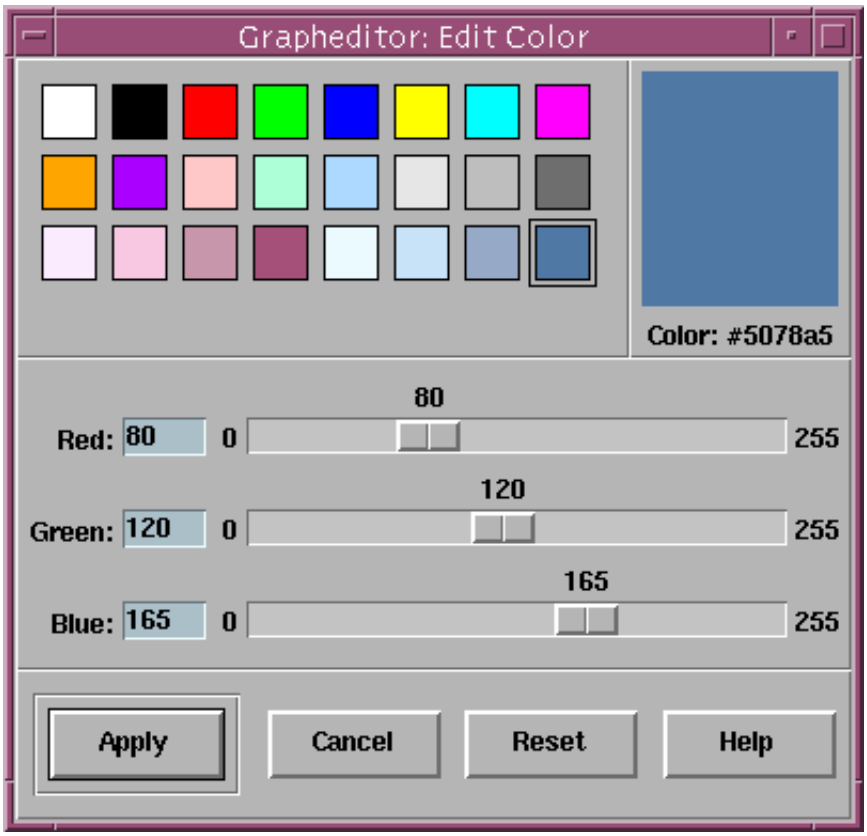
- **Cancel**
  Push button to close the node attribute dialog without performing any action.

- **Reset**
  Push button to undo any modification to these attributes unless **Apply** is not pressed. So **Reset** goes back to the current settings.

- **Help**
  Push button to start the online help system with this page.

daVinci V2.1 Online Documentation - Page update: June 15, 1998

Color Editor Window



This graph editor dialog window is used to modify the color palette to get individual color values. You can select one of the 24 colors from the palette and modify the color value by setting the red, green and blue portion with the three sliders. The first two colors (black and white) are default values, so they cannot be modified.

Note: This dialog window can also be used mouseless with keyboard operations.

Dialog Items of the Color Editor Window

- **Red**
  This item sets the red portion of the color.

- **Green**
  This item sets the green portion of the color.

- **Blue**
  This item sets the blue portion of the color.

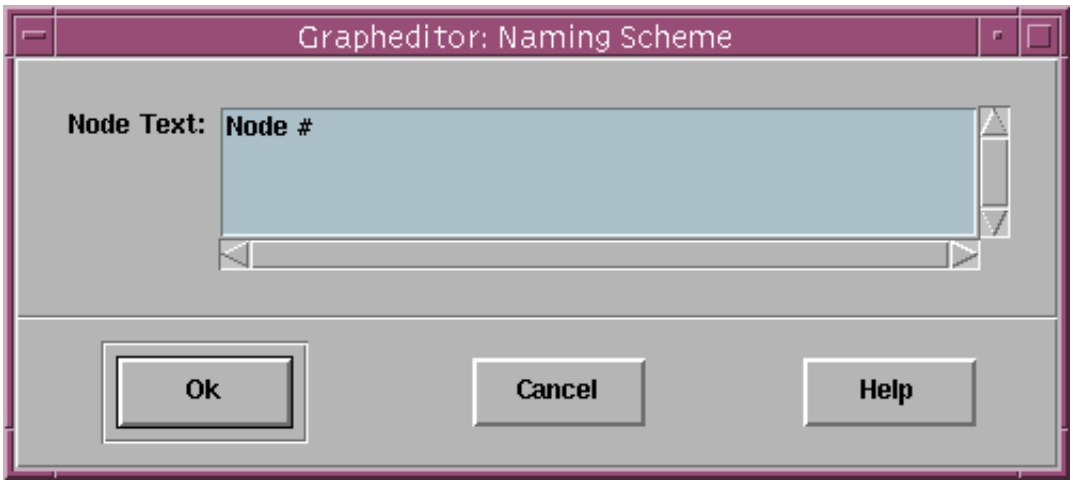
- **Apply**
  Push button to accept the current settings of this dialog window. Without pressing **Apply**, modifications to these options are not considered by *daVinci*.

- **Cancel**
  Push button to close the color editor without performing any action.

- **Reset**
  Push button to undo any modification to these attributes unless **Apply** is not pressed. So **Reset** goes back to the current settings.

- **Help**
  Push button to start the online help system with this page.

Naming Scheme Dialog Window



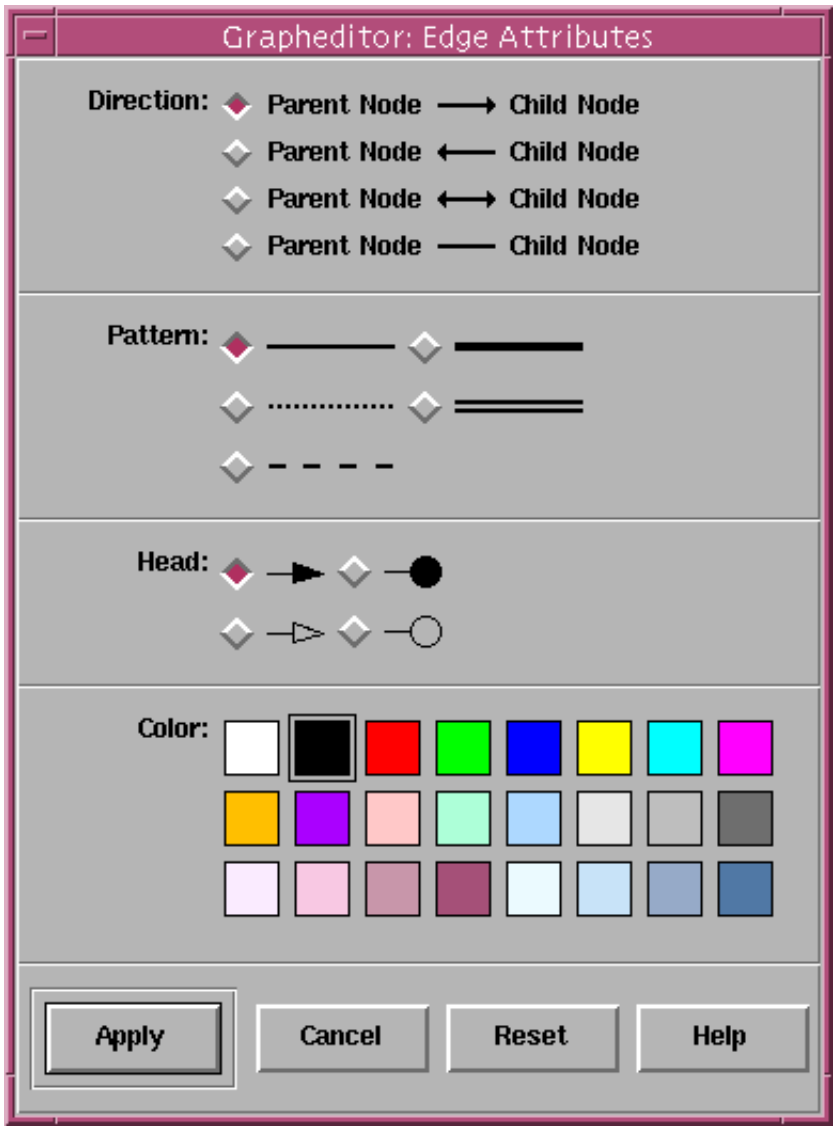
This graph editor dialog window is used to specify the naming scheme used for newly inserted nodes. Nodes are inserted into the graph with menu Edit/Node/Insert Root or Edit/Node/Insert Child, by selecting icons, the according popup-menu entries or with drag and drop. Each new node automatically gets the text of the naming scheme, whereby the placeholder character # is replaced by the current unique node number which is incremented with each new node.

Note: This dialog window can also be used mouseless with keyboard operations.

Dialog Items of the Naming Scheme Window

- **Node Text**
  This item specifies the naming scheme for the text of newly inserted nodes. It is possible to use multi-line text by typing return characters.
- **Ok**
  Push button to accept the current settings of this dialog window and close it.
- **Cancel**
  Push button to close the naming scheme dialog without performing any action.
- **Help**
  Push button to start the online help system with this page.

Edge Attribute Dialog Window



This graph editor dialog window is used to set the attributes of a particular edge in the graph (the one which is currently selected or the one over which the popup menu has been opened). The dialog items in this window reflect the currently supported edge attributes (refer to the term representation specification for details about edge attributes).

Note: This dialog window can also be used mouseless with keyboard operations.

Dialog Items of the Edge Attribute Window

- **Direction**
  This item specifies the use of arrows for an edge. By default, each edge has an arrow pointing to the child node. With this dialog item it is possible to let the edge pointing to the parent node, to both the parent and the child node or to omit the arrows for the currently selected edge. Note that the arrows are only a visual attribute of the edge and changing the edge direction does not manipulate the parent/child relationssships in the graph.

- **Pattern**
  This item specifies the pattern of the currently selected edge. An edge can be visualized with a solid (default), dotted, dashed, thick or double line.

- **Head**
  This item specifies the "arrow" head of the currently selected edge. An edge can be visualized with a filled arrow (default), arrow, filled circle or circle head.

- **Color**
  This item specifies the color of an edge which can be choosen from a palette of 24 colors. To modify the colors in the palette, double-click on a color to open the color editor.

- **Apply**
  Push button to accept the current settings of this dialog window. Without pressing **Apply**, modifications to these options are not considered by *daVinci*. For more convenience, the window remains open after pressing the **Apply** button, so use the **Cancel** button to close the window.

- **Cancel**
  Push button to close the edge attribute dialog without performing any action.

- **Reset**
  Push button to undo any modification to these attributes unless **Apply** is not pressed. So **Reset** goes back to the current settings.

- **Help**
  Push button to start the online help system with this page.

Available Tools for *daVinci*

This section describes some useful tools delivered with the *daVinci* distribution. The tools *checkterm*, *formatterm* and *flattenterm* can be found in directory *tools/* of the *daVinci* distribution.

checkterm

Syntax: `checkterm <daVinci term representation file>`

This tool can be used to check the syntax of a file containing a graph in the *daVinci* term representation format. Of course this analysis is also performed by *daVinci* when a graph is loaded, but *checkterm* is much faster and even simpler to use. Furthermore this tool can be used without a X11 window environment.

formatterm

Syntax: `formatterm [-i ident] <daVinci term representation file>`

This tool is a pretty printer for a file containing a *daVinci* graph in term representation format. *formatterm* is used to fill a term representation with blank and return characters such that it can be read easily in textual form. In the formated term representation, each node and edge of the graph is placed on a separate line by using return characters; different indents are used to clarify the level of the nodes. With option -i one can specify the indent of each level; the default value is 8 blanks. *formatterm* copies the original source file for backup by appending a percent character to the filename. The name of the destination file is the same as the original name of the source file.

It is possible to load a formatted term representation in *daVinci*, but it is not possible to send it to the API, because return characters are only allowed at the end of an API command. In this case, use tool *flattenterm* to undo the pretty printing.

flattenterm

Syntax: `flattenterm <daVinci term representation file>`

With this tool, pretty printing is undone for a file containing a *daVinci* graph in term representation format. This is simply done by removing all blank, tabulators and return characters between terminal symbols of the term representation (i.e. whitespace inside strings is left untouched). *flattenterm* copies the original source file for backup by appending a percent character to the filename. The name of the destination file is the same as the original name of the source file.

It is possible to load an unformatted term representation in *daVinci* as well as sending it to the API.

daVinci V2.1 Online Documentation - Page update: June 15, 1998

Command-Line Options and Environment Variables

Command-Line Options

The current version *daVinci* 2.1 supports 15 command-line options, described below. Further, the filename of either a term representation file or status file can be used when starting *daVinci* in a shell to load the file at initialization time.

daVinci Command-Line Syntax

```
daVinci [-display <X11 display>] [-font <normal | large | extra_large>]
        [-graphbg <X11 color>] [-graphedit] [-help] [-init <filename>]
        [-log <filename>] [-nocache] [-nopopup] [-pipe] [-speedy]
        [-startappl <directory> <filename/options>] [-verify]
        [-version] [-window_api] daVinci-file
```

Option *-display <X11 display>*

Sets the output to the specified X11 display, for example to let *daVinci* run on one computer and redirect the user interface to be displayed on another computer. X11 displays are given in a format like *somehost:0*.

Option *-font <normal | large | extra_large>*

Sets the font of the user interface to a larger size. This will also set the font size of the graph to the same size. Fonts larger than the default value *normal* are very useful for demonstrations when people should read the text of menus, dialog windows etc. from the distance.

Option *-graphbg <X11 color>*

Sets the background of the *daVinci* base window to the specified color. Valid color names can be found in file *lib/rgb.txt* in your X11 directory (probably */usr/local/X11*).

Option *-graphedit*

Connects the graph editor application at initialization time to the *daVinci* API. In the *daVinci* distribution, the graph editor application is located in directory *daVinci_V2.1/grapheditor*. If environment variable $DAVINCIHOME is correctly set to the absolute filename of directory *daVinci_V2.1*, then *daVinci* will be able to start the editor with option -graphedit (*daVinci* assumes that the filename of the editor binary is *$DAVINCIHOME/grapheditor/grapheditor*). Furthermore, environment variable $GRAPHEDITOR_DIRECTORY can be used to specify the directory with the editor explicitly to let *daVinci* look for the editor at a different location, but usually there is no need to do so. The graph editor can also be connected to *daVinci* at runtime by using menu File/Connect Application....

Option *-help*

Displays a help message to inform about the command-line syntax. *daVinci* will not start with this option.

Option *-init <filename>*

With this option you can specify a file containing initial API commands. These commands are executed immediately after starting *daVinci*. You can use all API commands, except those of the categories Multi, Application Menu and Drag & Drop. The answers of these commands, you usually get, if you send them via the API, are ignored, if no application is connected, otherwise they are send to the API answers pipe. Use this option carefully, if you've connected an application on command-line via options *-pipe* or *-startappl* or *-graphedit*, because this could result in a strange behaviour of the application.

The syntax of the file is just a list of API commands, defined for daVinci_commands.

Option *-log <filename>*

In the file specified with this option, *daVinci* will log all API commands and answers. Every command is preceded by the word 'Command: ' and every answer is preceded by the word 'Answer: '. This option is very usefull for debugging purposes. You can test this option by looking at the grapheditor communication. Start *daVinci* like this: *daVinci -graphedit -log /tmp/daVinci.log* and have a look at the file */tmp/daVinci.log*. Try the command *tail -f /tmp/daVinci.log* to see the communication continously or *grep 'Command:' /tmp/daVinci.log | tail -f* to see only the API commands and appropriately for API answers.

Option *-nocache*

Switches the pixmap cache off. This is useful in case of problems with not enough memory. *daVinci* will crash with a *X Error: BadAlloc* message as soon as no more memory is available.

Option *-nopopup*

Switches the pop-up menu off which will usually appear by pressing the right mouse button. This behaviour might be inappropriated for some window managers, so the pop-up menu can be disabled by using this option.

Option *-pipe*

This option should be used if *daVinci* is started by an application program. Read the API reference for details about applications communicating with *daVinci*. By using this option, the *daVinci* API is enabled and most of the entries of the File menu are deactivated to guarantee that the user will not be able to remove a graph sent by an application for visualization.

Option *-speedy*

Accelerates the drawing performance for very slow computers. This will only drop the arrows of edges in the graph visualization at the moment. For todays fast computers, drawing the arrows is not a problem, so using this option is usually not required.

Option *-startappl <Directory> <Filename/Options>*

Connects the application, specified by *Filename* and located in the given *Directory*, at initialization time to the *daVinci* API. Note that there is a space required between the directory and filename. If options are needed to start the application, then the filename has to be quoted in the shell together with the options (e.g. *-startappl /bin 'testappl -opt1 arg1 arg2 -opt2'* to start application *testappl* in directory */bin* with the given options). An application can also be connected at runtime by using menu File/Connect Application....

Option *-verify*

(Only available in *daVinci* V2.0.2 or higher.) This option is intended for debugging. By using this option, *daVinci* verifies some internal data structures after updating a graph with API command graph(update(...)). You should start *daVinci* with this option if you get strange effects or bugs after graph updates. The verification costs some time, so usually the function is switched off for performance reasons and must be explicitly switched on by the user.

Option *-version*

Displays the current version of *daVinci* on command-line. *daVinci* will not start with this option.

Option *-window_api*

In former versions of *daVinci*, the API supported only the multi-graph feature, but not the multi-window feature. Starting with *daVinci* V2.1 you can set the API into the new Window-API mode by setting this option on command-line at startup. You can't change this behaviour at runtime. See API command category Multi for more informations.

Environment Variables

$DAVINCIHOME

This is the most important environment variable which specifies the location of the *daVinci* distribution on your computer, i.e. the absolute file name of the topmost directory *daVinci_V2.1*. To get proper results, you need to set this variable in the shell before using the system. *daVinci* needs to know this location to find resources, e.g. the online documentation or the graph editor.

So for example, if you have installed *daVinci* in your home directory */home/me*, then you have to set the environment variable $DAVINCIHOME to */home/me/daVinci_V2.1*. *daVinci* will use the current directory, if this environment variable is undefined.

Note: Be careful with moving the *daVinci* distribution to another location on your filesystem later. You can do so (by setting $DAVINCIHOME to the new location), but users may have problems accessing the help system afterwards. The reason is that *daVinci* stores the current location of the online documentation when the user saves the options with menu Options/Save Options, so these users will always look for the documentation at the specified former location. The current setting can be seen and modified with the General Settings dialog, available after using menu Options/Save Options. So if you have to move the *daVinci* distribution, make sure that all users are updating or removing their saved options (stored in file *~/.daVinci*). Otherwise you can leave a symbolic link at the former *daVinci* location for backward compatibility.

$DAVINCI_ICONDIR

This environment variable specifies all directories where *daVinci* will look for files containing icon images in X11 bitmap format (*.xbm*). Icons are used in *daVinci* to visualize particular nodes of a graph (refer to term representation) and for the icon bar of the base window which can be extended by application specific icons with API command app_menu(create_icons(...)).

By default, *daVinci* will look for an icon in directory *$DAVINCIHOME/icons* first. One can specify additional directories that should be searched afterwards by setting this environment variable. The single directories are separated with a colon, so for example to define two additional icon directories */home/me/myicons* and */home/mycolleague/his_icons*, simply set $DAVINCI_ICONDIR to */home/me/myicons:/home/mycolleague/his_icons*.

$GRAPHEDITOR_DIRECTORY

By default, *daVinci* will look for the graph editor application in directory *$DAVINCIHOME/grapheditor*, if started with option -graphedit. To look for the editor at another location, one can set environment variable $GRAPHEDITOR_DIRECTORY to point to a different directory. But usually, it is not recommended to manipulate the files in the *daVinci* distribution this way.

Bibliography

[BET93]
G. DiBattista, P. Eades, R. Tamassia, I.G. Tollis:
"Algorithms for Drawing Graphs: an Annotated Bibliography"; Technical Report; Brown University; 1993
ftp://ftp.cs.brown.edu/pub/papers/compgeo/gdbiblio.ps.gz

[BS89]
B. Birgisson, G. Shannon:
"GraphView: An Extensible Interactive Platform for Manipulating and Displaying Graphs"; Technical Report No. 295; Computer Science Department; Indiana University; Bloomington, Indiana; 1989

[Com92]
C. L. Combs:
"An Automated Drawing System for Directed Graphs"; Master's Thesis; Department of Computer Science and Engineering; Auburn University; Alabama; 1992

[Frö97]
M. Fröhlich:
"Inkrementelles Graphlayout im Visualisierungssystem *daVinci*" (available only in german language, english title would be: Incremental Graphlayout in the Visualization System *daVinci*); PhD Thesis; Department of Computer Science; Universität Bremen; November 1997
ftp://ftp.tzi.de/tzi/biss/daVinci/papers/fro97.ps.gz (~2.4MB)

[FW93a]
M. Fröhlich, M. Werner:
"*daVinci* V1.1 User Manual"; Universität Bremen; August 1993

[FW93b]
M. Fröhlich, M. Werner:
"*daVinci* V1.2 User Manual"; Universität Bremen; October 1993

[FW94a]
M. Fröhlich, M. Werner:
"*daVinci* V1.3 User Manual"; Universität Bremen; March 1994

[FW94b]
M. Fröhlich, M. Werner:
"The Graph Visualization System *daVinci* - A User Interface for Applications"; Technical Report No. 5/94; Department of Computer Science; Universität Bremen; September 1994
ftp://ftp.tzi.de/tzi/biss/daVinci/papers/techrep0594.ps.gz

[FW94c]
M. Fröhlich, M. Werner:
"*daVinci* V1.4 User Manual"; Universität Bremen; November 1994

[FW95]
M. Fröhlich, M. Werner:
"Demonstration of the interactive Graph Visualization System *daVinci*"; in: R. Tamassia, I. Tollis: "Proceedings of DIMACS Workshop on Graph Drawing '94, Princeton (USA) 1994"; LNCS No. 894; Springer Verlag; 1995
ftp://ftp.tzi.de/tzi/biss/daVinci/papers/lncs.ps.gz

[FW96]
M. Fröhlich, M. Werner:
"*daVinci* V2.0.x Online Documentation"; Universität Bremen; June 1996
http://www.tzi.de/~davinci/doc_V2.0/

[GKNV93]
E. R. Gansner, E. Koutsofios, S. C. North, K. P. Vo:
"A Technique for Drawing Directed Graphs"; IEEE Transactions on Software Engineering; Vol. 19, No. 3, pp. 214-230; March 1993

[Hen92]
T. R. Henry:
"Interactive Graph Layout: The Exploration of Large Graphs"; Ph.D. Thesis; Technical Report 92-03; University of Arizona; 1992
ftp://cs.arizona.edu/reports/1992/TR92-03.ps

[Him95]
M. Himsolt:
"GraphEd: A Graphical Platform for the Implementation of Graph Algorithms"; in: R. Tamassia, I. Tollis: "Proceedings of DIMACS Workshop on Graph Drawing '94, Princeton (USA) 1994"; LNCS No. 894; Springer Verlag; 1995

[Juu94]
A. Juutistenaho:
"Linear Time Algorithms for Layout of Generalized Trees"; Technical Report No. A-1994-6; Departement of Computer Science, University of Tampere, Finland; 1994

[OSF92]
Open Software Foundation:
"OSF/Motif Style Guide, Revision 1.2"; 1992

[San95]
G. Sander:
"Graph Layout through the VCG Tool"; in: R. Tamassia, I. Tollis: "Proceedings of DIMACS Workshop on Graph Drawing '94, Princeton (USA) 1994"; LNCS No. 894; Springer Verlag; 1995

[SM91]
K. Sugiyama, K. Misue:
"Visualization of Structural Information: Automatic Drawing of Compound Digraphs"; IEEE Transactions on Systems, Man, and Cybernetics; Vol. SMC-21; No. 4; pp. 876 - 892;

July/August 1991

[STT81]
K. Sugiyama, S. Tagawa, M. Toda:
"Methods for Visual Understanding of Hierarchical System Structures"; IEEE Transactions on Systems, Man, and Cybernetics; Vol. SMC-11; No. 2; pp. 109 - 125; February 1981

[Wer98]
M. Werner:
"*daVinci* V2.1.x Online Documentation"; Universität Bremen; June 1998
http://www.tzi.de/~davinci/doc_V2.1/

Overview -> *daVinci* in the Internet

daVinci - Online Documentation V2.1

daVinci in the Internet

This chapter covers information about accessing important *daVinci* resources in the Internet. Clicking on the headlines will show the corresponding resource in a second Netscape window. Keep the second window visible when you want to get another resource on this page! The following resources are available:

- **Sending E-Mail to the Developers**
- **Visiting the *daVinci* WWW Server**
- **Subscribing to the Mailing List**
- **Getting the latest Bugreport**
- **Getting the latest Questions & Answers**
- **Downloading *daVinci***

Sending E-Mail to the Developers

When you have problems or questions, do not hesitate to write an e-mail to the *daVinci* developers. Before doing so, please consult the latest bugreport and this online documentation (especially chapter Questions & Answers) to see if there is already a solution for your problem. The *daVinci* project is also glad to be informed about applications of this software, suggestions for new features, or even criticism.

Visiting the daVinci *WWW Server*

To get more information about *daVinci*, you should visit the WWW server of the *daVinci* project at Universität Bremen (Center for Computing Technology, TZI) from time to time. There you can read more about:

- News and announcements
- Release notes
- Procedures for downloading the software
- Sites where *daVinci* is installed
- Selected real world applications
- ...

Subscribing to the Mailing List

The *daVinci* project maintains a mailing list to inform users about announcements and other news related to *daVinci*. You are automatically asked to subscribe to this list with the first use of the software. If you want to subscribe manually, please write an e-mail to the *daVinci* project.

Getting the latest Bugreport

The latest bugreport contains information about all known bugs and problems in *daVinci*, their current state and available fixes or work-arounds. The developers will update this report as soon as a new bug is found or a fix is released. When you find a malfunction in *daVinci*, please consult this report first to see if this is an already known problem.

Getting the latest Questions & Answers

The latest version of the *Questions & Answers* document is available on the *daVinci* WWW server. You have a local copy of this document which may not reflect the current state, so it might be a good idea to take a look into the latest version in case of problems.

Downloading daVinci

To download *daVinci*, it is recommended to use this form to get additional information about the distributions. You can also download *daVinci* directly from the ftp-server at Universität Bremen (Center for Computing Technology, TZI) (*ftp://ftp.tzi.de/tzi/biss/daVinci*). This is the place where you can find the latest version. Other servers (e.g. *ftp://ftp.wustl.edu/graphics/graphics/packages/daVinci* in the USA) are mirroring *daVinci*, but it usually takes some time to see a new release on one of the mirrors.

---

daVinci V2.1 Online Documentation - Page update: June 15, 1998

daVinci - Online Documentation - Title Page

Table of Contents

daVinci - Online Documentation - Title Page 1
daVinci - Online Documentation 2
Overview 2
daVinci - General Information 3
General Information 3
daVinci - Introduction 4
Introduction 4
What you should read next: 4
daVinci - Using the Online Documentation 6
Using the Online Documentation 6
Using the Online Documentation together with *daVinci* 6
How to get help in *daVinci* 7
Roadmap to the Online Documentation 7
Features of the Online Documentation 8
daVinci - Concepts 9
daVinci's Concepts 9
Graphs 9
Applications using *daVinci's* API 10
Graph Layout 11
Visualization definition 12
Fine Tuning of a Layout 12
Graph Status 14
Interactions 14
Multi-Graph / Multi-View 14
Drawing Technology and Pixmap Caching 15
Drag and Drop 15
daVinci - License 17
daVinci V2.1.x License 17
daVinci - Platform Requirements and Installation 19
Platform Requirements and Installation 19
Platform Requirements 19
Installation 20
daVinci - User Interface 22
User Interface 22
daVinci - Using the Mouse 23
Using the Mouse 23
BSelect (Left Mouse Button) 23
BTransfer (Middle Mouse Button) 23
BMenu (Right Mouse Button) 24
daVinci - Using the Keyboard 25
Using the Keyboard 25
Keyboard Usage in the Base Window 25
Mnemonic Keys to access Menus: 25
Accelerator Keys to call Menu Operations directly 26
Cursor Keys to navigate in a Graph 26
Keyboard Usage in Dialog Windows 26

Choosing the actual Item: 26
Text Entry Fields: 26
Lists: 27
Option Buttons 27
Sliders 27
Radio Buttons 28
Check Buttons 28
Push Buttons 28
daVinci - Base Window 29
Base Window 29
daVinci - Pop Up Menu 31
Pop up Menu 31
daVinci - File Menu 33
File Menu 33
File and Printer Management 33
Application Management 34
Closing Actions 34
daVinci - File Browser 35
File Browser to open Files 35
Dialog Items of the File Browser 36
daVinci - File Browser 38
File Browser to save Files 38
Dialog Items of the File Browser 39
daVinci - Print Dialog 41
Print Dialog Window 41
Dialog Items of the Print Window 42
daVinci - File Browser 44
File Browser to specify PostScript Files 44
Dialog Items of the File Browser 45
daVinci - File Browser 46
File Browser to connect Applications 46
Dialog Items of the File Browser 47
daVinci - Edit Menu 48
Edit Menu 48
daVinci - View Menu 49
View Menu 49
Multi-View Management 49
Scaling 50
Getting Informations 50
daVinci - Scale Dialog 51
Scale Dialog Window 51
Dialog Items of the Scale Window 51
daVinci - Graph Info Dialog 53
Graph Info Dialog Window 53
Dialog Items of the Graph Info Dialog 53
daVinci - daVinci Info Dialog 55
The daVinci Info Dialog Window 55
Dialog Items of the daVinci Info Dialog 55
daVinci - Navigation Menu 56
Navigation Menu 56

Node Selection Operations . . . 56
Search Operations . . . 56
daVinci - Navigator Dialog . . . 58
Navigator Dialog Window . . . 58
Dialog Items of the Navigator Window . . . 59
daVinci - Find Dialog . . . 60
Find Dialog Window . . . 60
Dialog Items of the Find Window . . . 60
daVinci - Abstraction Menu . . . 62
Abstraction Menu . . . 62
Subgraph Abstraction . . . 62
Edge Abstraction . . . 63
daVinci - Layout Menu . . . 64
Layout Menu . . . 64
Layout Optimization . . . 64
Controlling the Layout Orientation . . . 65
daVinci - Options Menu . . . 66
Options Menu . . . 66
Changing the Options . . . 66
Saving the Options . . . 66
daVinci - Layout Algorithm Dialog . . . 67
Layout Algorithm Dialog Window . . . 67
Dialog Items of the Layout Algorithm Window . . . 68
daVinci - Layout Dimensions Window . . . 69
Layout Dimensions Dialog Window . . . 69
Dialog Items of the Layout Dimensions Window . . . 70
daVinci - General Settings Window . . . 71
General Settings Dialog Window . . . 71
Dialog Items of the General Settings Window . . . 72
daVinci - Help Menu . . . 74
Help Menu . . . 74
Help Facilities of the Online Documentation . . . 74
daVinci - Tutorial . . . 76
Tutorial . . . 76
daVinci - Before You Begin . . . 77
Before you begin . . . 77
daVinci - Tutorial: Managing Files . . . 78
Section 1: Managing Files and Basic Tasks . . . 78
Loading a Graph from File . . . 78
Starting the Graph Layout Algorithm . . . 79
Saving the Status of a Graph . . . 79
Printing a Graph Visualization . . . 80
daVinci - Tutorial: Graph Layout . . . 82
Section 2: Optimizing the Graph Layout . . . 82
Setting the Layout Dimensions . . . 82
Horizontal Fine Tuning . . . 82
Vertical Fine Tuning . . . 83
daVinci - Tutorial: Interactions . . . 85
Section 3: About Scaling, Abstractions, and Navigation . . . 85
Get an Overview with Scaling . . . 85

Using Abstractions to simplify a Layout . . . 85
Navigation Features to browse in a Graph . . . 86
daVinci - Tutorial: Multi-Graph . . . 88
Section 4: Multi-Graph and Multi-View . . . 88
Multi-View: Using many Views for one Graph . . . 88
Working with a Survey View . . . 88
Multi-Graph: Visualizing different Graphs simultaneously . . . 89
daVinci - Tutorial: Graph Editor . . . 90
Section 5: Using the Graph Editor . . . 90
Starting the Graph Editor . . . 90
Building a new Graph from Scratch . . . 91
Modifying existing Graphs . . . 91
Editing several Graphs . . . 92
Using Drag and Drop . . . 92
daVinci - Questions & Answers . . . 95
Questions & Answers . . . 95
Overview: . . . 95
Availability: . . . 96
License: . . . 97
System Features: . . . 97
Graph Layout: . . . 97
Term Representation (*daVinci's* external Graph Format): . . . 98
Application Programmer Interface (API): . . . 99
Graph Editor Application: . . . 99
Problems: . . . 100
Help System / Online Documentation: . . . 101
More Information: . . . 101
daVinci - New Features . . . 103
New Features in *daVinci* V2.1.x . . . 103
Fully incremental graph layout . . . 103
Subgraph abstractions using incremental layout . . . 104
Completely rewritten, much faster graph layout algorithm . . . 104
New automatic layout functions . . . 104
Fine-Tuning for all layout orientations . . . 104
Visualization rules (API) . . . 104
Pop-Up menus for nodes and edges (API) . . . 105
Drag and Drop for nodes and edges (API) . . . 105
All open windows accessible via API commands . . . 105
Some more enhancements to the API . . . 106
New command-line options . . . 106
New Attributes for Nodes and Edges . . . 106
grapheditor supports all new API features, including drag and drop . . . 107
Several bug fixes and reduced memory footprint . . . 107
Shortcomings in *daVinci* V2.1.x . . . 107
daVinci - Reference . . . 108
Reference . . . 108
daVinci - Term Representation . . . 109
daVinci Term Representation . . . 109
Structure of Term Representation . . . 109
Multi-Edges and Self-Edges . . . 110

Node and Edge Types 111
Attributes 111
Supported Node Attributes 112
Supported Edge Attributes 114
Example of a Term Representation 114
daVinci API 117
daVinci API 117
daVinci API - Introduction 118
daVinci API - Introduction 118
daVinci API - How to Connect an Application 120
daVinci API - How to Connect an Application 120
Choose which way to go 120
Either *daVinci* connects the Application... 121
... or the Application connects *daVinci* 121
Transmitting Commands 121
daVinci API - Programmers Guide 123
daVinci API - Programmers Guide 123
Commands of the API 123
Answers from the API 125
Important Notes about the API 125
daVinci API - Transition Guide 128
daVinci API - Transition Guide 128
Term Representation of Graphs 128
New API Syntax 128
Transform V1.x to V2.x API Commands 128
daVinci API - Overview of Commands 132
daVinci API Definition - Overview of Commands 132
daVinci API - Syntax 136
daVinci API - Syntax 136
daVinci API - Parameters 137
daVinci API Definition - Parameters 137
daVinci API - Graph Commands 140
daVinci API Definition - Graph Commands 140
graph(new(graphterm)) 140
graph(new_placed(graphterm)) 140
graph(update(node_updates,edge_updates)) graph(update(mixed_updates)) . . . 140
graph(change_attr(attr_changes)) 141
graph(update_and_change_attr(node_updates,edge_updates,attr_changes))
graph(update_and_change_attr(mixed_updates,attr_changes)) 142
graph(change_type(type_changes)) 142
daVinci API - Multi Commands 143
daVinci API Definition - Multi Commands 143
Multi-Graph Background: 143
Multi-View Background: 144
Multi-Window Background: 145
Documentation of the Category Commands: 145
multi(new_context) 145
multi(open_context(context_id)) 146
multi(set_context(context_id)) multi(set_context(context_id,window_id)) . . . 146

daVinci API - Menu Commands 147
daVinci API Definition - Menu Commands 147
File Menu Category: 147
menu(file(new)) 147
menu(file(open_graph(filename))) 147
menu(file(open_graph_placed(filename))) 148
menu(file(open_status(filename))) 148
menu(file(save_graph(filename))) 148
menu(file(save_status(filename))) 148
menu(file(print(filename))) 148
menu(file(print)) 148
menu(file(close)) 148
menu(file(exit)) 149
View Menu Category: 149
menu(view(open_new_view)) 149
menu(view(open_survey_view)) 149
menu(view(full_scale)) 150
menu(view(fit_scale_to_window)) 150
menu(view(scale(integer))) 150
menu(view(scale)) 150
menu(view(graph_info)) 150
menu(view(daVinci_info)) 150
Navigation Menu Category: 150
menu(navigation(select_parents(node_ids))) 151
menu(navigation(select_siblings(node_ids))) 151
menu(navigation(select_children(node_ids))) 151
menu(navigation(navigator(node_id,direction,boolean))) 151
menu(navigation(navigator)) 152
menu(navigation(find(string,boolean,boolean))) 152
menu(navigation(find)) 152
Abstraction Menu Category: 152
menu(abstraction(hide_subgraph(node_ids))) 152
menu(abstraction(show_subgraph(node_ids))) 152
menu(abstraction(restore_all_subgraphs)) 153
menu(abstraction(hide_edges(node_ids))) 153
menu(abstraction(show_edges(node_ids))) 153
menu(abstraction(restore_all_edges)) 153
Layout Menu Category: 153
menu(layout(improve_all)) 153
menu(layout(improve_edges)) 153
menu(layout(improve_selected)) 153
menu(layout(improve_nodes(node_ids))) 154
menu(layout(improve_spacing)) 154
menu(layout(orientation(orientation))) 154
daVinci API - Application Menu Commands 155
daVinci API Definition - Application Menu Commands 155
Why attaching Menus and Icons to the *daVinci* User Interface? 155
Background of Application Menus and Icons 155
A closer Integration of I/O by interposing the File Menu Events 156
Recommended Actions on File Events 157

Documentation of the Category Commands: 158
app_menu(create_menus(menus)) 158
app_menu(create_icons(icons)) 159
app_menu(activate_menus(menu_ids)) 160
app_menu(activate_icons(icon_ids)) 160
app_menu(control_file_events) 160
daVinci API - Set Commands 161
daVinci API Definition - Set Commands 161
set(layout_accuracy(integer)) 161
set(keep_nodes_at_levels(boolean)) 161
set(font_size(integer)) 161
set(gap_width(integer)) 161
set(gap_height(integer)) 161
set(multi_edge_gap(integer)) 162
set(self_edge_radius(integer)) 162
set(scrolling_on_selection(boolean)) 162
set(animation_speed(integer)) 162
set(no_cache(boolean)) 162
set(rules_first(boolean)) 163
daVinci API - Window Commands 164
daVinci API Definition - Window Commands 164
window(title(string)) 164
window(show_message(string)) 164
window(show_status(string)) 164
window(position(integer,integer)) 164
window(size(integer,integer)) 164
window(raise) 164
window(iconify) 165
window(deiconify) 165
window(activate) 165
window(deactivate) 165
window(file_browser(boolean,string,string,string,string,btypes,boolean)) 165
daVinci API - Tcl/Tk Commands 167
daVinci API Definition - Tcl/Tk Commands 167
The *daVinci* Tcl/Tk Interface 167
The *daVinci* command in Tcl 167
Tcl command *daVinci tcl_answer string* 168
Tcl command *daVinci command api-command* 168
Documentation of the Category Commands: 168
tcl(eval(string)) 168
tcl(eval_file(filename)) 169
daVinci API - Special Commands 170
daVinci API Definition - Special Commands 170
special(select_nodes(node_ids)) 170
special(select_edge(edge_id)) 170
special(focus_node(node_id)) 170
special(focus_node_animated(node_id)) 170
special(show_url(string)) 170
daVinci API - Visual Commands 172
daVinci API Definition - Visual Commands 172

Documentation of the Category Commands: 172
visual(new_rules(visual_rules)) 172
visual(add_rules(visual_rules)) 172
daVinci API - Drag and Drop Commands 173
daVinci API Definition - Drag and Drop Commands 173
Possible Drag and Drop scenarios 173
Documentation of the Category Commands: 174
drag_and_drop(dragging_on) 174
drag_and_drop(dragging_off) 174
drag_and_drop(new_node_at_coord(node_update)) 174
drag_and_drop(new_edge_and_node_at_coord(node_update,edge_update)) . . . 175
daVinci API - Overview of Answers 176
daVinci API Definition - Overview of Answers 176
daVinci API - Answers 178
daVinci API Definition - Answers 178
Answers at Initialization Time: 178
Answers for confirming API Commands: 178
Answers for Event Notification: 179
Answers in Multi-Graph Mode: 179
Answers in Multi-Window Mode: 180
Documentation of all API Answers: 180
ok 180
communication_error(string) 180
node_selections_labels(node_ids) 180
node_double_click 181
edge_selection_label(edge_id) 181
edge_selection_labels(node_id,node_id) 181
edge_double_click 181
menu_selection(menu_id) 181
icon_selection(icon_id) 182
popup_selection_node(node_id,menu_id) 182
popup_selection_edge(edge_id,menu_id) 182
context(context_id) 182
context_window(context_id,window_id) 182
open_window 183
close_window(window_id) 183
tcl_answer(string) 183
browser_answer(string,string) 184
create_node 184
create_node_and_edge(node_id) 184
create_edge(node_id,node_id) 184
drop_node(node_id,context_id,window_id,node_id) 184
disconnect 185
close 185
quit 185
daVinci - PostScript Interface 187
daVinci PostScript Interface 187
Notes about the *daVinci* PostScript interface: 187
daVinci - Graph Editor Application 188
Graph Editor Application 188

Starting the Graph Editor . . . 188
Using the Graph Editor . . . 188
Menu Edit/Node . . . 189
Menu Edit/Edge . . . 190
Menu Edit/Toggle Drag & Drop . . . 190
Node Popup-Menu . . . 191
Edge Popup-Menu . . . 192
File Operations . . . 192
Graph Editor Icons on the Icon Bar . . . 193
Drag and Drop in the grapheditor . . . 193
Observing the communication between *daVinci* and the grapheditor . . . 193
daVinci - Node Attribute Dialog . . . 195
Node Attribute Dialog Window . . . 195
Dialog Items of the Node Attribute Window . . . 197
daVinci - Color Editor . . . 199
Color Editor Window . . . 199
Dialog Items of the Color Editor Window . . . 199
daVinci - Naming Scheme Dialog . . . 201
Naming Scheme Dialog Window . . . 201
Dialog Items of the Naming Scheme Window . . . 201
daVinci - Edge Attribute Dialog . . . 202
Edge Attribute Dialog Window . . . 202
Dialog Items of the Edge Attribute Window . . . 203
daVinci - Available Tools . . . 204
Available Tools for *daVinci* . . . 204
checkterm . . . 204
formatterm . . . 204
flattenterm . . . 204
daVinci - Command-Line Options . . . 206
Command-Line Options and Environment Variables . . . 206
Command-Line Options . . . 206
Environment Variables . . . 208
daVinci - Bibliography . . . 210
Bibliography . . . 210
daVinci - daVinci in the Internet . . . 213
daVinci in the Internet . . . 213
Sending E-Mail to the Developers . . . 213
Visiting the *daVinci* WWW Server . . . 213
Subscribing to the Mailing List . . . 214
Getting the latest Bugreport . . . 214
Getting the latest Questions & Answers . . . 214
Downloading *daVinci* . . . 214